



Driver Manual

CIF Device Driver for IntervalZero/VenturCom/Ardence RTX

Edition: 5

Language: English (EN)

Hilscher Gesellschaft für Systemautomation mbH

Rheinstraße 15
D-65795 Hattersheim
Germany

Tel. +49 (0) 6190 / 99070

Fax. +49 (0) 6190 / 990750

Sales: +49 (0) 6190 / 99070

Hotline and Support: +49 (0) 6190 / 990799

Sales email: sales@hilscher.com

Hotline and Support email: hotline@hilscher.com

Web: www.hilscher.com

Index	Date	Version	Chapter	Revision
1	27.08.01	1	All	created
2	04.07.02	2	1.3	- Support for COM modules and standard protocol CIF cards included
3	21.05.03	3	All	- New parameter in function DevInitBoard() and DevGetBoardInfo()
4	29.11.05	4	All	- Information about RTX 6.x included - Firmware list extended - CIF 100 support removed
5	17.07.09	5	1	- Support for RTX Version 8.1 and 2009 included

Although this program has been developed with great care and intensively tested, Hilscher Gesellschaft für Systemautomation mbH cannot guarantee the suitability of this program for any purpose not confirmed by us in writing.

Guarantee claims shall be limited to the right to require rectification. Liability for any damages which may have arisen from the use of this program or its documentation shall be limited to cases of intent.

We reserve the right to modify our products and their specifications at any time in as far as this contributes to technical progress. The version of the manual supplied with the program applies.

Please notice that software and hardware names, used in this manual, are normally protected by trademarks or patents of corresponding companies.

1	INTRODUCTION	5
1.1	Implementation	5
1.2	Requirements, Features and Limitations	5
1.3	Supported CIF Cards, Fieldbus Systems and Protocols	6
1.3.1	PROFIBUS	6
1.3.2	DeviceNet	7
1.3.3	CANopen	8
1.3.4	InterBus	9
1.3.5	AS-Interface	9
1.3.6	ControlNet	10
1.3.7	Serial Protocol	11
2	INSTALLATION	12
3	FIELDBUS CONFIGURATION DETAILS	13
3.1	Module CIFConfiguration	14
3.2	Fieldbus Startup Behaviour	15
3.3	Supported Handshake Modes	16
4	DRIVER FUNCTIONS	17
4.1	Initialization Process	17
4.2	Extended Configuration Checking	17
4.3	New I/O Data Exchange Function	18
5	PROGRAMMING	19
5.1	Standard API	19
5.1.1	DevInitBoard()	20
5.1.2	DevGetBoardInfo()	21
5.2	Additional API Functions	22
5.2.1	DevExchangePLCData	22
5.2.2	DevIsPLCDataReady	23
5.2.3	DevGetInfo	24
5.3	API Files	25
5.4	Example Program CIFRTXTest	25
6	ADDITIONAL INFORMATION	26
7	ERROR NUMBERS	27
7.1	List of Error Numbers	27

7.2 Additional Error Information..... 30

1 Introduction

IntervalZero (formerly VenturCom/Ardence) RTX is a **Real Time eXtension** for the Windows NT 4, Windows 2000 and Windows XP environment.

The Hilscher RTX driver is a RTDLL (CIFRTXDriver.rtdll) which supports CIF30 (ISA), and CIF50 (PCI) cards. Please refer to section '**Supported CIF Cards and Fieldbus Systems**' which fieldbus systems and devices are currently supported by the driver.

1.1 Implementation

The RTX driver API is based on the existing CIF Device Driver API for Windows.

The driver offers transparent access to the different devices. Therefore it hides the functionality's for ISA and PCI cards. The cards can be selected by the original 'Device Type' string and a board number between 0 to 3.

For better performance and easier access to the hardware, the standard API functions are extended by new parameters. These parameters are used to pass a driver and device handle down to the driver.

1.2 Requirements, Features and Limitations

Requirements:

- VenturCom RTX version 5.x or 6.x
Microsoft Visual Studio V6.0 SP 3 or higher
- OR ---
- IntervalZero RTX version 8.1 or 2009
Microsoft Visual Studio .NET 2003 or higher

Features:

- Automatic hardware detection for ISA and PCI hardware
- Device selection by name and board number
- All devices can be used simultaneously (ISA, PCI)
- Up to four devices of the same type (ISA, PCI) possible
- Automatic firmware comparison and download during startup possible
- Automatic configuration comparison and download during startup possible

Limitations:

- Only one RTX process can use the driver at the same time
- Only device polling is supported

1.3 Supported CIF Cards, Fieldbus Systems and Protocols

The following table shows the currently supported CIF cards fieldbus systems and protocols.

1.3.1 PROFIBUS

Fieldbus system	Device Type	Firmware Name	Firmware File Name
PROFIBUS	Master devices		
	CPS1-DPM	DPM CPS1-DPM	DPM.X52
	COM-DPM	DPM COM-DPM	DPM.H58
	COM-CA-DPM	DPM COMCADPM	DPMCOMCA.E01
	COM-PB	PB-COMBICOM-PB	PB.H5C
	CIF30PB	PB-COMBICIF30PB	PB.H32
	CIF30FMS	PB-FMS CIF30FMS	FMS.H33
	CIF30DPM	DPM CIF30DPM	DPM.H33
	CIF104DP	DPM CIF104DP	DPM.H73
	C104DPMR	DPM C104DPMR	DPM.H7B
	C104-PB	PB-COMBIC104-PB	PB.H7C
	C104PBR	PB-COMBIC104PBR	PB.H7B
	C104PBE	PB-COMBIC104PBE	PB.H7U
	CIF104P-PB	PB-COMBIC104P-PB	PBC104P.E01
	CIF50-PB	PB-COMBICIF50-PB	PB.H62
	CIF80-PB	PB-COMBICIF80-PB	PBC80.E01
	PMC-PB	PB-COMBIPMC-PB	PBPMC.E01
	Slave devices		
	COM-DPS	DPS COM-DPS	DPS.H57
	COM-CA-DPS	DPS COMCDPS	DPSCOMC.E13
	CPS1-DPS	DPS CPS1-DPS	DPS.X51
	C104-DPS	DPS C104-DPS	DPS.H74
	C104-DPS	DPS C104-DPS	DPS.H7F
	C104-DPS	DPS C104-DPS	DPS.H7P
	CIF30DPS	DPS CIF30DPS	DPS.H34
	CIF50DPS	DPS CIF50DPS	DPS.H6D

1.3.2 DeviceNet

Fieldbus system	Device Type	Firmware Name	Firmware File Name
DeviceNet	Master devices		
	COM-DNM	DNM COM-DNM	DNM.H5F
	COM-CA-DNM	DNM COMCDNM	DNMCOMC.E03
	COMCDNM	DNM COMCDNM	DNMCOMC.E03
	CIF30DNM	DNM CIF30DNM	DNM.H38
	C104-DNM	DNM C104-DNM	DNM.H7L
	C104PDNM	DNM C104PDNM	DNM104P.E03
	CIF50DNM	DNM CIF50DNM	DNM.H68
	CIF50DNM	DNM CIF50DNM	DNM CIF50DNM
	CIF80DNM	DNM CIF80DNM	DNMC80.E03
	PMC	DNM PMC	DNMPMC.E03
	PMCDNM	DNM PMCDNM	DNMPMC.E03
	Slave devices		
	COM-DNS	DNS COM-DNS	DNS.H5K
	COM-CA-DNS	DNS COMCDNS	DNSCOMC.E12
	C104-DNS	DNS C104-DNS	DNS.H7G
	CIF30DNS	DNS CIF30DNS	DNS.H39
	CIF50DNS	DNS CIF50DNS	DNS.H6H

1.3.3 CANopen

Fieldbus system	Device Type	Firmware Name	Firmware File Name
CANopen	Master devices		
	COM-COM	CANopen COM-COM	COM.H5E
	COM-CA-COM	CANopen COMCCOM	COMCOMC.E02
	COMCCOM	CANopen COMCCOM	COMCOMC.E02
	CIF30CAN	CANopen CIF30CAN	COM.H36
	C104-CAN	CANOpen C104-CAN	COM.H7N
	C104-CAN	CANOpen C104-CAN	COM.H7X
	C104PCOM	CANOpen C104PCOM	COM104P.E02
	CIF50CAN	CANopen CIF50CAN	COM.H66
	CIF50CAN	CANopen CIF50CAN	COMC50.E2E
	CIF80COM	CANopen CIF80COM	COM.E02
	PMCCOM	CANopen PMCCOM	COMPMC.E02
	Slave devices		
	COM-COS	COS COM-COS	COS.H5I
	C104-COS	COS C104-COS	COS.H7E
	CIF30COS	COS CIF30COS	COS.H3C
	CIF50COS	COS CIF50COS	COS.H6F

1.3.4 InterBus

Fieldbus system	Device Type	Firmware Name	Firmware File Name
InterBus	Master devices		
	COM-IBM	IBM COM-IBM	IBM.H53
	COM-IBM	IBM COM-IBM	IBM.H5P
	CIF30IBM	IBM CIF30IBM	IBM.H35
	C104IBM	IBM C104IBM	IBM.H78
	C104IBM	IBM C104IBM	IBM.H7Y
	C50IBM	IBM C50IBM	IBM.H69
	C50IBM	IBM C50IBM	IBM.H6L
	C100IBM	COM C100COM	COM.C50
	Slave devices		
	COM-IBS	IBS COM-IBS	IBS.H5H
	C-104IBS	IBS C-104IBS	IBS.H7A
	C4IBSIDR	IBS C4IBSIDR	IBS.H7I
	C104IBSR	IBS C104IBSR	IBS.H7O
	CIF30IBS	IBS CIF30IBS	IBS.H3A

1.3.5 AS-Interface

Fieldbus system	Device Type	Firmware Name	Firmware File Name
ASI	Master devices		
	COM-ASM	ASIM COM-ASM	ASIM.H5L
	COM-ASM	ASIM COM-ASM	ASIMCOM.E06
	COMC-ASM	ASIM COMC-ASM	ASIMCOMC.E28
	COMB-ASM	ASIM COMB-ASM	ASIMCOMB.E28
	C104-ASM	ASIM C104-ASM	ASIM.H7Q
	C104-ASM	ASIM C104-ASM	ASIMC104.E06
	CIF50ASM	ASIM CIF50ASM	ASIM.H6I
	C50-ASM	ASIM C50-ASM	ASIMC50.E06

1.3.6 ControlNet

Fieldbus system	Device Type	Firmware Name	Firmware File Name
ControlNet	Slave devices		
	COM-CNS	CNS COM-CNS	CNS.H5J
	CIF30CNS	CNS CIF30CNS	CNS.H3B
	C-104CNS	CNS C-104CNS	CNS.H7D

1.3.7 Serial Protocol

System	Device Type	Firmware Name	Firmware File Name
Protocol	ASCII		
	C30-ASC	ASIM COM-ASM	ASIM.H5L
	C50-ASC	ASIM C104-ASM	ASIM.H7Q
	CIF50ASM	ASIM CIF50ASM	ASIM.H6I
	Modnet 1N		
	C50-M1N	MOD1N CIF50	M1N.H60
	Modbus RTU		
	C50-MBR	MODBUS CIF50	MBR.H60
	3964R		
	C104-NVR	3964R CIF104	NVR.H70
	C30-NVR	3964R CIF30	NVR.H21
	C50-NVR	3964R CIF50	NVR.H60
	RK512		
	C30-RK	RK512 CIF30	RK.H21
	C50-RK	RK512 CIF50	RK.H60

2 Installation

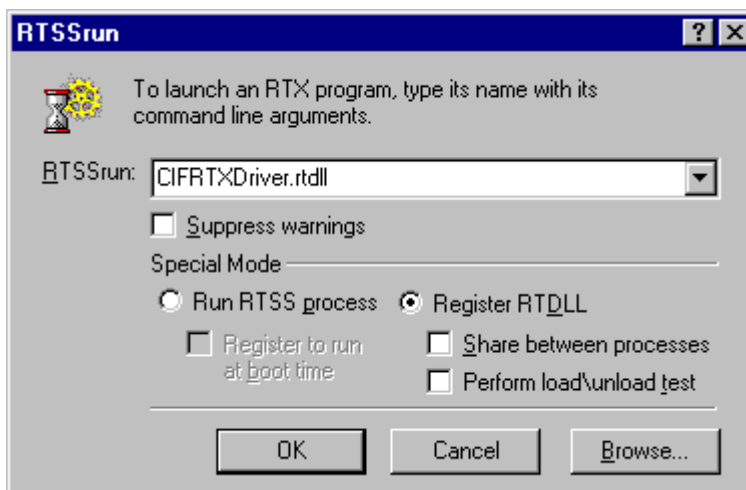
Two steps must be performed to install the Hilscher RTX driver.

Step1: Copy driver and firmware files

- Copy the Hilscher RTX driver DLL **CIFRTXDriver.rtdll** from the installation CD to the local harddisk.
- Copy the necessary firmware files to the local harddisk. Check the table **Supported Devices** which file correspond to the used CIF hardware.

Step2: Register the driver DLL

- The driver DLL **CIFRTXDriver.rtdll** must be registered within the RTX environment. Therefore the RTX program RTSSrun can be used. The RTX environment will generate a local copy of the driver DLL and makes the driver DLL available for RTSS applications.



RTSSrun program example

Note: You have to repeat this step if you getting an updated driver DLL

3 Fieldbus Configuration Details

This section describes how to configure a CIF device and how to check a configuration which already exists on the CIF hardware against an application local configuration.

Usually a fieldbus configuration is created and checked by SyCon (**S**ystem **C**onfigurator). After creating a valid configuration SyCon is able to export the configuration into a database file (.DBM). This exported file can be used in the driver to run the automatic database checking and download (see *DevInitBoard()* function). By using this functionality, an application can always be sure the CIF card is correctly configured.

If an application generates an own configuration (mostly SoftPLCs) it could be necessary to compare it against an actual CIF configuration. To enable a configuration check in such cases, the external C module *CifConfiguration.c* can be used. This module includes all necessary functions to read the actual configuration from the hardware and to compare it against an application local configuration.

3.1 Module CIFConfiguration

CIFConfiguration is a C module which enables an application to check the own (local) configuration against a configuration which is already loaded on a CIF card. It also includes functions for configuration dependent aspects like reading the configured watchdog time, or functions which can be used during runtime to check a fieldbus station.

Note: These functions are not on all CIF cards available (e.g. ASI and all serial protocols).

Function overview:	
CNFSwapWord	Word swap function
CNFSwapDword	Double word swap function
CNFGetStationDescription	Read the station description of a given station
CNFGetActualSendRecvSize	Calculate the necessary send/receive size to exchange all slave data
CNFIsStationConfigured	Check if a given station is configured
CNFIsStationInDataExchangeMode	Check if station is in data exchange mode
CNFGetWatchdogTime	Read the watchdog time configured by SyCon
CNFCalculateSyconIdx	Create the values Slot and Idx from a SyCon configuration(*)
CNFTestModuleConfig	Compare the module information against a module previously found by Slot and Idx Check values: Direction DataType, Offset Size
CNFTestModuleConfigNoSlotIdx	Compare the module information only using module parameters (no Slot and Idx used) Check values: Direction DataType, Offset Size

3.2 Fieldbus Startup Behaviour

SyCon allows to configure the start up behaviour of a fieldbus system.

It can be '*Automatic release of the communication by the device*', which means the master starts the fieldbus system as soon as the card has finished it's power on sequence. The fieldbus becomes active and slaves are able to driver there outputs.

The option '*Controlled release of the communication by the application program*' can be used to prevent an automatic startup of the field bus system. This enables the application to be fully initialized before starting the fieldbus. In this case the application has to use the function *DevSetHostState(.., HOST_READY,..)* to signal the master to start the fieldbus system.

SyCon settings	
<i>'Automatic release of the communication by the device'</i>	The fieldbus system will start up as soon as the card has finished it's power on sequence
<i>'Controlled release of the communication by the application program'</i> (recommended)	The start of the fieldbus system can be controlled by an application

3.3 Supported Handshake Modes

Handshake modes are used to control the access to the I/O process data image between the PC (Host) and the CIF card (Device). The setting of the transfer mode is very important because it will influence the consistency of the process data and the fieldbus behaviour. Transfer modes are only available on master cards.

The modes are configured via SyCon which offers up to six modes, depending on the used CIF hardware and fieldbus system.

Mode	Supported
Bus synchronous, device controlled	NO (Only possible on dedicated systems, because the system must respond during a bus data cycle which can be less than 350 micro seconds)
Buffered device controlled	YES
No consistence, uncontrolled	YES (Not recommended, process data which are not of the type byte can be transferred inconsistent)
Buffered host controlled	YES (recommended)
Bus synchronous, host controlled	YES (Not on all fieldbus systems and CIF cards available. System is responsible to drive the bus)
Buffered, extended host controlled	NO

Note: For more information about transfer modes, consult the 'TOOLKIT' manual section 'IO Communication with a Process Image'

4 Driver Functions

The following section describes the internal driver handling and new functions specially designed for some SoftPLC requirements.

4.1 Initialization Process

The following work will be carried out by the driver during initialization:

- Hardware scan Scanning for installed CIF hardware. This includes ISA PCI and DMA cards.
- Hardware reset Each CIF card will be restart by processing a cold reboot. This will be done to get boards working which have been previously stopped by a watchdog hit.
- Firmware download If the application supports a firmware file directory, the driver will check the firmware file version against the version currently running on the CIF card. If the they are different, the driver carries out an automatic firmware download.
- Configuration download If the application supports a configuration database file directory and name, the driver will check the configuration database against the database currently located on the CIF card. If the databases are different, the driver carries out an automatic database download.
- Fieldbus Communication The fieldbus communication will be always stopped before leaving the driver.

After the initialization procedure the CIF card is ready to start. From this point the configuration, generated by SyCon, decides if the fieldbus system starts up automatically or controlled by the application (see function *DevSetHostState()*).

4.2 Extended Configuration Checking

An extended configuration check can be implemented in the application by using a external program module. This module includes functions to run a configuration check on a per slave module base.

Following information can be checked:

- Station address
- Data type
- Data size
- Data offset in the I/O process image

4.3 New I/O Data Exchange Function

Some of the SoftPLCs are reading their input values on the beginning of the SPS cycle than processing the data and writing the data on the end of the cycle.

Therefore the driver offers two new functions (*DevExchangePLCData()* and *DevIsPLCDataReady()*). It is also possible to write directly to the I/O image without handling an internal process data map.

Refer to the section 'Programming' for a detailed description.

5 Programming

The Application Programming Interface (API) of the Hilscher RTX driver is based on the already known CIF Device Driver Interface (see CIF Device Driver manual). Since the RTX driver supports ISA, PCI and DMA cards in one driver the original driver functions have been extended by a DRIVERHANDLE and DEVICEHANDLE which replaces the original board number (*usDevNumber*) These new parameters are used in all function calls and necessary to simplify the internal driver handling and to eliminate driver overhead during the calls.

DevInitBoard() will return the driver and device handles which are needed. Also the automatic firmware and configuration file download will be handled during this function.

5.1 Standard API

Because of the new functionality's in *DevInitBoard()* you will find the description in the following section. All other functions from the standard API are described in the CIF Device Driver manual.

5.1.1 DevInitBoard()

Description:

After an application has opened the driver with *DevOpenDriver()* it must call *DevInitBoard()*.

The application has to supply either a driver type or a device name and a device number between 0..3, because the driver is able to handle more than one device at a time. If a device is installed and running the function returns a driver and device handle. These handles are necessary for all other driver function.

DevInitBoard() is also able to check the actual firmware and configuration against given files and if they are different a download will be carried out.

To run the firmware check, only a directory path is necessary. The driver will choose the firmware file by itself, according to the hardware. For the configuration check a filename including the path (full file name) must be provided by the application

Note: In case of DMA cards (CIF100), the application must always provide a firmware path and configuration file.

```
short DevInitBoard ( unsigned short   usDriverType,
                   char              *pszDevName,
                   unsigned short   usDevNumber,
                   char              *pszFWFilePath,
                   char              *pszCFFullFileName,
                   DRIVERHANDLE     *pDriver,
                   CIFHANDLE        *pDevInstance );
```

Parameter:

Type	Parameter	Description
unsigned char	usDriverType	Select a device by a driver type CIF_ISA = 1 (ISA hardware) CIF_PCI = 2 (PCI hardware) CIF_DMA = 3 (DMA hardware) If usDriverType = 0, pszDevName is used to select the device
char	*pszDevName	Device name as a zero terminated string See section ' <i>Supported CIF Cards, Fieldbus Systems and Protocols</i> ' for valid names (e.g. "CIF50-PB"). Device name is ignored if usDriverType is not 0
unsigned short	usDevNumber	Device number 0..3
char	*pszFWFilePath	File path to the directory where the firmware files are located as a zero terminated string.
char	*pszCFFullFileName	File path and filename for the configuration database file as a zero terminated string
DRIVERHANDLE	*pDriver	Buffer pointer to store the driver handle for this device
CIFHANDLE	*pCIF	Buffer pointer to store the device handle for this device

5.1.2 DevGetBoardInfo()

Description:

With *DevGetBoardInfo()*, the user can read global information of all communication boards the device driver knows. The users interface offers a data structure which describes the board information. The function copies the number of data, given in the parameter *usSize*. This function can be used after *DevOpenDriver()* and before opening a specific device with *DevInitBoard()*.

```
short DevGetBoardInfo (      unsigned short      usDriverType
                           unsigned short      usSize,
                           void                *pvData);
```

Type	Parameter	Description
unsigned short	usDriverType	Select a device by a driver type CIF_ISA = 1 (ISA hardware) CIF_PCI = 2 (PCI hardware) CIF_DMA = 3 (DMA hardware)
unsigned short	usSize	Size of the users data buffer and length of data to be read
void *	pvData	Pointer to the users data buffer

5.2 Additional API Functions

In addition to the standard CIF Device Driver interface, the API is extended by functions which allowing an application to directly access the I/O process data image.

5.2.1 DevExchangePLCData

Description:

DevExchangePLCData() works only in conjunction with *DevGetInfo()* and *DevIsPLCDataReady()*.

The function only starts a data exchange between the host (PC) and the device (CIF card). No data will be copied into the I/O process image of the CIF card. This must be done by the application by using the I/O image pointers returned from the *DevGetInfo()* function. *DevIsPLCDataReady()* is used to determine if a previously started data exchange is complete.

Attention: During an active data transfer, it is not allowed to access the I/O process image and to mix this function with other *DevExchangeIO* functions.

```
short DevExchangePLCData ( DRIVERHANDLE    hDriver,
                          CIFHANDLE        hCif,
                          unsigned short   usSendSize,
                          unsigned short   usReceiveSize,
                          unsigned long    ulTimeout );
```

Parameter:

Type	Parameter	Description
DRIVERHANDLE	hDriver	Driver handle for this device
CIFHANDLE	hCIF	Device handle
unsigned short *	usSendSize	Only for DMA cards otherwise not used Number of bytes to send to the hardware. Transfer will always start with offset 0 in the send process data image
unsigned short	usReceiveSize	Only for DMA cards otherwise not used Number of bytes to read from the hardware. Transfer will always start with offset 0 in the receive process data image
unsigned long	ulTimeout	Not used

A working test program is included. The functions are located in the module *CIFRTXTestIOTransfer.c*.

5.2.2 DevIsPLCDataReady

Description:

DevIsPLCDataReady() is used to check if a data exchange between a host and the CIF card is complete.

```
short DevIsPLCDataReady(    DRIVERHANDLE    hDriver,  
                           CIFHANDLE            hCif,  
                           unsigned short      *pusState);
```

Parameter:

Type	Parameter	Description
DRIVERHANDLE	hDriver	Driver handle for this device
CIFHANDLE	hCIF	Device handle
unsigned short *	pusState	Pointer to state buffer TRUE = data transfer done FALSE = data transfer activ

5.2.3 DevGetInfo

The function *DevGetInfo()* offers the new selection *GET_CIF_PLC_DRIVER_INFO* to get the pointer to the I/O process image and the size of it.

The pointers are needed to directly access the I/O process data image of a CIF card.

```
short DevGetInfo( DRIVERHANDLE      hDriver,
                 CIFHANDLE        hCif,
                 unsigned short    usFunction
                 unsigned short    usSize
                 void              *pvData);
```

Parameter:

Type	Parameter	Description
DRIVERHANDLE	hDriver	Driver handle for this device
CIFHANDLE	hCIF	Device handle
unsigned short *	usFunction	Mode of the information to read GET_CIF_PLC_DRIVER_INFO
unsigned short	usSize	Size of information buffer
void*	pvData	Pointer to informationstructure

/ PLC information structure */*

```
typedef struct tagCIF_PLC_DRIVER_INFO
{
    void          pvInput;          /* Pointer to input data */
    unsigned long ulInputSize;      /* Size of the input image */
    void          *pvOutput;        /* Pointer to output data */
    unsigned long ulOutputSize;     /* Size of the output image */
    TASKSTATE     *ptTaskState;     /* Protocol specific structure*/
} CIF_PLC_DRIVER_INFO;
```

5.3 API Files

CIFRTXDriver.rtdll

This is the Hilscher RTX driver.

CIFRTXDriver.h

This is the RTX driver definition file. It offers the function prototypes from the RTX driver DLL (CIFRTXDriver.rtdll) and some general definitions.

It requires the standard CIF API definition file CIFUSER.H and the hardware specific definitions from RCS_USER:H

CifConfiguration.h

Holds the definitions for the module CifConfiguration.c.

This one needs also the RTX driver definitions from CIFRTXDriver.h and the following fieldbus specific header files:

COM_user.h	CANOpen Master
COS_user.h	CANOpen Slave
DNM_user.h	DeviceNet Master
DNS_user.h	DeviceNet Slave
DPM_user.h	PROFIBUS-DP Master
DPS_user.h	PROFIBUS-DP Slave
IBM_user.h	InterBus Master

5.4 Example Program CIFRTXTest

CIFRTXTest is an example program which shows the initialization of a device and how to use the configuration checking function from the module *CifConfiguration.c*.

6 Additional Information

Please use the listed manuals if you are searching additional information.

Manual name	Content
CIF Device Driver	General driver description, functions and API
Protocol Manual	General protocol information and definition
Protocol Interface Manual	Protocol specific definitions and functions

7 Error Numbers

7.1 List of Error Numbers

The column *Hint* shows if there is additional error information available. If 'Yes' then see section *Additional Error Information*, which is the next section.

Value	Parameter	Description	Hint
0	DRV_NO_ERROR	no error	
-1	DRV_BOARD_NOT_INITIALIZED	DRIVER Board not initialized	yes
-2	DRV_INIT_STATE_ERROR	DRIVER Error in internal init state	
-3	DRV_READ_STATE_ERROR	DRIVER Error in internal read state	
-4	DRV_CMD_ACTIVE	DRIVER Command on this channel is active	
-5	DRV_PARAMETER_UNKNOWN	DRIVER Unknown parameter in function occurred	
-6	DRV_WRONG_DRIVER_VERSION	DRIVER Version is incompatible with DLL	
-7	DRV_PCI_SET_CONFIG_MODE	DRIVER Error during PCI set run mode	
-8	DRV_PCI_READ_DPM_LENGTH	DRIVER Could not read PCI dual port memory length	
-9	DRV_PCI_SET_RUN_MODE	DRIVER Error during PCI set run mode	
-10	DRV_DEV_DPM_ACCESS_ERROR	DEVICE Dual port ram not accessible (board not found)	yes
-11	DRV_DEV_NOT_READY	DEVICE Not ready (ready flag failed)	yes
-12	DRV_DEV_NOT_RUNNING	DEVICE Not running (running flag failed)	yes
-13	DRV_DEV_WATCHDOG_FAILED	DEVICE Watchdog test failed	yes
-14	DRV_DEV_OS_VERSION_ERROR	DEVICE Signals wrong OS version	yes
-15	DRV_DEV_SYSERR	DEVICE Error in dual port flags	
-16	DRV_DEV_MAILBOX_FULL	DEVICE Send mailbox is full	
-17	DRV_DEV_PUT_TIMEOUT	DEVICE PutMessage timeout	yes
-18	DRV_DEV_GET_TIMEOUT	DEVICE GetMessage timeout	yes
-19	DRV_DEV_GET_NO_MESSAGE	DEVICE No message available	
-20	DRV_DEV_RESET_TIMEOUT	DEVICE RESET command timeout	yes
-21	DRV_DEV_NO_COM_FLAG	DEVICE COM-flag not set	yes
-22	DRV_DEV_EXCHANGE_FAILED	DEVICE IO data exchange failed	
-23	DRV_DEV_EXCHANGE_TIMEOUT	DEVICE IO data exchange timeout	yes
-24	DRV_DEV_COM_MODE_UNKNOWN	DEVICE IO data mode unknown	
-25	DRV_DEV_FUNCTION_FAILED	DEVICE Function call failed	
-26	DRV_DEV_DPMSIZE_MISMATCH	DEVICE DPM size differs from configuration	
-27	DRV_DEV_STATE_MODE_UNKNOWN	DEVICE State mode unknown	
-30	DRV_USR_OPEN_ERROR	USER Driver not opened (device driver not loaded)	yes
-31	DRV_USR_INIT_DRV_ERROR	USER Can't connect with device	
-32	DRV_USR_NOT_INITIALIZED	USER Board not initialized (DevInitBoard not called)	

Value	Parameter	Description	Hint
-33	DRV_USR_COMM_ERR	USER IOCTL function failed	
-34	DRV_USR_DEV_NUMBER_INVALID	USER Parameter DeviceNumber invalid	
-35	DRV_USR_INFO_AREA_INVALID	USER Parameter InfoArea unknown	
-36	DRV_USR_NUMBER_INVALID	USER Parameter Number invalid	
-37	DRV_USR_MODE_INVALID	USER Parameter Mode invalid	
-38	DRV_USR_MSG_BUF_NULL_PTR	USER NULL pointer assignment	
-39	DRV_USR_MSG_BUF_TOO_SHORT	USER Message buffer too short	
-40	DRV_USR_SIZE_INVALID	USER Parameter Size invalid	
-42	DRV_USR_SIZE_ZERO	USER Parameter Size with zero length	
-43	DRV_USR_SIZE_TOO_LONG	USER Parameter Size too long	
-44	DRV_USR_DEV_PTR_NULL	USER Device address null pointer	
-45	DRV_USR_BUF_PTR_NULL	USER Pointer to buffer is a null pointer	
-46	DRV_USR_SENDSIZE_TOO_LONG	USER Parameter SendSize too long	
-47	DRV_USR_RECVSIZE_TOO_LONG	USER Parameter ReceiveSize too long	
-48	DRV_USR_SENDBUF_PTR_NULL	USER Pointer to send buffer is a null pointer	
-49	DRV_USR_RECVBUF_PTR_NULL	USER Pointer to receive buffer is a null pointer	
-50	DRV_DMA_TIMEOUT_CH4	DMA read IO timeout	
-51	DRV_DMA_TIMEOUT_CH5	DMA write IO timeout	
-52	DRV_DMA_TIMEOUT_CH6	DMA PCI transfer timeout	
-53	DRV_DMA_TIMEOUT_CH7	DMA download timeout	
-54	DRV_DMA_INSUFF_RES_MEM	DMA Memory allocation error	
-70	DRV_ERR_ERROR	DRIVER General error	
-71	DRV_DMA_ERROR	DRIVER General DMA error	
-72	DRV_BATT_ERROR	DRIVER Battery error	
-73	DRV_PWF_ERROR	DRIVER Power failed error	
-80	DRV_USR_DRIVER_UNKNOWN	USER driver unknown	
-81	DRV_USR_DEVICE_NAME_INVALID	USER device name invalid	
-82	DRV_USR_DEVICE_NAME_UNKNOWN	USER device name unknown	
-83	DRV_USR_DEVICE_FUNC_NOTIMPL	USER device function not implemented	
-100	DRV_USR_FILE_OPEN_FAILED	USER file not opened	
-101	DRV_USR_FILE_SIZE_ZERO	USER file size zero	
-102	DRV_USR_FILE_NO_MEMORY	USER not enough memory to load file	
-103	DRV_USR_FILE_READ_FAILED	USER file read failed	
-104	DRV_USR_INVALID_FILETYPE	USER file type invalid	
-105	DRV_USR_FILENAME_INVALID	USER file name not valid	
-110	DRV_FW_FILE_OPEN_FAILED	USER firmware file not opened	
-111	DRV_FW_FILE_SIZE_ZERO	USER firmware file size zero	
-112	DRV_FW_FILE_NO_MEMORY	USER not enough memory to load firmware file	
-113	DRV_FW_FILE_READ_FAILED	USER firmware file read failed	
-114	DRV_FW_INVALID_FILETYPE	USER firmware file type invalid	
-115	DRV_FW_FILENAME_INVALID	USER firmware file name not valid	

Value	Parameter	Description	Hint
-116	DRV_FW_DOWNLOAD_ERROR	USER firmware file download error	
-117	DRV_FW_FILENAME_NOT_FOUND	USER firmware file not found in the internal table	
-118	DRV_FW_BOOTLOADER_ACTIVE	USER firmware file BOOTLOADER active	
-119	DRV_FW_NO_FILE_PATH	USER firmware file not file path	
-120	DRV_CF_FILE_OPEN_FAILED	USER configuration file not opened	
-121	DRV_CF_FILE_SIZE_ZERO	USER configuration file size zero	
-122	DRV_CF_FILE_NO_MEMORY	USER not enough memory to load configuration file	
-123	DRV_CF_FILE_READ_FAILED	USER configuration file read failed	
-124	DRV_CF_INVALID_FILETYPE	USER configuration file type invalid	
-125	DRV_CF_FILENAME_INVALID	USER configuration file name not valid	
-126	DRV_CF_DOWNLOAD_ERROR	USER configuration file download error	
-127	DRV_CF_FILE_NO_SEGMENT	USER no flash segment in the configuration file	
-128	DRV_CF_DIFFERS_FROM_DBM	USER configuration file differs from database	
-131	DRV_DBM_SIZE_ZERO	USER database size zero	
-132	DRV_DBM_NO_MEMORY	USER not enough memory to upload database	
-133	DRV_DBM_READ_FAILED	USER database read failed	
-136	DRV_DBM_NO_FLASH_SEGMENT	USER database segment unknown	
-150	DEV_CF_INVALID_DESCRIPTOR_VERSION	CONFIG version of the descriptor table invalid	
-151	DEV_CF_INVALID_INPUT_OFFSET	CONFIG input offset is invalid	
-152	DEV_CF_NO_INPUT_SIZE	CONFIG input size is 0	
-153	DEV_CF_MISMATCH_INPUT_SIZE	CONFIG input size does not match configuration	
-154	DEV_CF_INVALID_OUTPUT_OFFSET	CONFIG invalid output offset	
-155	DEV_CF_NO_OUTPUT_SIZE	CONFIG output size is 0	
-156	DEV_CF_MISMATCH_OUTPUT_SIZE	CONFIG output size does not match configuration	
-157	DEV_CF_STN_NOT_CONFIGURED	CONFIG Station not configured	
-158	DEV_CF_CANNOT_GET_STN_CONFIG	CONFIG cannot get the Station configuration	
-159	DEV_CF_MODULE_DEF_MISSING	CONFIG Module definition is missing	
-160	DEV_CF_MISMATCH_EMPTY_SLOT	CONFIG empty slot mismatch	
-161	DEV_CF_MISMATCH_INPUT_OFFSET	CONFIG input offset mismatch	
-162	DEV_CF_MISMATCH_OUTPUT_OFFSET	CONFIG output offset mismatch	
-163	DEV_CF_MISMATCH_DATA_TYPE	CONFIG data type mismatch	
-164	DEV_CF_MODULE_DEF_MISSING_NO_SLOT	CONFIG Module definition is missing,(no Slot/idx)	
>=1000	RCS_ERROR	Board operation system errors will be passed with this offset (e.g. error 1234 means RCS error 234). Only if a ready fault occurred during board initialization.	

7.2 Additional Error Information

This section contains more informations about possible reasons to certain error numbers.

Error: -1

The communication board is not initialized by the driver.

No or wrong configuration found for the given board.

- Check the driver configuration
- Driver function used without calling DevOpenDriver() first

Error: -6

The device driver version does not corresponds to the driver DLL version. From version V1.200 the internal command structure between DLL and driver has changed.

- Make sure to use the same version of the device driver and the driver DLL

Error: -10

Dual ported RAM (DPM) not accessible / no hardware found.

This error occurs, when the driver is not able to read or write to the DPM

- Check the BIOS setting of the PC
- Memory address conflict with other PC components, try another memory address
- Check the driver configuration for this board
- Check the jumper setting of the board

Error: -11

Board is not ready.

This is a general error, the board has a hardware malfunction.

Error: -12

At least one task is not initialized. The board is ready but not all tasks are running.

- No data base is loaded into the device
- Wrong parameter that causes that a task can't initialize. Use ComPro menu *Online-task-version*.

Error: -14

No license code found on the communication board.

- Device has no license for the used operating system or customer software.
- No firmware or no data base on the device loaded.

Error: -17

No message could be send during the timeout period given in the

DevPutMessage() function.

- Using device interrupts

Wrong or no interrupt selected. Check interrupt on the device and in driver registration.

They have to be the same!. Interrupt already used by an other PC component.

- Device internal segment buffer full

PutMessage() function not possible, because all segments on the device are in use.

This error occurs, when only PutMessage() is used but not GetMessage().

- HOST flag not set for the device

No messages are taken by the device. Use DevSetHostState() to signal a board an application is available.

Error: -18

No message received during the timeout period given in the

DevGetMessage() function.

- Using device interrupts

Wrong or no interrupt selected. Check interrupt on the device and in driver registration. They have to be the same!. Interrupt already used by an other PC component.

- The used protocol on the device needs longer than the timeout period given in the `DevGetMessage ()` function

Error: -20

The device needs longer than the timeout period given in the `DevReset ()` function

- Using device interrupts

This error occurs when for example interrupt 9 is set in the driver registration but no or a wrong interrupt is jumpered on the device (=device in pollmode).

Interrupt already used by an other PC component.

- The timeout period can differ between fieldbus protocols

Error: -21

The device can not reach communication state.

- Device not connected to the fieldbus

- No station found on the fieldbus

- Wrong configuration on the device

Error: -23

The device needs longer than the timeout period given in the

`DevExchangeIO ()` function.

- Using device interrupts

Wrong or no interrupt selected. Check interrupt on the device and in driver registration.

They have to be the same!. Interrupt already used by an other PC component.

Error: -30

The device driver could not be opened.

- Device driver not installed

- Wrong parameters in the driver configuration

If the driver finds invalid parameters for a communication board and no other boards with valid parameters are available, the driver will not be loaded.

Error: -33

A driver function could not be called. This is an internal error between the device driver and the DLL.

- Make sure to use a device driver and a DLL with the same version.

- An incompatible old driver DLL is used.