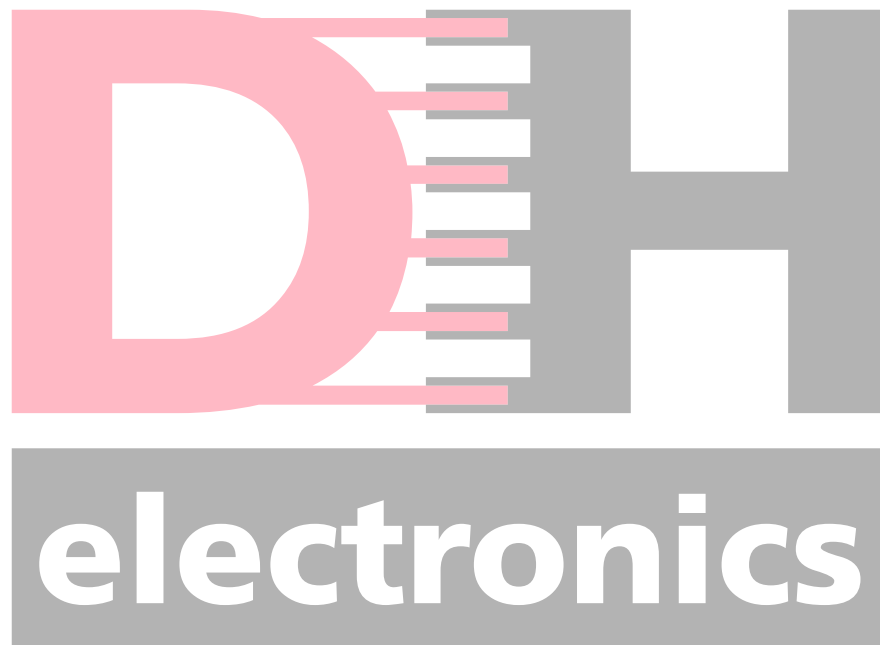


Comprehensive User's Guide

XLON[®] PCI Adapter

Version 1.0



DH electronics GmbH

Am Anger 8
83346 Bergen
Germany
Tel.: +49 8662 4882-0
Fax.: +49 8662 4882-99
E-Mail: info@xlon.de
www.xlon.de




This documentation is subject to changes without notice. The manufacturer assumes no responsibility or liability for any errors or inaccuracies that may appear in this document.

The manufacturer have no liability or responsibility to the original purchaser or any other person or entity with respect to any claim, loss, liability, or damage caused or alleged to be caused directly or indirectly by any product of the manufacturer or the accompanying documentation.

Comprehensive User's Guide

XLON[®] PCI Adapter


Version 1.0

1	About this manual	5
2	Introduction / Product Information	5
3	Installation of the  XLON[®] PCI adapter	7
3.1	Hints-	7
3.2	Hardware installation	8
3.3	Software Installation	10
3.3.1	Initial installation under Windows	10
3.3.2	Initial Installation under Windows CE 3.0-	13
3.3.2.1	Installing the driver to a running Windows CE System	13
3.3.2.2	Creating a new Windows CE Image	14
3.3.2.3	Registration Entry	14
3.3.2.4	Hardware Configuration	16
3.3.3	Initial Installation under Linux	16
3.3.4	Driver update	18
3.3.4.1	Windows	18
3.3.4.2	Windows CE 3.0-	23
3.3.4.3	Linux	23
3.4	Software De-installation	23
4	Configuring and Testing-	24
4.1	Check of settings under Windows	25
4.1.1	General settings	25
4.1.2	Driver information	26
4.1.3	Occupied Hardware-Resources	28
4.1.4	Properties of the  XLON[®] PCI adapter	29
4.2	Test of the  XLON[®] PCI adapter under Windows	31
4.2.1	Diagnosis by Software	31
4.2.2	Diagnosis by LED	32
5	Technical Specification	33
5.1	Hardware	33
5.1.1	General Information	33
5.1.2	Plug Connection	34
5.1.3	Block model	34

XLON[®] is registered trademark of DH electronics GmbH.
 Echelon[®], LON[®], LonWorks[®], LonManager[®], LonMark[®], LonPoint[®], LonTalk[®], LonUsers[®], Neuron, 3120, 3150 are registered trademarks of Echelon Corporation.
 Windows[®] is registered trademark of Microsoft Corporation.
 Other brand and product names are trademarks or registered trademarks of their respective holders..

5.1.4 Technical Details of the Hardware - - - - -	35
5.1.5 Supported Transceivers - - - - -	35
6 Software Access - - - - -	37
6.1 Application Interface under Windows - - - - -	37
6.1.1 LNS-applications- - - - -	37
6.1.2 Configuration of the Network Interface Buffer - - - - -	38
6.1.3 Programming your own application - - - - -	38
6.1.3.1 Opening the device driver - - - - -	38
6.1.3.2 Registration of an Event-Handle - - - - -	40
6.1.3.3 Reading of data from the device driver- - - - -	41
6.1.3.4 Writing of data on the device drivers- - - - -	41
6.1.3.5 Closing the device driver - - - - -	42
6.1.3.6 Important Programing Information - - - - -	42
6.2 Application interface under Windows CE 3.0- - - - -	43
6.2.1 CreateFile() - - - - -	43
6.2.5.1 „GetVersion“ by DeviceIoControl()- - - - -	48
6.2.5.2 „ReadWait“ by DeviceIoControl() - - - - -	49
6.2.6 GetLastError() - - - - -	50
6.3 Application Interface under Linux - - - - -	52
7 Appendix - - - - -	53
7.1 Declaration of Conformity - - - - -	53
8 Revision History- - - - -	54

1 About this manual

This guide describes the hardware installation, software driver installation and the setup and configuration of the  adapter.

The developer is given information about creating suitable application software.

Used ideograms and symbols

In this manual the following ideograms and symbols are used to emphasize special points.



Attention! Very important point concerning safety.



Danger of injury caused by voltage.



Danger of damage to electronic components caused by static loading.





Danger of injury caused by mechanic components.

- enumeration, working step

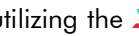

 remarkable instruction


2 Introduction / Product Information

The  LonTalk® adapter can be used to connect your PC to a LonWorks® network via the PCI bus, according to PCI specification Rev. 2.2. It is designed for use in industrial control, process control and building automation.

The  supports not only the LNS Network Services Interface (NSI) for all LNS tools, but also the LonManager®-API interface on older applications.

Thanks to its Client-Server-Architecture, the LNS network operating system provides simultaneous access to highly diverse applications on the Network-Services-Server (NSS). As a result LonWorks® network tools produced by different manufacturers can be simultaneously implemented for installation, maintenance, monitoring and control.

By utilizing the  it is also possible, to transform a PC into an extremely efficient LonWorks® node. In this case, the LonWorks® application runs on the PC and the  handles the operation of the LonTalk® protocol. This provides much more processing power for a LonWorks® application, in comparison to a Neuron® chip based node. In addition, the number of possible network variables is been considerably increased from 62 up to 4096, which can frequently play an important role when it comes to maintenance and monitoring applications.

The  has an integrated FTT-10A transceiver for Free Topology and Link Power networks or a RS485 transceiver for Twisted Pair networks.


An external Service button and Service LED is for use with manual installations and for

visualization of the state of the LonTalk® adapter. An extra LED displays any bus traffic on the network.

All available drivers are included in the  XLON® PCI Kit. Sample programs for accessing the driver with C/C++ and VisualBasic can be downloaded from:

<http://www.xlon.de>

Scope of delivery

-  XLON® PCI controller card
- Weidmueller-clamp für LonWorks® network connection
- Floppydisk/CD with device drivers
- User's manual for installation.

Available variants

- PCI1-WM-FTT with integrated FTT-10A Transceiver
- PCI1-WM-RS with integrated RS485 Transceiver




Find more informations about LonWorks® networks under:

www.echelon.com

3 Installation of the XLON[®] PCI adapter

3.1 Hints

For installation of the  XLON[®] PCI adapter the computer needs to have a free PCI slot.

Look up the manual of the computer manufacturer for questions concerning running down and opening of the PC or the arrangement of slots.

Before opening the device please make sure:




- Shut down the operating system and turn off the PC.

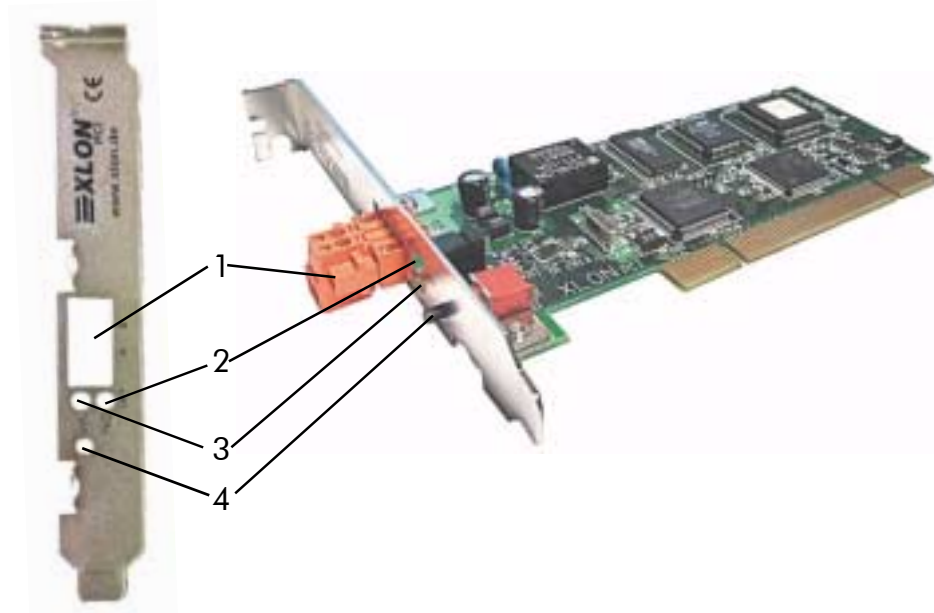


- Remove the power cord



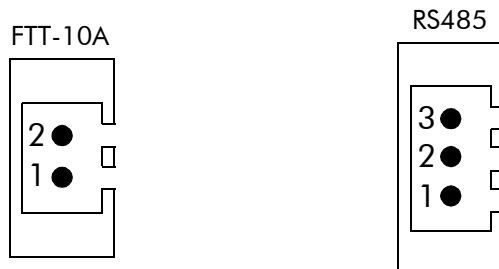
- This product contains devices which are sensitive to static electricity. Before installing or removing the  XLON[®] PCI adapter or the network cables, discharge any static electricity which may have accumulated to earth ground (for example by touching the metallic frame of the computer).

3.2 Hardware installation






No.	Term	Remark
1	LON network connector	2-pin: FTT-10A 3-pin: RS485
2	LED green „PKD“	Display of data traffic on the LonWorks® network
3	LED yellow „SVC“	Display of Service Pin Neuron Processor
4	Service Pin Button	Manual release of a Service Pin message

Pin allocation of the LON network connector



Pin	FTT-10A	RS485
1	NET A	RS485 A
2	NET B	RS485 B
3	not available	GND

-
- Open PC
 - Remove the corresponding blank panel of a free PCI-slot
 - Insert the  adapter into the slot accurately
 - Fix metallic mounting frame of the  adapter to the PC-frame
 - Close PC
 - Plug in the LonWorks® network cable
 - Boot up the computer
 - Under Windows based operating systems a „Found New Hardware Wizard“ starts, (see chapter 3.3)
 - For installation of device drivers under Windows CE refer to chapter 3.3.2
 - Under LINUX  adapters are recognized automatically while loading the driver module.

De-installation

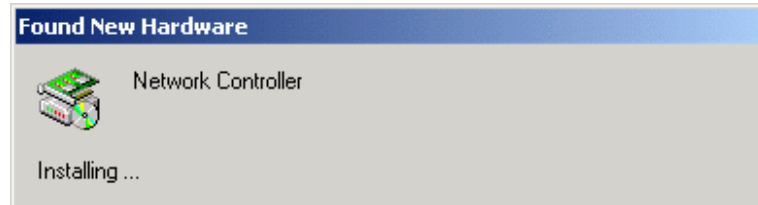
De-installation of the  adapter follows reverse order above. De-installation of driver software is not necessary.

3.3 Software Installation

3.3.1 Initial installation under Windows

Initial installation under Windows 2000 is shown exemplary below. Installation under other Windows operating systems works analogously!

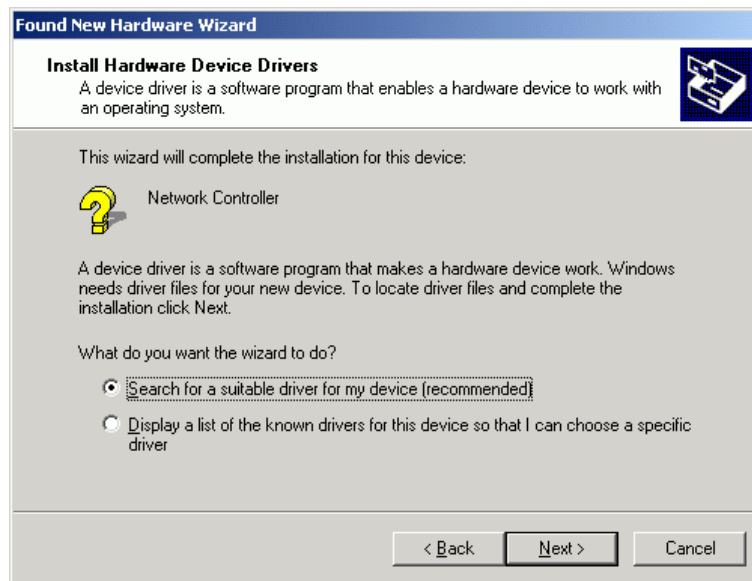
After restarting the computer you get the following message:



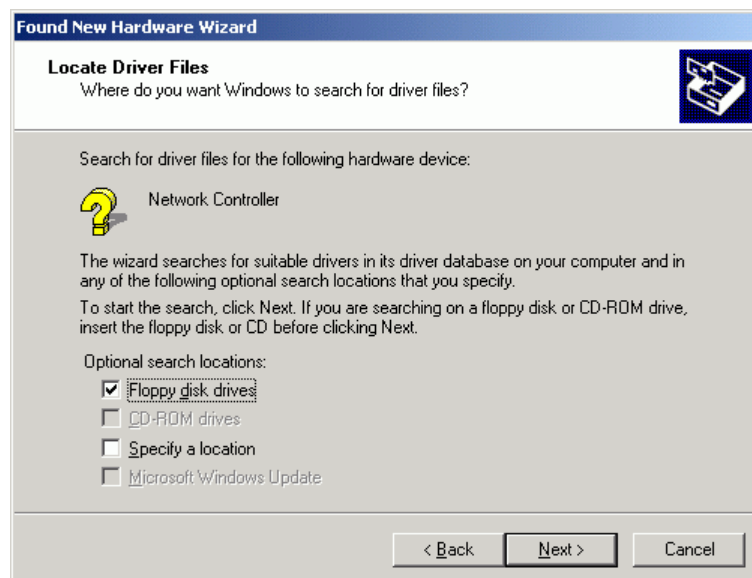
The wizard for device driver installation is started automatically:



- Click on „Next>“

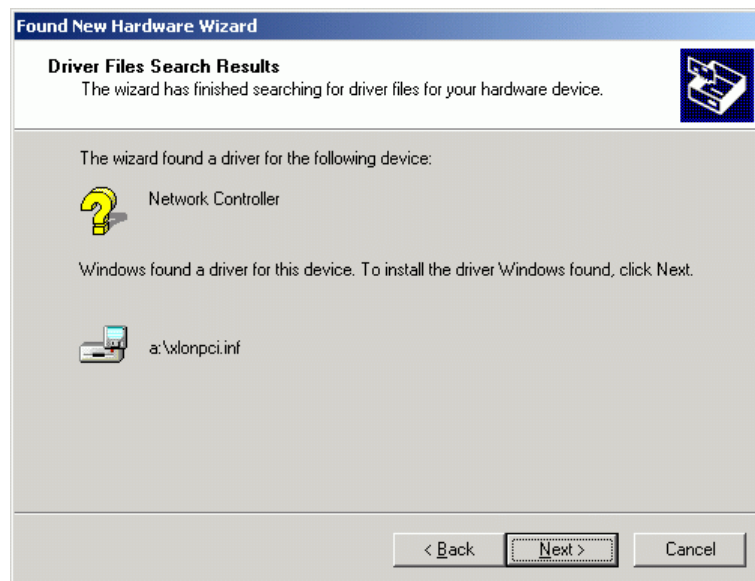


- Follow the procedure shown above and click on „Next>“

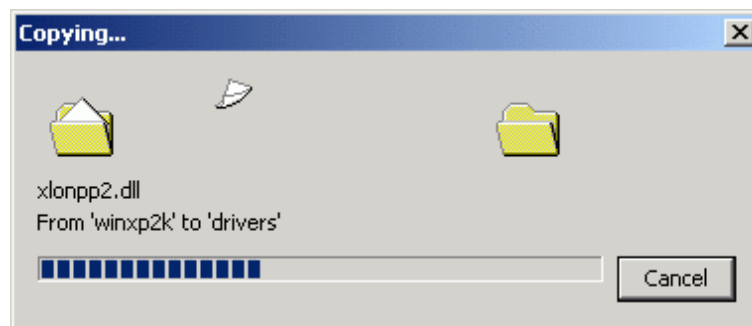


- Depending on the storage medium enclosed to your adapter either choose „Floppy disk drives“ or „CD-ROM drives“
- Insert Floppy disk or CD-ROM in the corresponding drive.
- Click on „Next>“

A driver has been found:



- Click on „Next>“



After ending the installation process:




- Click on „Finish“

3.3.2 Initial Installation under Windows CE 3.0

The device driver for Windows CE 3.0 is implemented in the shape of a „Stream Interface Device Driver“ as Dynamic Link Library (DLL) for the following processor platforms:

- x86

The files required for installation under Windows CE 3.0 can be downloaded from the website www.xlon.de.


The device driver supports up to 8  adapters within one system. The interrupts are allocated automatically. Although Windows CE 3.0 does not support a „native“ Interrupt Sharing the particular structure of the device driver enables Interrupt Sharing between any number of  adapters.




The device driver can either be added dynamically to a running Windows CE System or can be statically tied up in the Windows CE Image which has to be created. The second operation should only be done by experienced users who want to create a new Windows CE 3.0 Image. Both operations are explained below.

3.3.2.1 Installing the driver to a running Windows CE System

For driver installation on a Windows CE device with static RAM the driver file „xlon_pci.dll“ has to be copied manually into the directory „\Windows“. This has to be followed by registration entry as explained in chapter 3.3.2.3.

3.3.2.2 Creating a new Windows CE Image

For making the device driver available to the  adapter under „Microsoft Platform Builder 3.0“ the below steps have to be followed. They refer to a x86 Hardware Platform. The procedure is analog under other hardware architectures. Depending on the specific platform directory paths may however differ.


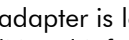

- Copy device driver file „xlon_pci.dll“ to directory „_WINCEROOT\PLATFORM\CEPC\FILES“.
- Copy component file „xlon.cec“ to directory „_CEPBDir\CEPB\CEC“.
- Start „Microsoft Platform Builder 3.0“ and load your Platform working area.
- Open the „File“ Menu and go to „Manage Platform Builder Components“
- Import the component file „xlon.cec“ by clicking „Import New“
- For the following point you need information on which PCI-Bus and in which PCI-Slot the  adapters are found. Find detailed information under chapter 3.3.2.4.
- Under the „Platform Builder“ manually add Windows CE Registry Information to file „platform.reg“ as described in chapter 3.3.2.3.
- Under the „Platform Builder“ manually add the content of file „xlon_pci.bib“ to file „platform.bib“.
- Under the „Platform Builder“ open menu „View“, and click on „Catalog“. The catalog view should open.
- In Catalog, open the Tree View „Catalog/Drivers/CEPC/XLON“.
- Add the  component to the current platform by clicking the right mouse button on „Add to Platform“.
- Restart the „Platform Builder“, create your new Windows CE 3.0 Image and load it on your target system. After the target system has been booted up the device driver for  adapter is loaded automatically by the „Device Manager“ of Windows CE and can then be used for your application.

3.3.2.3 Registration Entry



Find an example for a correct registration entry in file „xlon_pci.reg“. The content is explained below.


[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\XlonPCIx]

The registration code HKEY_LOCAL_MACHINE\Drivers\BuiltIn contains subcodes defining the device drivers that are loaded by the Device Manager after restarting the system. Each subcode holds the name of the device driver and contains the entries listed in the table below. The „x“ in subcode „...XlonPCIx“ is identical to the entry „Index“ in the table below.


Value name	Value type	Description
Dll	REG_SZ	This required entry specifies the file name for a driver DLL which is loaded by the Device Manager, e.g. xlon_pci.dll
Prefix	REG_SZ	This required value specifies the driver's device file name prefix. It is a three character identifier, such as LON.
Index	REG_DWORD	This optional value specifies the device index, a value from 0 to 9, that the Device Manger should assign to the driver.
Order	REG_DWORD	This value specifies the load order for a driver. If two drivers have the same load order value, the drivers load in the order they occur in the registry. This entry is useful when one driver must load after another.
FlushCancel	REG_DWORD	This value specifies wether the device driver sends a niFLUSHCANCEL command down to the network interface after receiving a niRESET command from the network interface. A value equal to 0 enables this feature, a value other than 0 disables this feature.
TranceiverID	REG_DWORD	This value specifies the Transceiver ID of the network transceiver used on the  adapter as specified by Echelon. Refer to chapter 5.1.5 for more information about Transceiver ID's.
Bus	REG_DWORD	This value specifies the number of the PCI bus on which the  adapter is located. Refer to chapter 3.3.2.4 for additional information.
Slot	REG_DWORD	This value specifies the number of the Slot on the specified bus on which the  adapter is located. Refer to chapter 3.3.2.4 for additional information.
FriendlyName	REG_SZ	A optional string containing a human-readable description of name for the device driver.

3.3.2.4 Hardware Configuration

With the Windows CE 3.0 device driver you can use up to 8  **XLON[®] PCI** adapters in one system. If this should not be sufficient for you, please contact DH electronics. The special software design of the device driver enables interrupt sharing between any number of  **XLON[®] PCI** adapters. This is very important because Windows CE does not support „native“ Interrupt Sharing.

Several Free- and Shareware Programs can be loaded from the Internet to find out the PCI-Bus Numbers and PCI-Slot Numbers for the  **XLON[®] PCI** adapters in the system. Often this can also be found in the manual enclosed by the manufacturer of the motherboard.

3.3.3 Initial Installation under Linux

The device driver for Linux is implemented in the form of a loadable kernel module. Currently the kernel versions 2.0, 2.2 und 2.4 are supported. The device driver supports up to 6  **XLON[®] PCI** adapters in one system. Interrupts are allocated automatically, interrupt sharing is supported.

After making sure that the kernel module fits with the current kernel version, loading can be tried. Please notice that the kernel module can only be loaded with the rights of a Superuser. The device driver module can be added to the kernel by the below order.

```
# /sbin/insmod xlonpci.o
```

If the module can't be loaded and you get message „Kernel version mismatch“ and if the first two digits are identical with the kernel version (i.e. 2.2 or 2.4) loading can be enforced by the below order including the option „-f“. The given warning has no effects on the function of the device driver.


```
# /sbin/insmod -f xlonpci.o
```

By the following order you can check if loading of the device driver module has been successful:


```
# /sbin/lsmmod
```

After this order the messages on the screen should contain the below lines:

Module	Size	Used by
xlonpci.o	6960	0 (unused)


If the module xlonpci.o does not appear or if you want to check how many  **XLON[®] PCI** adapters have been recognized you can switch to the console indicating the kernel messages. If this console is not available the kernel messages can also be found in the file /var/log/. To remove the device driver module from the kernel please use the following order:

```
# sbin/rmmod xlonpci
```

To use the  **XLON[®] PCI** adapter in your own application you must create a „Character Device File“. Please use the following orders:

```
# mknod /dev/pclta c 40 0  
# chmod 666 /dev/pclta
```


In the first line a „Character Device File“ called „pclta“ is created. This name enables access to the device driver. The parameter „c“ defines that the device driver works unbuffered. It is

followed by the major- and minor device numbers of the device driver which enables access to the device. In this case it is the device driver for the  adapter.

If you are intending to use more than one  adapter a „Character Device File“ has to be created for each physical LON channel. Please use the below series of orders (in this case two  adapters).

```
# mknod /dev/pcltaa c 40 0
# chmod 666 /dev/pcltaa
# mknod /dev/pcltab c 40 1
# chmod 666 /dev/pcltab
```

The device /dev/pcltaa appears as first LON channel, the device /dev/pcltab stands for the second LON channel. The example shows that the minor device number stand for the specific LON channel. This index starts with zero and is raised by one for each additional LON channel.

For the „Character Device File“ you can basically use any name. Under Linux applications the name „pclta“ is used as standard. It is also possible to create several „Character Device Files“ for one  adapter.

If you want to automatize loading and unloading of the device driver the orders above can be incased in Shell-Scripts or added to the rc-Scripts.

3.3.4 Driver update



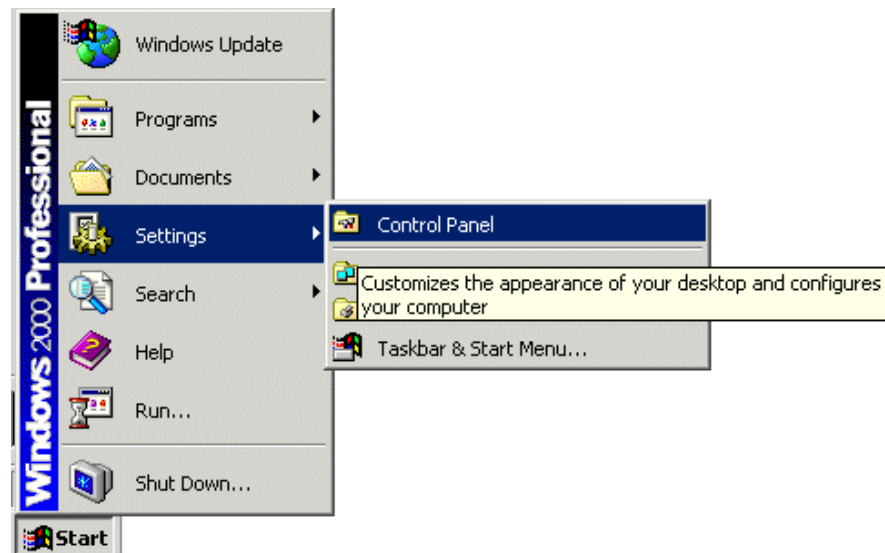
Find the current drivers for download under:

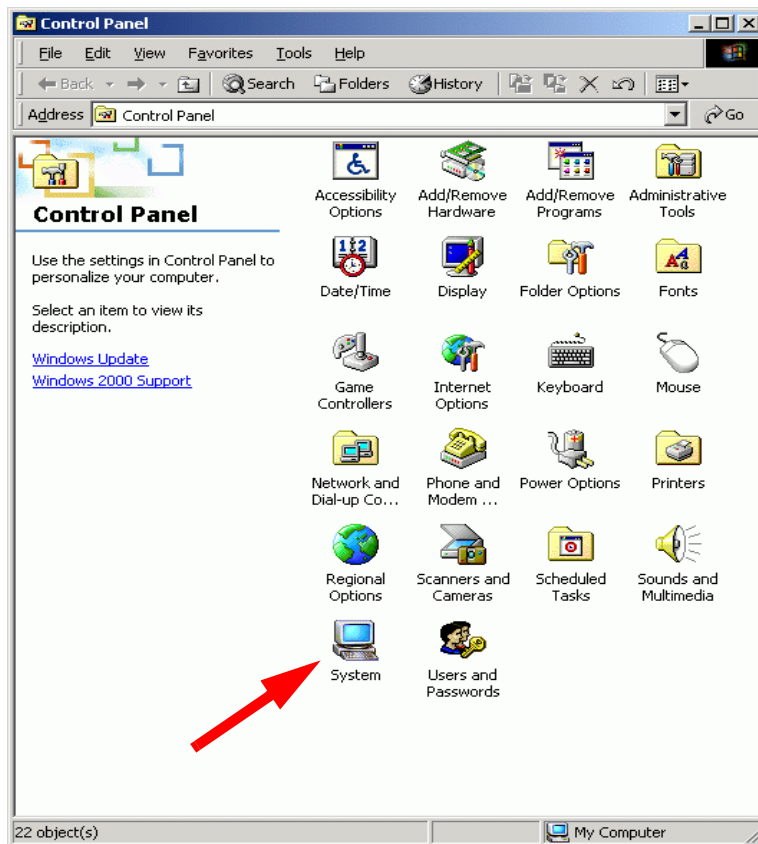
www.xlon.de

- Download the driver corresponding to your operating system and store it at any place

3.3.4.1 Windows

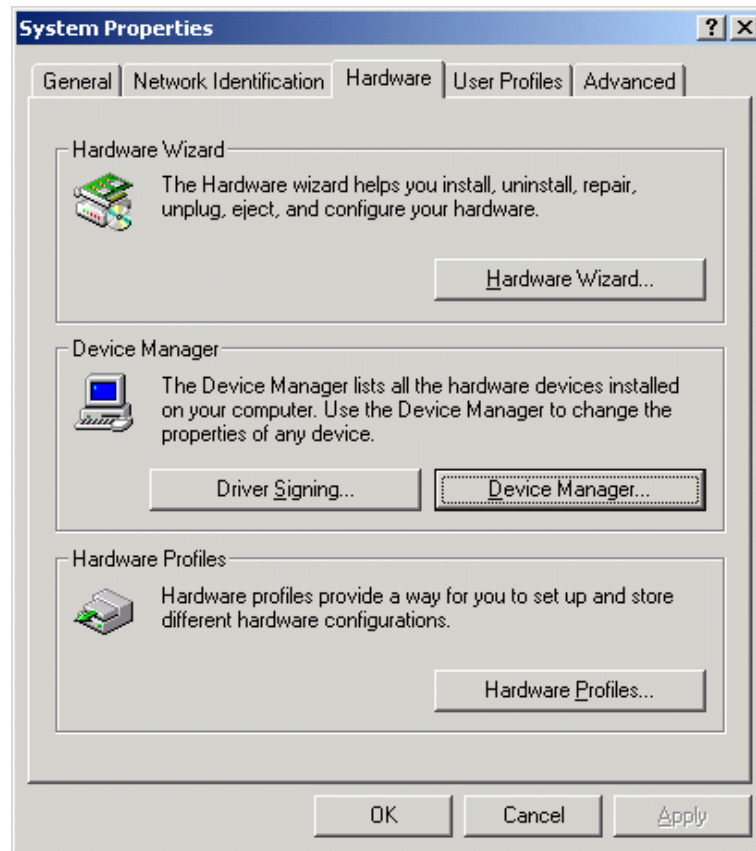
- Open the Control Panel
 - Click on „Start“
 - Go to „Settings“ with the mousepointer
 - Click on „Control Panel“



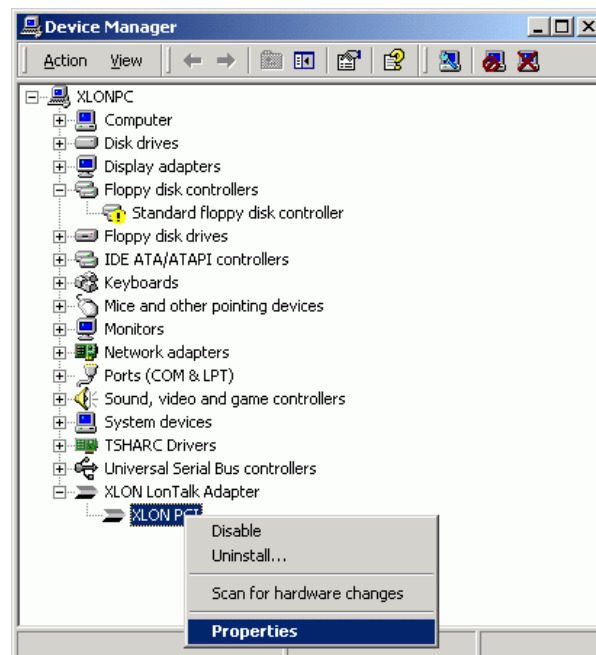


- Select „System“ by doubleclick

- Go to tab „Hardware“

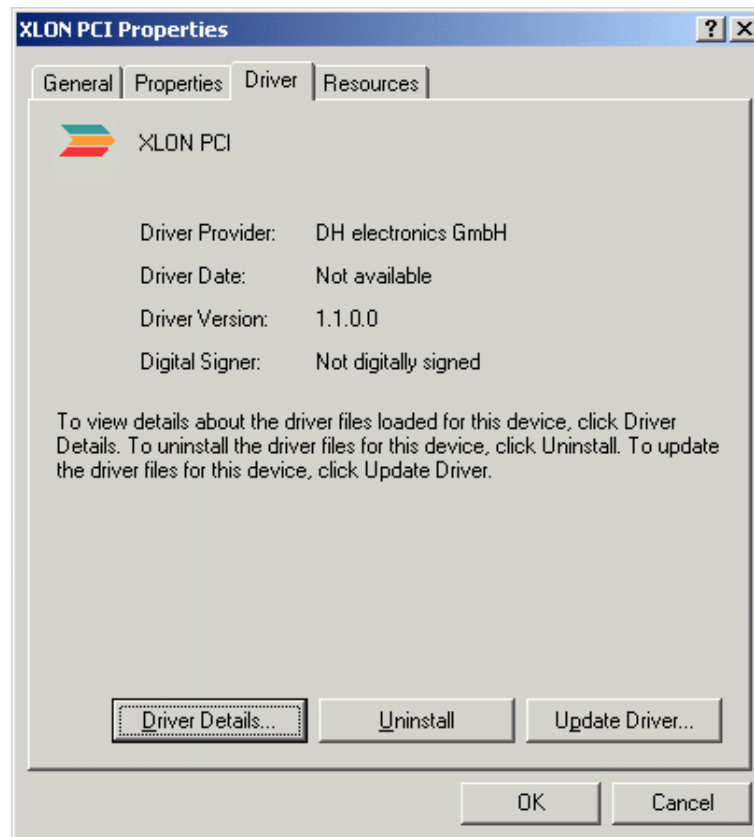


- Open the „Device Manager...“ by clicking the mouse



- Open the „Properties“ window of the  adapter (right mouse button)

- Click on tab „Driver“



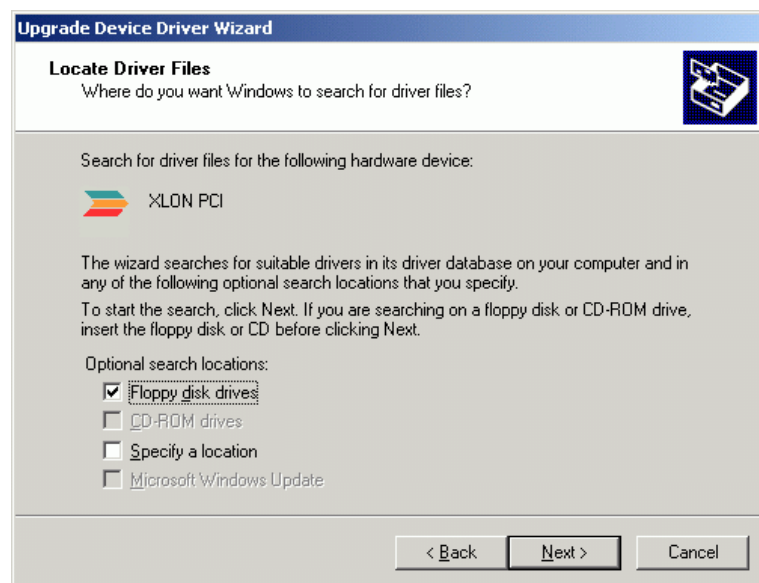
- Check if the driver version installed on your system is lower than the version you are intending to install.
- Click on „Update Driver...“
- The wizard for driver upgrade opens
- Follow the instructions displayed on the screen



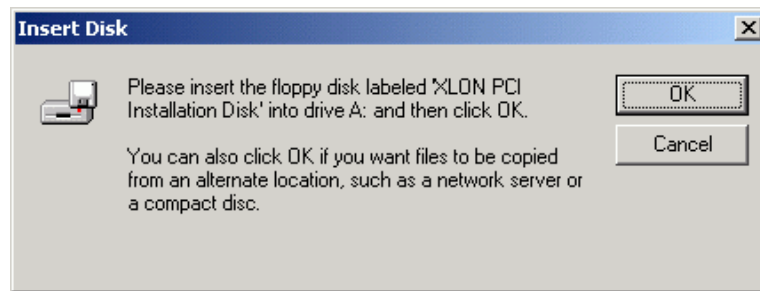
- Click on „Next>“



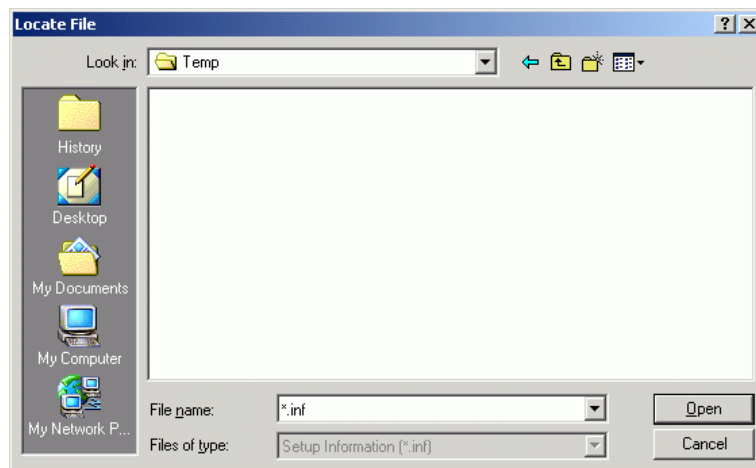
- Click on „Next>“



- Select „Specify a location“ and deactivate all other options
- Click on „Next>“



- Click on „OK“



- Search for the downloaded driver in the „Explorer“ window
- Highlight the driver and click on „Open“
- The rest of the driver actualisation follows the procedure of initial installation


3.3.4.2 Windows CE 3.0

Driver update corresponds to initial installation as described in chapter 3.3.2.


3.3.4.3 Linux

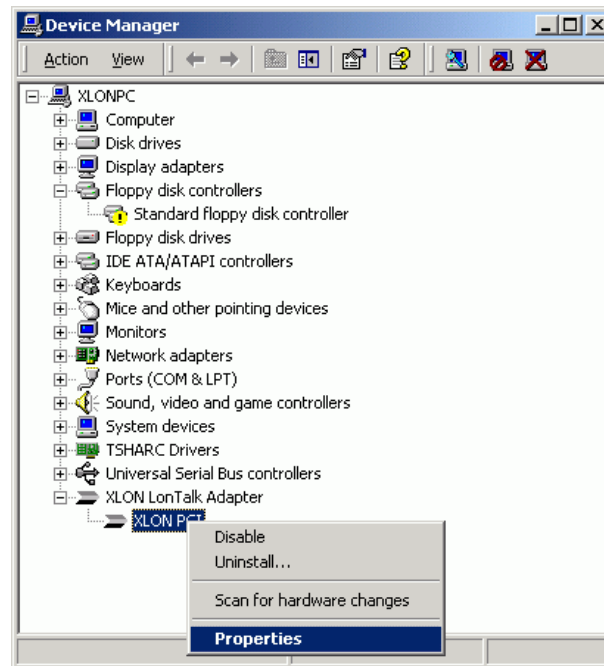
Driver update corresponds to initial installation as described in chapter 3.3.3.

3.4 Software De-installation

De-installation of software is not necessary. Simply remove the  adapter. Find information in chapter 3.2.

4 Configuring and Testing

- Start the device manager and open the „Properties“ window of the  adapter. Compare chapter 3.3.4

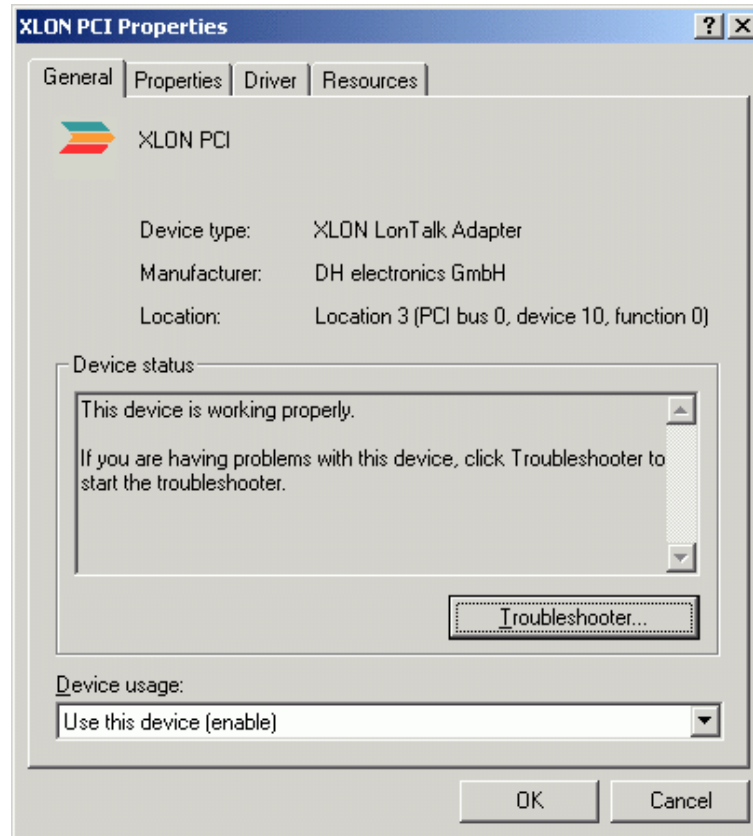


- The „Properties“ window opens

4.1 Check of settings under Windows

4.1.1 General settings

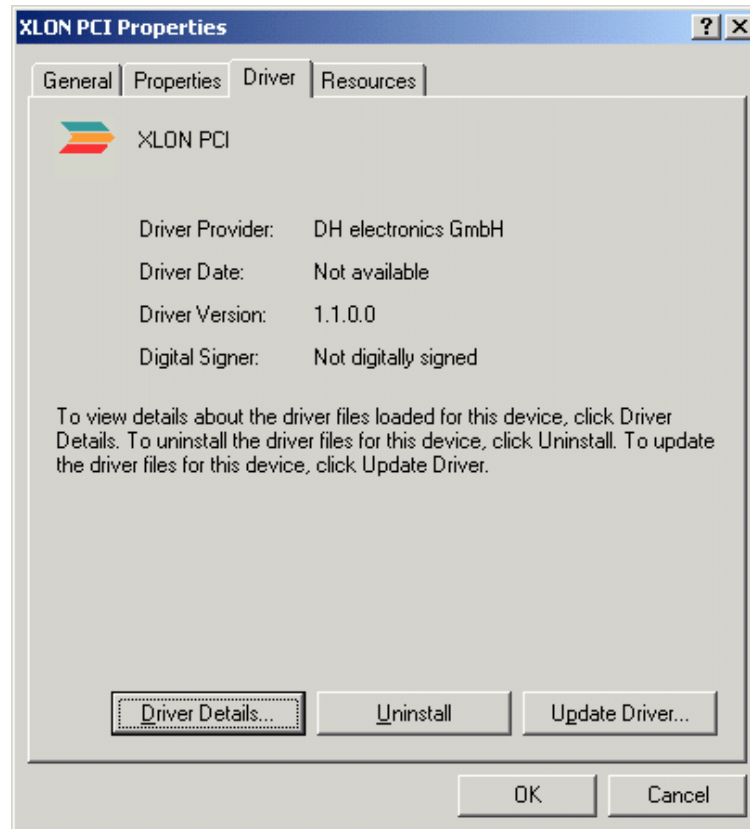
Tab General:



☞ Important: „This Device is working properly“ has to be indicated in the item „Device status“ !

4.1.2 Driver information

- Click on tab „Driver“

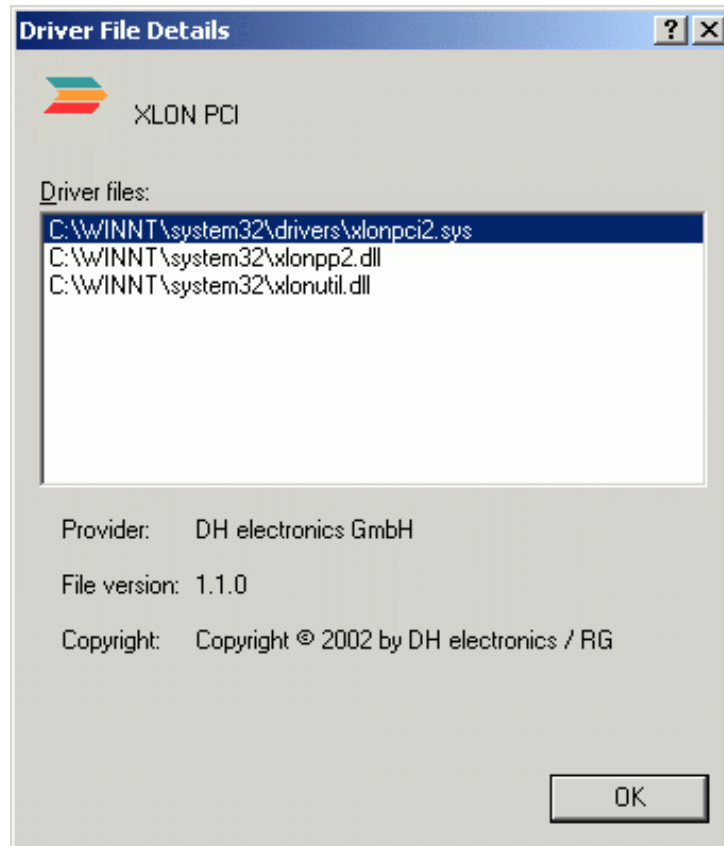


Data of the current driver is indicated.



The indicated version number behind „Driver Version“ is not the version number of the device driver file but only the number of the driver installation!

- To find out the version number of the drivers click on „Driver Details...”



Selecting one of the indicated files you get information about the Provider, File version and Copyright.

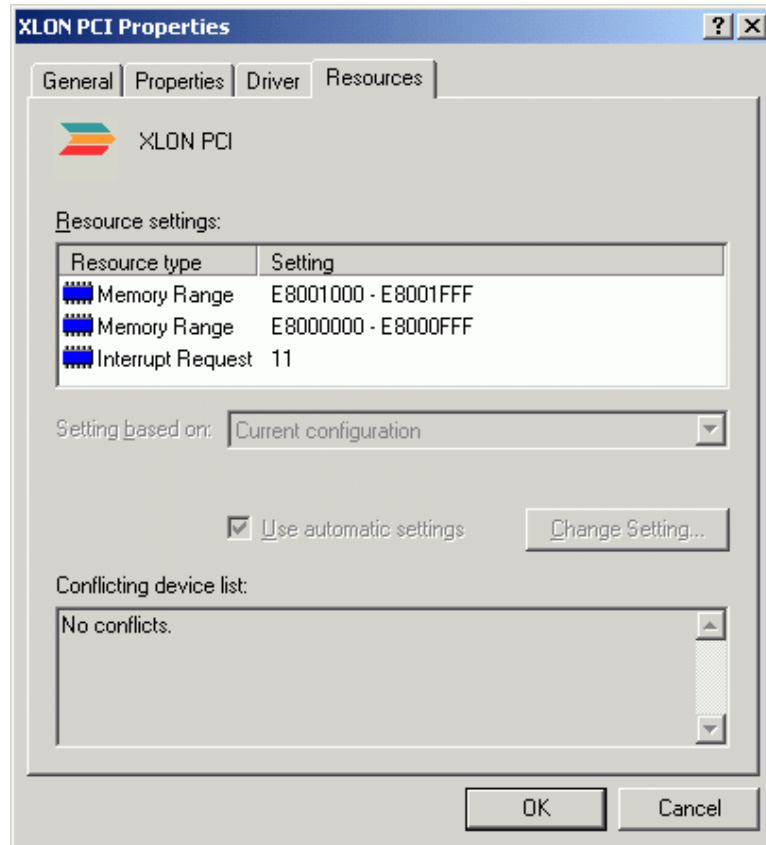


For enquiries concerning support you should always hold ready this information!

Find information about buttons „Uninstall“ and „Update Driver...” in chapter 3.3.

4.1.3 Occupied Hardware-Resources

- Click on tab „Resources“

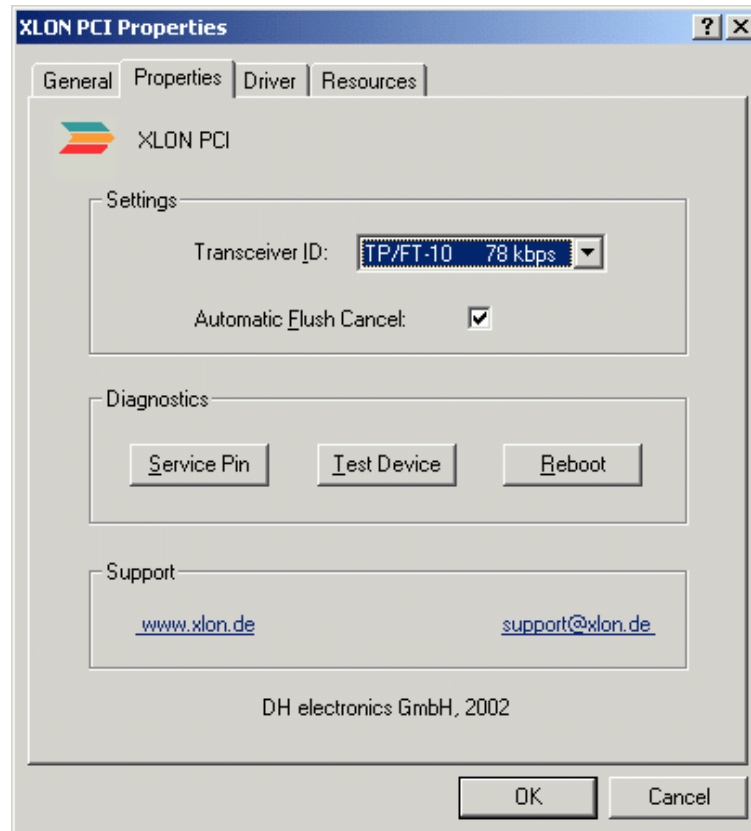


In this window you find information about occupied hardware resources.

No device conflicts must be shown in the Conflicting device list!

4.1.4 Properties of the XLON[®] PCI adapter

- Click on tab „Properties“




 The current configuration of the  XLON[®] PCI adapter for „Transceiver ID“ and „Automatic Flush Cancel“ is indicated.

The following settings are made by state of delivery:

Transceiver ID: TP/FT-10 78kbps


Automatic Flush Cancel: activated

Alteration of Transceiver ID

- Make sure that possible Transceiver settings match with the hardware. The  XLON[®] PCI adapter will not work with incorrect settings!
- Click on the „Triangular button“ (▼) beside the display field
- Select the intended ID with the left mouse button in the „Select“ menu
- Confirm by clicking the „OK“ button.

Activating/De-activating Automatic Flush Cancel

After each reset of the  XLON[®] PCI adapter communication via the LonWorks[®] network is locked by standard. Receiving or sending of data is only possible after a Flush Cancel command has been sent to the  XLON[®] PCI adapter.

If the Automatic Flush Cancel function is activated the device driver sends a Flush Cancel command to the  XLON[®] PCI adapter after every reset. Communication via the LonWorks[®] network is then possible.



Please notice that if the Automatic Flush Cancel function is de-activated a Flush Cancel Command has to be sent from the application to the  XLON[®] PCI adapter!

- Click on the field beside „Automatic Flush Cancel“ with the left mouse button.
- : activated
- : de-activated
- Confirm by clicking the „OK“ button

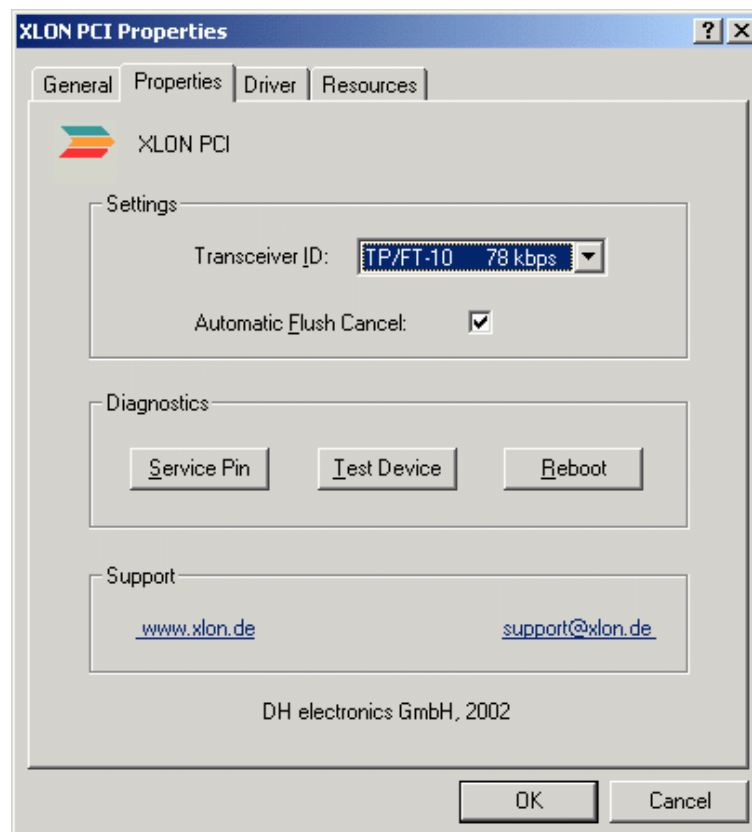
The function of the buttons „Service Pin“, „Test Device“ and „Reboot“ is explained in chapter 4.2.

If you need additional information or product support please make use of the links listed under „Support“.

4.2 Test of the XLON[®] PCI adapter under Windows

4.2.1 Diagnosis by Software

- Open the „Properties“ window of the  XLON[®] PCI adapter as described in chapter 4
- Open the tab „Properties“




For the device test you will find three buttons under the group „Diagnostics“:

Service Pin:


By clicking the button „Service Pin“ the  XLON[®] PCI adapter will send a Service Pin message to the LonWorks[®] network. Pressing the Service Pin button has the same effect (compare chapter 3.2).

Test Device:


By clicking the button „Test Device“ the  XLON[®] PCI adapter can be tested:

After clicking the button the yellow Service LED of the adapter should flash for a moment. Successful communication with the  XLON[®] PCI adapter will be indicated.

Reboot

Clicking the „Reboot“ button will restore state of delivery of the  XLON[®] PCI adapter. This action should only be carried out by experienced users.


4.2.2 Diagnosis by LED

As described in chapter 3.2 the  adapter is fitted with two LED's for visualisation. A green LonWorks[®] network traffic LED and a yellow Service-Pin LED.

Green LonWorks[®]-network-traffic LED:



By blinking (frequency is not defined) the green LED indicates whether there is data traffic on the LonWorks[®]-network.




Initial state is not defined, this means that in the basic state the LED can be activated or deactivated. Moreover this LED can blink with the frequency of the yellow Service-Pin LED (1,25 Hz). This happens if no application has accessed the  adapter before.

Yellow Service-Pin LED:






The yellow LED indicates the state of the Neuron Processor Service Pin line.

- If the device driver is installed correctly and access to the  adapter is made from an application the LED is de-activated.
- If the LED blinks with a frequency of 1/2 Hz the state of the  adapter is „Unconfigured“.
- If a device test according to the „Properties“ window is made (compare chapter 4.2.1) or a „Reset“-command from an application is executed the LED flashes for a moment



If the LED blinks with a frequency of 1,25 Hz and an application has no access to the  adapter there is a problem with the driver installation.

If access from an application is not possible and the state of the LED doesn't change probably no device driver has been installed.

Yellow Service-Pin LED	Description
Constantly off	- Successful access from the application to the  adapter or - No driver installed
Constantly on / Constantly off	- No driver installed
Flash frequency 1,25 Hz	- No application has accessed the  adapter
Flash frequency 1/2 Hz	- State of the  adapter is „Unconfigured“, this means the  adapter has no network address
Flashes for a moment	- A „Reset“ to the  adapter has been made or - A device test has been made

5 Technical Specification


5.1 Hardware

5.1.1 General Information

Bus Interface		PCI conform, in accordance with PCI specification Revision 2.2
Network Connection	FTT-10A: RS485:	2-conductor Weidmueller connector with tension clamp connection and screw flanges 3-conductor Weidmueller connector with tension clamp connection and screw flanges
Power Supply		Via the PCI bus
Service Pin Function		Controlled by host or Service button
Configuration State		Displayed on host and via Service LED
Network Transceiver		FTT-10A or RS485 (integrated)
Network-Topologies		FTT-10A: Free Topology and Link Power RS485: Twisted Pair
Power Supply Data		5 V DC, $\pm 5\%$, 80 mA typical and 3.3VDC, $\pm 5\%$, 40 mA
Operating temperature		0°C to +70°C (+32°F to +158°F)
Non-operating temperature		-45°C to +85°C (-49°F to +185°F)
Maximum humidity		90%@+50°C (90%@+122°F), non condensing
EMI		EN55022 Level B, EN61000-4-2, EN61000-4-4, EN50140, EN50141
Listings		CE and FCC
Processor		Neuron [®] 3150 - Chip@10 MHz
Dimensions		127 x 55,7 mm (5" x 2.19") (length x width)
Weight		100 g

The hardware of the  adapter supports up to 127 adapters per PCI Bus System in the PC (Multiple Device Support).

5.1.2 Plug Connection

As plug connection for the LonWorks®-connection of the  adapter a Weidmueller plug connection (Series BL3.5) is used.

Type of network	Pin #	Weidmueller Termination	Order number
Free Topologie FTT	2 pin	BL3.5/2F SN OR	160664
Twisted Pair RS485	3 pin	BL3.5/3F SN OR	160665

The cross-section of the line which can be clamped at maximum is 1.5 mm².

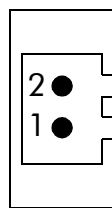
Find order information as well as further detailed information about this connector under:

www.weidmueller.de

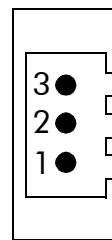
Plug Allocation:

Pin allocation LonWorks® connector

FTT-10A

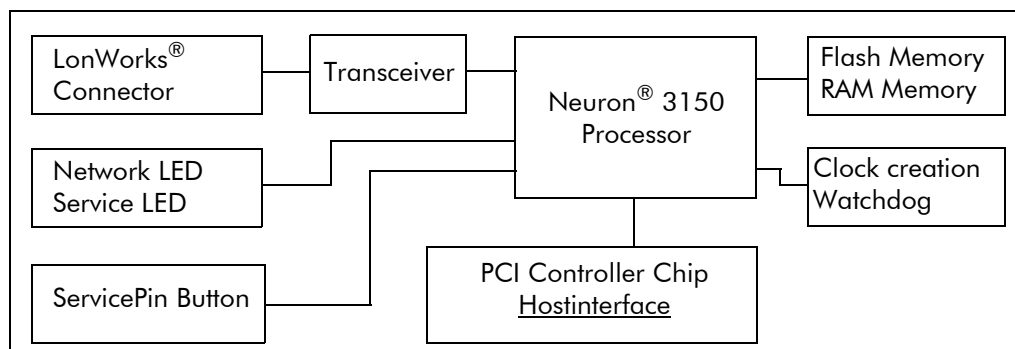


RS485



Pin	FTT-10A	RS485
1	NET A	RS485 A
2	NET B	RS485 B
3	not existing	GND


5.1.3 Block model




5.1.4 Technical Details of the Hardware


Connection to the Host System via PCI Bus:

Connection to the PCI Bus follows PCI-Specification Revision 2.2. The width of data bus is 32 Bit.

The  adapter can be operated from a 5 volt as well as from a 3,3 volt PCI signaling environment.

The  adapter is completely compatible to Plug&Play. Within the system it is identified by the DH electronics GmbH Vendor ID ,0x17E9' which is given by the ,PCI Special Interest Group' and the Device ID ,0x0001'.

LonWorks®-Network Interface:

There are two different Transceiver variants for the LonWorks®-Network at disposal: Free Topology Transceiver FTT-10A (2 pin connector) and RS485 Transceiver (3 pin connector). The transmission rate of the FTT-10A Transceiver is 78,5 kBit/s. If the  adapter is fitted with RS485 Transceiver different transmission rates can be set by software (compare chapter 4.1.4). The maximum datarate is 250kBit/s.


Neuron Processor Core:

A 3150® Neuron Processor with external storage interface is used. A reprogrammable Flash memory is used as program memory. A SRAM memory is used as data memory. The Neuron Processor is connected to the PCI-Controller-module by the Neuron ,Parallel IO Model'.

Address Register of the Neuron Processor:

Type of memory	Address Zone	Memory Size
ROM-memory	0x0000 - 0xC2FF	49919 Byte (48.75 kB)
RAM-memory, read and write	0xC300 - 0xE6FF	9215 Byte (9.00 kB)
IO range for Interrupt-Generation	0xE700 - 0xE7FF	
Reserved Neuron® Processor intern	0xE800 - 0xFFFF	

5.1.5 Supported Transceivers





Generally the Transceiver configuration doesn't have to be changed. If required it can be changed in the Device Manager under Microsoft Windows Desktop operating systems. For operating systems like Microsoft Windows CE or Linux manual setting of the Transceiver ID into certain configuration files might be necessary. Find more information about this in chapter „Driver Installation“ of the corresponding operating system. The register below shows the LON Transceiver IDs which are generally supported by the  adapter. Depending on the Transceiver (TP/FT-10 oder TP-RS485) which is physically existing on the adapter not all of the Transceiver operating types are supported.


ID	Name	Media	Network Bit Rate
04	TP/FT-10	Free topology/link power	78 kbps
05	TP-RS485-39	RS-485 twisted pair	39 kbps
12	TP-RS485-78	RS-485 twisted pair	78 kbps

6 Software Access

6.1 Application Interface under Windows

6.1.1 LNS-applications

If you want to access the  **XLON[®] PCI** adapter via a LNS application as network interface you only have to indicate the  **XLON[®] PCI** adapter you are intending to use. The device driver supports up to 127  **XLON[®] PCI** adapters per PC-System, this means that several  **XLON[®] PCI** adapters can be installed in parallel on your system. The adapters are responsive by different device names and they distinguished by there Network Interface Number.

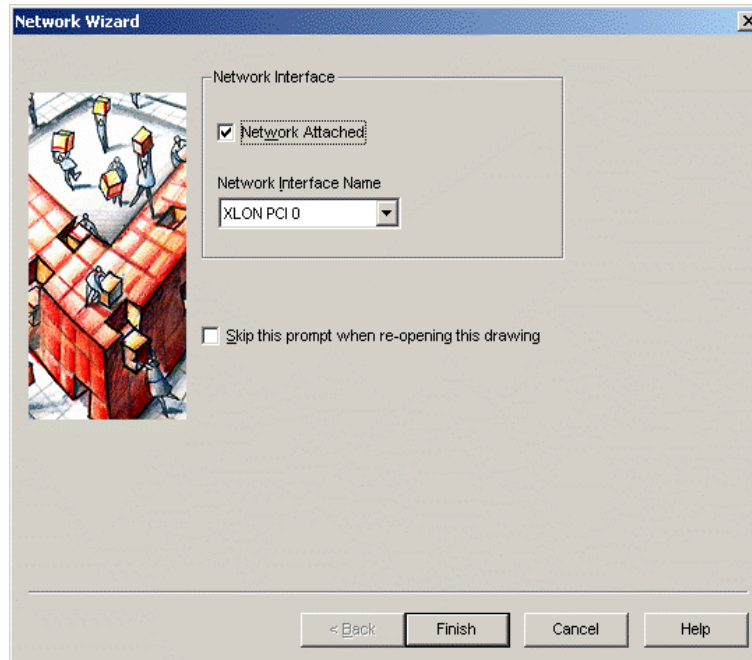
The LNS-application shows a „Select“ menu for the Network Interface Name. Find the  **XLON[®] PCI** adapter under the Network Interface Name:

XLON PCI x

x:  **XLON[®] PCI** Network Interface Number

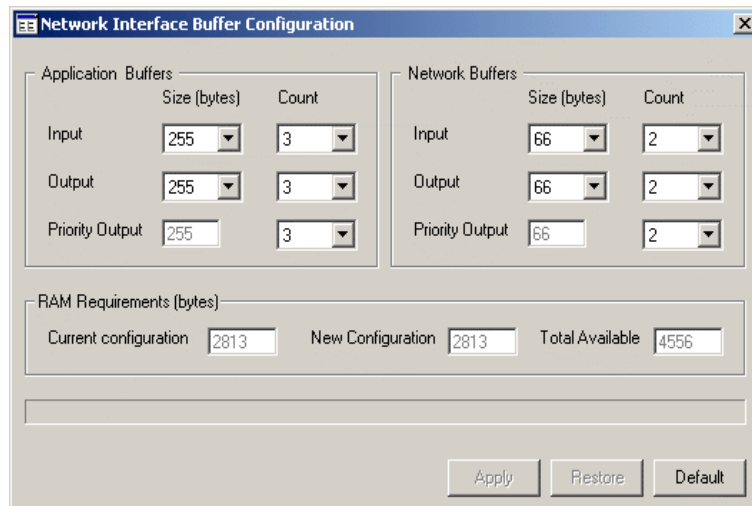
For example: XLON PCI 0 => first  **XLON[®] PCI** adapter installed on the system.

The Window Screen below shows configuration in the application ‚LonMaker for Windows‘:




6.1.2 Configuration of the Network Interface Buffer

The Network- and Application Buffer of the Neuron Processor can be changed by the Echelon LNS Plug-In ,Network Interface Buffer Configuration' (www.echelon.com).



6.1.3 Programming your own application

Based on the information of the ,LonWorks Host Application Programmer's Guide' it is possible to program your own LonWorks Host application. The programming instruction can be obtained from Echelon (www.echelon.com).

The process of coding access to the device driver of the  adapter is explained in this chapter. As the Echelon Standard Driver Interface of systems based on Windows 98/ME and Windows 2k/XP is different, you have to distinguish between the two types of operating systems.

All functions listed below are Windows 32-bit API-functions.

Find a more detailed example for download under the following internet site:

<http://www.xlon.de>

6.1.3.1 Opening the device driver

Before the driver can be accessed it has to be opened. If the device driver has been opened successfully it delivers a handle which enables access to the driver.


The name of the device driver for the  adapter is: ,\\.\xlonpci0'

This name can also be read from the Registry by an Alias. If the Network Interface which must be used is intended to be for selection it is advisable to turn access selectable from the Registry by an Alias Name.

For this device indicated by registration code:

<HKEY_LOCAL_MACHINE\SOFTWARE\LonWorks\DeviceDrivers\>

can be offered for selection.

Find the  **XLON[®] PCI** adapter under the Alias Name: XLON PCI x

x:  **XLON[®] PCI** Network-Interface-Number

For example: XLON PCI 0 => first  **XLON[®] PCI** adapter installed in the system.

Necessary Command and Type Definitions:

Commands under Windows98/ME:

```
#define LDV_Acquire          1    // Sign access to the driver
#define LDV_Release         2    // Release access to the driver
#define LDV_Register_Event_Handle 7 // Register an event handle
#define LDV_Read            10   // Read from the driver
#define LDV_Write           11   // Write to the driver
```

Type Definition of the application buffer:

```
#define MAXLONMSG          253 // Maximum length of message data
typedef struct APILNI_Message_Struct
{
    BYTE NiCmd;                // Network Interface Command
    BYTE Length;               // Size of ExpAppBuffer
    BYTE ExpAppBuffer[MAXLONMSG]; // Buffer for Data
} APILNI_Message;
```

Structure of the application buffer (API LNI Message)

	Command	2 Byte
	Length	
Length	Begin of Data	3 Byte
	Network Adress	11 Byte
	Data	variable Length


Definition of a Access Handle

1.) HANDLE* phandle = new HANDLE; // Handle to access the device driver

Definition of the Application Buffers

2.) APILNI_Message* ni_in_msg = new APILNI_Message; // NSI Message In structure
 APILNI_Message* ni_out_msg = new APILNI_Message; // NSI Message Out structure

Windows98/ME:

The function ,CreateFile' opens the device driver of the  adapter and passes back a handle for access to the device driver. ,\\.\xlonpci0' has to be used as name for the device driver.

```
2.) *pHandle = CreateFile(“\\\\.\xlonpci0”, GENERIC_READ | GENERIC_WRITE, 0, 0, OPEN_EXISTING, 0, 0);
```

Possible error codes passed back by the driver:

error:	Not enough memory for allocation of driver buffer
error code:	ERROR_NOT_ENOUGH_MEMORY


Under Windows98/ME operating systems the command 'LDV_Aquire' has to be obeyed after opening the driver. Hereby the device driver is informed that the driver has been accessed.

```
3.) DeviceIoControl(*pHandle, MAKELONG(LDV_Acquire, 0), &inBuf, sizeof(char), &RetInfo, sizeof(RetInfo), &nBytesReturned, NULL);
```

Possible error codes passed back by the driver:

error:	none
--------	------

Windows2000/XP:

The function ,CreateFile' opens the device driver for the  adapter and passes back a handle for access to the device driver. ,\\.\xlonpci0' has to be used as name for the device driver.

```
2.) *pHandle = CreateFile(“\\\\.\xlonpci0”, GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ | FILE_SHARE_WRITE, (LPSECURITY_ATTRIBUTES) NULL, OPEN_EXISTING, 0, (HANDLE) NULL);
```

Possible error codes passed back by the driver:

error:	Not enough memory for allocation of the driver bufferr
error code:	ERROR_NOT_ENOUGH_MEMORY

6.1.3.2 Registration of an Event-Handle

For communication between driver and application a common Event-Handle can be registered. Therefore the application has to demand a Handle from the operating system. The application subsequently delivers the handle to the driver. When the driver has got data for the host application it creates an event.

Windows98/ME:

Creation of an Synchronisation Event Handle:

```
1.) CreateCommonEvent(&hEventR3, &hEventR0, FALSE, FALSE);
```

Delivery of the Event Handle to the Driver:

```
2.) DeviceIoControl(pHandle, MAKELONG(LDV_Register_Event_Handle,0), hEventR0, sizeof(HANDLE), &RetInfo, sizeof(RetInfo), &nBytesReturned, NULL);
```

Windows2000/XP:

Hereby another Event mechanism is in action. Access is made via Overlapped IO.

6.1.3.3 Reading of data from the device driver

In order to read data the command ,LDV_Read' in the Windows API function ,DeviceIoControl' is used under Windows98/ME.

Under Windows 2000/XP the Windows API function ,ReadFile' is used.

Windows98/ME:

```
DeviceIoControl(pHandle, MAKELONG(LDV_Read, 0), NULL, sizeof(char), ni_in_Msg, length, &nBytesReturned, NULL);
```

Possible error codes passed back by the driver:

error:	No LDV_ACQUIRE was executed before the driver has been accessed.
error code:	ERROR_ACCESS_DENIED

Windows2000/XP:

```
ReadFile( pHandle, ni_in_Msg, length+1, &nBytesReturned, NULL );
```

Possible error codes passed back by the driver:

error:	Overlapped IO and no data available.
errorcode:	STATUS_PENDING
error:	Non Overlapped IO and no data available.
errorcode:	STATUS_UNSUCCESSFUL

6.1.3.4 Writing of data on the device drivers

In order to write data the command ,LDV_Write' in the Windows API function ,DeviceIoControl' is used under Windows98/ME. Under Windows 2000/XP the Windows API function ,WriteFile' is used.

Windows98/ME:

```
DeviceIoControl(pHandle, MAKELONG(LDV_Write, 0) , ni_out_Msg, length, NULL, sizeof(char), &nBytesReturned, NULL);
```

Possible error codes passed back by the driver:

error:	No LDV_ACQUIRE was executed before the driver has been accessed.
error code:	ERROR_ACCESS_DENIED
error:	No free application buffer available in the driver
error code:	ERROR_NOT_ENOUGH_MEMORY
error:	Data record sent before application has been too big
error code:	ERROR_ACCESS_DENIED

Windows2000/XP:

```
WriteFile( pHandle, ni_out_Msg, length, &nBytesReturned, NULL );
```

Possible error codes passed back by the driver:

error	Application Buffer exhausted for writing
error code:	ERROR_NOT_ENOUGH_MEMORY

6.1.3.5 Closing the device driver

If you intend to end the application the driver has to be closed.

Windows98/ME:

The command ‚LDV_Release’ has to be obeyed before the driver can finally be closed. Hereby the occupancy of the device driver is cancelled.

- 1.) DeviceIoControl(pHandle, MAKELONG(LDV_Release, 0), &inBuf, sizeof(char), &RetInfo, sizeof(RetInfo), &nBytesReturned, NULL);


The driver has to be closed subsequently.

- 2.) CloseHandle(pHandle);


Windows2000/XP:

- 2.) CloseHandle(pHandle);

6.1.3.6 Important Programing Information

If you program your own application for the  adapter a Program ID has to be programmed into the adapter during the process of initialisation. The Program ID which has to be used can be defined by the application.

The network and application buffers of the Neuron[®] Processor can be changed. The maximum valid amount of bytes for all buffers used must not be higher than 4556 bytes. Admissible buffer settings should be located by the LNS Plug-In ‚Network Interface Buffer Configuration’ (compare chapter 6.1.2.)

In some cases the  adapter can lose its function because of false buffer settings. This can only be canceled by a ‚Reboot’ of the adapter or in some cases it is not possible to revoke.

If the RS485 variant requires a special Transmission Rate which can not be set in the „Properties” window (compare chapter 4.1.4) the Transceiver ID ‚Custom Transceiver’ must be parameterized in the „Properties” Window. Please notice that then the application is responsible for correct settings of the Transceiver and the Transmission Rate.

6.2 Application interface under Windows CE 3.0

For creating a C/C++ LON Host Application under Windows CE 3.0 basically the documentation under chapter 6.1.3 can be used. The Application Programming Interface (API) for access to device driver under Desktop Windows and Windows CE 3.0. however differs.

For access from an own C/C++ LON Host Application to the device driver under Windows CE 3.0 the following Standard-Operating-System-Commands (Windows CE 3.0 API) for „Stream Interface Devices“ are available. The following subsections offer a survey of different API-functions. For a more detailed description look up the Windows CE 3.0 documentation which can be found in the MSDN Library or in the Microsoft Windows CE Platform Builder 3.0 Library.

CreateFile()	ReadFile()
CloseHandle()	DeviceIoControl()
WriteFile()	GetLastError()

6.2.1 CreateFile()

Prototype:

```
HANDLE CreateFile( LPCTSTR lpFileName,  
                  DWORD dwDesiredAccess,  
                  DWORD dwShareMode,  
                  LPSECURITY_ATTRIBUTES lpSecurityAttributes,  
                  DWORD dwCreationDisposition,  
                  DWORD dwFlagsAndAttributes,  
                  HANDLE hTemplateFile );
```

Description:

This function opens the device driver which is specified by „lpFileName“. The name of device driver is composed of the device prefix and the device index followed by a doublepoint, for example "LON1: ", "LON2: ", "LON3: ", and so on. Chapter 3.3.2.1 delivers detailed information about the composition of the device driver name . Find more information about further function parameters and about the function Create File() in general in the Windows CE 3.0 documentation.

Example:

```
HANDLE myHandle;  
  
myHandle = CreateFile( TEXT("LON1:"),  
                      GENERIC_READ | GENERIC_WRITE,  
                      0x00, NULL  
                      OPEN_EXISTING,  
                      FILE_ATTRIBUTE_NORMAL,  
                      NULL )
```

Return value:

If the device driver has been opened successfully a Handle is passed back to the device driver. By this Handle further operations can be made on the device driver or the device driver can be closed again. In case of an error INVALID_HANDLE_VALUE will be passed back. More detailed information about error cause can be obtained by calling the function GetLastError().

Error cause:

Possible error causes for a failure of this function are:

A device driver which has not been loaded, a false name of the device driver, a device driver which has not been closed orderly beforehand or function parameters which have not been set correctly.

6.2.2 CloseHandle()

Prototype:

```
BOOL CloseHandle( HANDLE hObject );
```

Description:

This function closes the device driver which is specified by the Handle „hObject“ . As function parameter the handle passed back after successfully calling the function Create File() has to be passed to the device driver. Find more detailed information about the function CloseHandle() in the CE 3.0 documentation.

Example:

```
BOOL bResult;  
  
bResult = CloseHandle( myHandle );
```

Return value:

If the device driver has been closed successfully the value „TRUE“ is passed back otherwise the value „FALSE“ is passed back. In this case detailed information about error cause can be obtained by calling the function GetLastError(). After a successful call of CloseHandle() the handle that has been passed on is not valid any longer and can no longer be used for operations on the device driver.

Error causes:


An invalid Handle on the device driver might be the error cause of a failure of this function.

6.2.3 WriteFile()


Prototype:

```
BOOL WriteFile( HANDLE hFile,  
                LPCVOID lpBuffer,  
                DWORD nNumberOfBytesToWrite,  
                LPDWORD lpNumberOfBytesWritten,  
                LPOVERLAPPED lpOverlapped );
```

Description:

By this function data from a LON application is written on the device driver and consequently on the -Network Interface. Calls made by this function are asynchronous. This means that the function returns as soon as data has been taken into the internal buffer of the device driver or an error has occurred. Processing of data is then running in parallel to the

LON application in the background.

Each -Network Interface is specified by its „hFile“. The cursor „lpBuffer“ must point to a data structure of the type APILNI_Message which is also noted as Application Layer Buffer. The size of this data structure is variable. The structure is shown in the point below. For each specific call the size of this variable data structure is set by the parameter „nNumberOfBytesToWrite“. By the parameter „lpNumberOfBytesWritten“ the actual amount of bytes which has been transmitted to the Network Interface is passed back. In case of success these two values are identical and not zero. The parameter „lpOverlapped“ has no usage under Windows CE 3.0. Find detailed information about the different function parameters and about function WriteFile() in general in the Windows CE 3.0 documentation.

Application Layer Buffer:

The below C-Code defines the data type „APILNI_Message“ for Application Layer Buffer as specified in the Echelon NSI Firmware User’s Guide. Find details on the structure of the Structure Element „ExpAppBuffer[]“ in the „LonWorks Host Application Programmer’s Guide“.

```
#define MAXLONMSG 253

typedef struct APILNI_Message_Struct {
    BYTE NiCmd;                // NSI command
    BYTE Length;              // Size of ExpAppBuffer
    BYTE ExpAppBuffer[MAXLONMSG]; // Message data
} APILNI_Message;
```

The graphics below shows again the structure of the Application Layer Buffer

command	2 Bytes	Application Layer Header
length		
message header	3 Bytes	ExpAppBuffer[MAXLONMSG], maximum size 253 Bytes
network address	11 Bytes	
message data	variable length	

Example:

```
BOOL bResult;
DWORD dwBytesWritten;
APILNI_Message lni_msg = { niRESET, 0x00 } // Buffer with Reset Command to NSI

bResult = WriteFile( myHandle,
                    (LPCVOID)&lni_msg,
                    0x02,
                    &dwBytesWritten,
                    NULL );
```

Return value:

If the writing operation on the device driver was successful the value „TRUE“ is passed back otherwise the value „FALSE“ is passed back. In this case detailed information about error cause can be obtained by calling the function GetLastError().

Error causes:

Possible error causes for a failure of this function:
An invalid Handle on the device driver, an Application Layer Buffer which has not been built



up correctly or the parameter „nNumberOfBytesToWrite“ which is located outside the valid zone or does not conform with the actual length of the Application Layer Buffer.


6.2.4 ReadFile()

Prototype:

```
BOOL ReadFile(          HANDLE hFile,  
                      LPVOID lpBuffer,  
                      DWORD nNumberOfBytesToRead,  
                      LPDWORD lpNumberOfBytesRead,  
                      LPOVERLAPPED lpOverlapped );
```

Description:

By this function data is read from the device driver and consequently from the -Network Interface by a LON application. Calls made by this function are asynchronous. This means that the function returns as soon as data has been taken from the internal buffer of the device driver. If no data is available or if there has been an error the function also returns immediately. This means that data from the -Network Interface is not being waited for.

The -Network Interface is specified by its Handle „hFile“. The cursor „lpBuffer“ must point to a data structure of the type APILNI_Message which is also noted as Application Layer Buffer. The size of this data structure is variable. During a reading operation it should however be set at maximum value as the actual amount of data which has to be read is not known by time of calling the function. The structure is shown in the point below. For each specific call the size of this variable data structure is set by the parameter „nNumberOfBytesToRead“. By the parameter „lpNumberOfBytesRead“ the actual amount of bytes which has been transmitted to the Network Interface is passed back. In case of success this value equals „nNumberOfBytesToRead“ or is lower than that and differs from zero. The parameter „lpOverlapped“ has no usage under Windows CE 3.0. Find detailed information about the different function parameters and about function ReadFile() in general in the Windows CE 3.0 documentation.

Application Layer Buffer:

The below C-Code defines the data type „APILNI_Message“ for Application Layer Buffer as specified in the Echelon NSI Firmware User’s Guide. Find details on the structure of the Structure Element „ExpAppBuffer[]“ in the „LonWorks Host Application Programmer’s Guide“.

```
#define MAXLONMSG 253  
  
typedef struct APILNI_Message_Struct {  
    BYTE NiCmd;                // NSI command  
    BYTE Length;               // Size of ExpAppBuffer  
    BYTE ExpAppBuffer[MAXLONMSG]; // Message data  
} APILNI_Message;
```

The graphics below shows again the structure of the Application Layer Buffer.

command	2 Bytes	Application Layer Header
length		
message header	3 Bytes	ExpAppBuffer[MAXLONMSG], maximum size 253 Bytes
network address	11 Bytes	
message data	variable length	

Example:

```

BOOL bResult;
DWORD dwBytesRead;
APILNI_Message Ini_msg           // Application Layer Buffer for message from NSI

bResult = ReadFile(               myHandle,
                                  (LPCVOID)&Ini_msg,
                                  255,
                                  &dwBytesRead,
                                  NULL );
  
```

Return value:

If the reading operation on the device driver has been successful the value „TRUE“ is passed back. If the amount of read bytes passed back in „lpNumberOfBytesRead“ equals zero no data had been available. If the reading operation on the device driver has not been successful the value „FALSE“ is passed back. In this case detailed information about error causes can be obtained by calling the function `GetLastError()`.

Error causes:

Possible error causes for a failure of this function:
An invalid Handle on the device driver or a parameter „nNumberOfBytesToRead“ which is located outside the valid zone or does not conform with the actual length of the Application Layer Buffer.

6.2.5 Devicelo Control()

Prototype:

```

BOOL DeviceloControl( HANDLE hFile,
                      DWORD dwloControlCode,
                      LPVOID lpInBuffer,
                      DWORD nInBufferSize,
                      LPVOID lpOutBuffer,
                      DWORD nOutBufferSize,
                      LPDWORD lpBytesReturned,
                      LPOVERLAPPED lpOverlapped );
  
```

Description:


This function enables certain operations on the device driver which are not practicable by the functions listed above. At present these operations are „GetVersion“ und „ReadWait“ which can be called by the commands „IOCTL_XLON_GETVERSION“ or „IOCTL_XLON_READWAIT“.

The operation „GetVersion“ enables read out of a version from the device driver. The operation „ReadWait“ is the synchronous (blocking) variant of the function `ReadFile()`. These two functions which can be operated by `DeviceloControl()` are explained below.

6.2.5.1 „GetVersion“ by DeviceIoControl()

Description:

The operation „GetVersion“ is realized by the API-Function DeviceIoControl() and enables read out of a version code from a device driver. The IO-Control-Code for this operation is defined as „IOCTL_XLON_GETVERSION“. The driver version is coded by a DWORD. Each of the four bytes represents a decimal digit. The Major-Version is coded in Bit 16 to Bit 23 (Byte 2) and the Minor-Version in Bit 8 to Bit 15 (Byte 1) the remaining Bits (Byte 0 and Byte 3) have no meaning at the moment. A read out DWORD from 0x00010200 has the value 0.1.2.0 this means that the driver version is 1.2.

Calling DeviceIoControl() the -Network Interface has to be specified by its Handle „hFile“. The parameter „dwIoControlCode“ has to be occupied by the command „IOCTL_XLON_GETVERSION“. The parameters „lpInBuffer“ or „nInBufferSize“ are not needed. For the parameter „lpOutBuffer“ a pointer is passed on to a DWORD where the driver version is put later. In „nOutBufferSize“ the size of the DWORD is passed into Byte. In the parameter „lpBytesReturned“ the amount of Bytes which has been read is passed back. The parameter „lpOverlapped“ is not used under CE 3.0. Find detailed information about the specific function parameters and about the function DeviceIoControl() in general in the Windows CE 3.0 documentation.

Example:

```
#define IOCTL_XLON_GETVERSION (DWORD)0x01 // IOCTL-Code for command
                                           „GetVersion“

BOOL bResult;
DWORD dwVersion, dwBytesReturned;

bResult = DeviceIoControl( myHandle,
                          IOCTL_XLON_GETVERSION,
                          NULL, 0,
                          &dwVersion,
                          sizeof( dwVersion ),
                          &dwBytesReturned
                          NULL );
```

Return value:


If read out of the driver version from the device driver has been successful the value „TRUE“ is passed back otherwise the value „FALSE“ is passed back. In this case detailed information about the error cause will be obtained by calling the function GetLastError().



Error causes:

A possible error cause for a failure of this function is an invalid Handle on the device driver.

6.2.5.2 „ReadWait“ by DeviceIoControl()

Description:

The operation „ReadWait“ is the synchronous (blocking) variant of the function ReadFile(). By this operation a LON Application reads data from the device driver and consequently from -Network Interface. If there is no data in the internal buffer of the device driver data is waited for within a free defined period until the call returns (Blocking Call). If the waiting time is defined as „Zero“ the function is identical to the call of the function ReadFile(). If the waiting time is defined as „INFINITE“ there is no Timeout while waiting.

Calling DeviceIoControl() the -Network Interface has to be specified by its Handle „hFile“. The parameter „dwIoControlCode“ has to be occupied by the command „IOCTL_XLON_READWAIT“. The waiting time for the operation „ReadWait“ is defined in the parameter „lpInBuffer“ this has to be a pointer to a DWORD. The value of this DWORD specifies waiting time in milliseconds . If the value is „INFINITE“ the process of waiting lasts until data has arrived from the -Network Interface or until the driver has been closed. The parameter „nInBufferSize“ defines the length of the preceding DWORD including waiting period. The pointer „lpOutBuffer“ has to indicate a data structure of the type APILNI_Message which is also noted as Application Layer Buffer. The size of this data structure is variable during a reading operation it should however be put at maximum as the actual amount of data which has to be read is not known by the time of calling the function. The structure is shown below. For each specific call the valid size of this variable data structure is defined by the parameter „nOutBufferSize“. The actual amount of Bytes read from the Network Interface is passed back by the parameter „lpBytesReturned“. In case of success this value lays below or equals „nOutBufferSize“ and is not zero. The parameter „lpOverlapped“ is not used under Windows CE 3.0. Find detailed information about the different functions and about the function ReadFile() in general in the Windows CE 3.0 documentation.

Application Layer Buffer:

The below C-Code defines the data type „APILNI_Message“ for Application Layer Buffer as specified in the Echelon NSI Firmware User’s Guide. Find details on the structure of the Structure Element „ExpAppBuffer[]“ in the „LonWorks Host Application Programmer’s Guide“.

```
#define MAXLONMSG 253

typedef struct APILNI_Message_Struct {
    BYTE NiCmd; // NSI command
    BYTE Length; // Size of ExpAppBuffer
    BYTE ExpAppBuffer[MAXLONMSG]; // Message data
} APILNI_Message;
```

The graphics below shows again the structure of the Application Layer Buffer

command	2 Bytes	Application Layer Header
length		
message header	3 Bytes	ExpAppBuffer[MAXLONMSG], Maximum size. 253 Bytes
network address	11 Bytes	
message data	variable length	

Example:

```
#define IOCTL_XLON_READWAIT (DWORD)0x00 // IOCTL-Code for Command
                                         „ReadWait“

BOOL bResult;
DWORD dwTimeout = INFINITE;
DWORD dwBytesReturned;
APILNI_Message Ini_msg           // Application Layer Buffer for message from NSI

bResult = DeviceIoControl( myHandle,
                           IOCTL_XLON_READWAIT,
                           &dwTimeout,
                           sizeof( dwTimeout ),
                           (LPVOID)&Ini_msg,
                           255,
                           &dwBytesReturned
                           NULL );
```

Return value:

If the reading operation on the device driver has been successful the value „TRUE“ is passed back. If the amount of read bytes passed back in „lpBytesReturned“ equals zero no data had been available even after the waiting period has run down. If the reading operation on the device driver has not been successful the value „FALSE“ is passed back. In this case detailed information about error causes can be obtained by calling the function `GetLastError()`.

Error cause:

Possible error causes for a failure of this function:

An invalid Handle on the device driver, or a parameter „nOutBufferSize“ which is located outside the valid zone or does not conform with the actual length of the Application Layer Buffer.

6.2.6 **GetLastError()**

Prototype:

```
DWORD GetLastError( void );
```

Description:

This function gives detailed information about error cause if there is a failure in the functions `CreateFile()`, `CloseHandle()`, `ReadFile()`, `WriteFile()` or `DeviceIoControl()`. Find detailed information about the function `GetLastError()` in the Windows CE 3.0 documentation.

Example:

```
DWORD dwLastError;

dwLastError = GetLastError();
```

Return value:

Error code of the last operation. The below error codes are defined for the XLON device driver under Windows CE 3.0:

```
typedef enum
{
    LONDEV_SUCCESS = 0,           // No error detected
    LONDEV_NOT_FOUND,           // Device not found
    LONDEV_ALREADY_OPEN,       // Device already open
    LONDEV_NOT_OPEN,           // Not a handle to a open device
    LONDEV_DEVICE_ERR,         // Device detect error
    LONDEV_INVALID_DEVICE_ID,  // Invalid device id was detected
    LONDEV_NO_MSG_AVAIL,       // No message available
    LONDEV_NO_BUFF_AVAIL,      // Buffer is full
    LONDEV_NO_RESOURCES,       // No more resources available
    LONDEV_INVALID_BUF_LEN,    // Invalid buffer length
    LONDEV_DEVICE_BUSY         // Device is busy
} LonDevCode;
```

6.3 Application Interface under Linux

For the creation of a LON Host application under Linux you can basically refer to the documentation under 5.2.1. For access to the device driver from an own LON Host application under Linux commands of the operating system correspond to the commands used for other file operations under Linux. Find detailed information in the specific Linux documentation.


open()	Open driver of a specific device
close()	Close driver
read()	Read an „Uplink“ packet from the driver
write()	Write a „Downlink“ packet to the device

The structure of the data packet used for data exchange between LON Host application and device driver is explained in the „LonWorks Host Application Programmer's Guide“ as already mentioned in chapter 6.1.3. If there are no data packets available the read()-function blocks. This doesn't happen if the device driver has been opened with the „O_NONBLOCK“ Flag. In this case an error is passed back with the value „EWOULDBLOCK“. The write()-function works in the same way.

7 Appendix

7.1 Declaration of Conformity

The Product

Product	
Model Number	PCI1-WM-FTT, PCI1-PH-FTT, PCI1-WM-485, DH291-200-WM-FTT10A, DH291-200-WM-RS485
	is developed, constructed and produced in accordance with EC-guidelines 89/336/EG, altered 92/31/EG in exclusive responsibility of
Manufacturer	DH electronics GmbH Am Anger 8 83346 Bergen Germany

Standards to which Conformity is declared:

EMI Emission

Basic engineering standard: EN 50081-1

EMI Immunity

Basic engineering standard: EN 50082-1

In addition to, the basic engineering standard EN 50082-2 for industrial applications was checked.

National standards, guidelines and specifications used in addition:

None.

A complete technical documentation is available. The user's manual for the product is present in original version.

Bergen, 17th May 2001
Location, Date



Dipl.-Ing. (FH) Stefan Daxenberger
Managing Director

This EC-conformity declaration is valid for standard sets and therefore valid as copy.

8 Revision History

Version	Date	Description	State	By	Comments
1.0	06/03/03	Initial version	released	StS	