Comprehensive User's Guide
XLON® USB 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

Comprehensive User's Guide
XLON® USB Adapter

Version 1.0

Table of Contents

1 About this manual

This guide describes the hardware installation, software driver installation and the setup and configuration of the XLON® USB 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 XLON® USB LonTalk® adapter can be used to connect your PC or Notebook to a LonWorks ® network via the Universal Serial Bus. It is designed for use in industrial control, process control and building automation.

The XLON® USB 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 XLON® USB it is also possible, to transform a PC or Notebook into an extremely efficient LonWorks® node. In this case, the LonWorks® application runs on the PC and the XLON® USB 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 XLON® USB has an integrated FTT-10A transceiver for Free Topology and Link Power networks or a RS485 transceiver for Twisted Pair networks.

The XLON® USB owns a Service LED and a State LED for visualization of the LonTalk®

adapter's state. For use with manual installations an external Service Pin button is provided.

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

http://www.xlon.de

Scope of delivery

- XLON USB device
- USB cable (1 meter)
- Weidmueller-clamp für LonWorks® network connection (not in RS485 version)
- Floppydisk/CD with device drivers
- User's manual for installation.

Available variants

- USB4-WM-FTT with integrated FTT-10A Transceiver
- USB4-RJ-485 with integrated RS485 Transceiver

Find more informations about LonWorks® networks under:

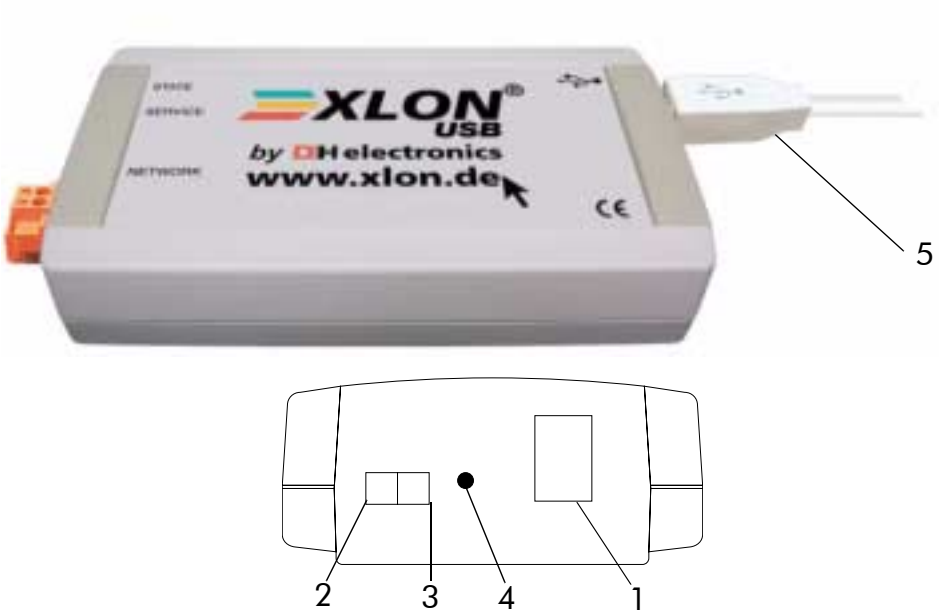
www.echelon.com

3 Installation of the XLON® USB adapter

3.1 Hints

You must not shut down the operating system for the hardware installation of the XLON® USB adapter.

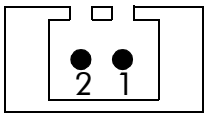
3.2 Hardware installation



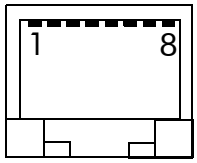
No.	Term	Remark
1	LON network connector	2-pin: FTT-10A transceiver RJ45: RS485 transceiver
2	LED green „STATE“	State of the XLON® USB adapter
3	LED yellow „SVC“	Display of Service Pin Neuron Processor
4	Service Pin Button	Manual release of a Service Pin message
5	USB connector	Please use delivered USB cable!

Pin allocation of the LON network connector

FTT-10A



RS485



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

- Connect the XLON® USB adapter to a free USB port, using the supplied USB-cable
- Plug in the LonWorks® network cable
- Under Windows based operating systems a „Found New Hardware Wizard“ starts, (see chapter 3.3.1)
- For installation of device drivers under Windows CE refer to chapter 3.3.2

De-installation

Disconnect the XLON® USB adapter from the USB port. De-installation of driver software is not necessary.

3.3 Software Installation

3.3.1 Initial installation under Windows

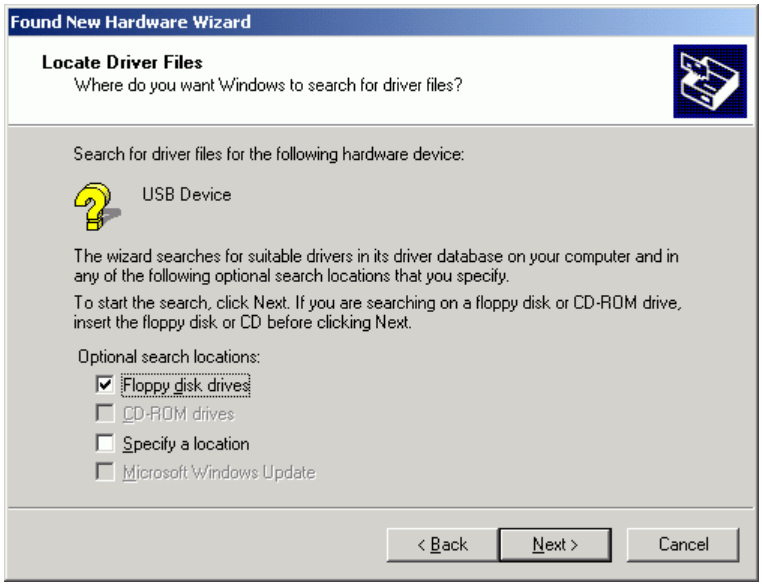
Initial installation under Windows 2000 is shown exemplary below. Installation under other Windows operating systems works analogously!

After conneting the XLON USB adapter to the computer you get the following message:

The wizard for device driver installation is started automatically:

- Click on „Next>“

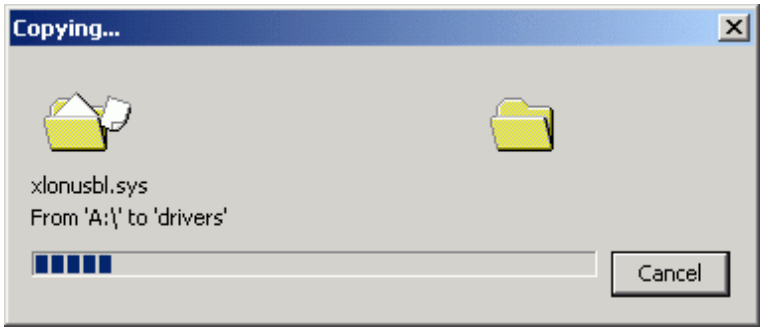
- Follow the procedure shown above and click on „Next>“



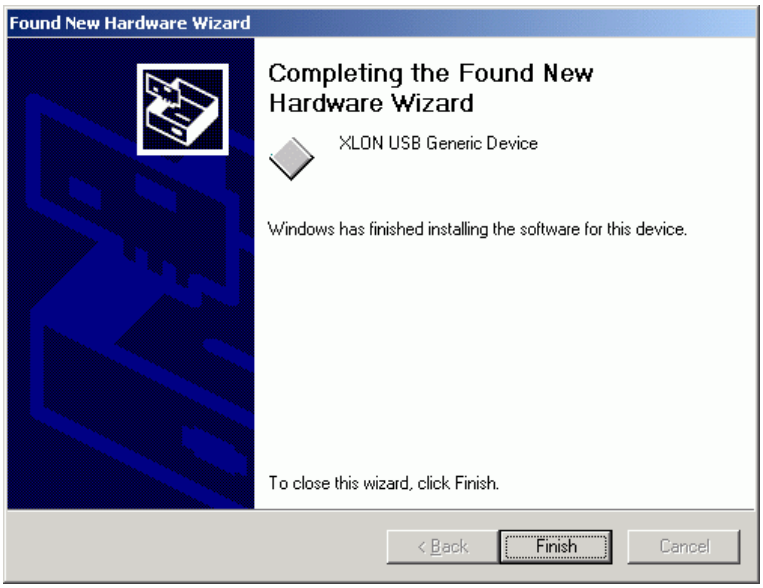
- Depending on the storage medium enclosed to your adapter either choose „Floppy disk drives“ or „CD-ROM drives“
- Insert Floppy disk or CD-ROM in the corresponding drive.
- Click on „Next>“

A driver has been found:

- Click on „Next>“



After ending the installation process:



- Click on „Finish“

After installation of the XLON® USB generic device driver, a second driver has to be installed. You get the following message:

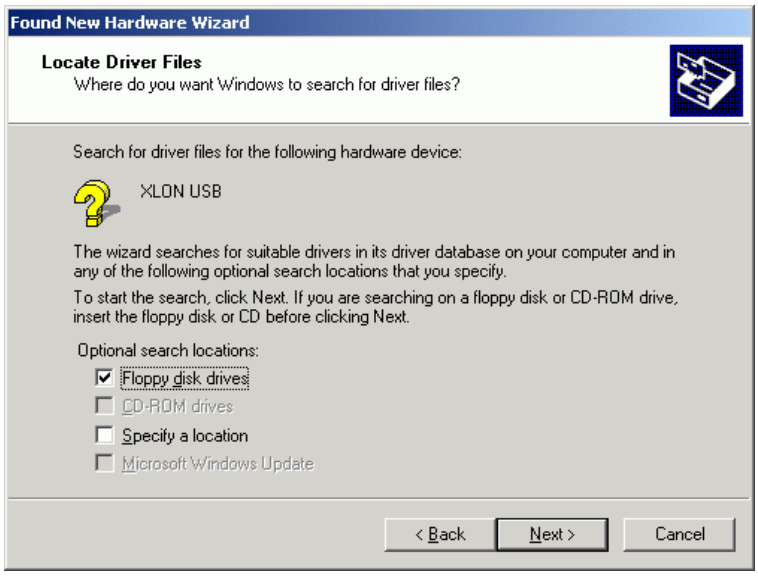
The wizard for device driver installation is started automatically:

- Click on „Next>“

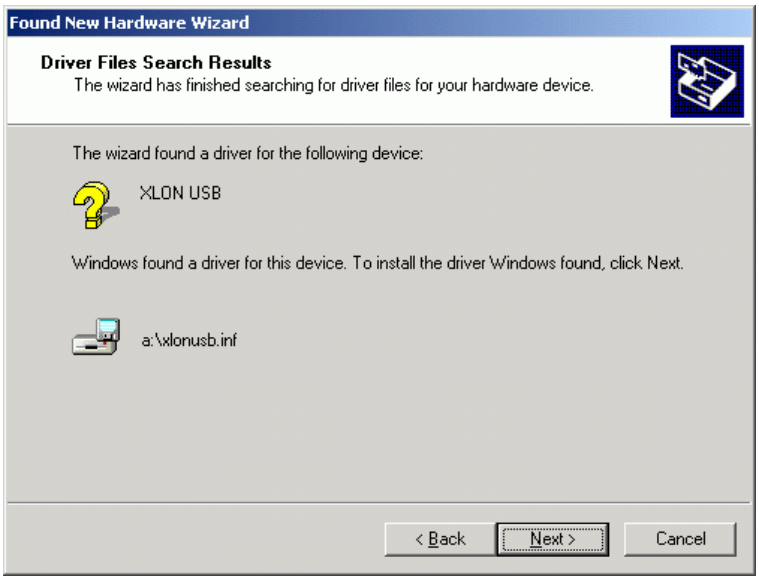
- Follow the procedure shown above and click on „Next>“

- 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>“



- 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:

- ARM
- MIPS
- SH3
- SH4
- 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 127 XLON USB adapters within one system.

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 files „xlon_usbl.dll“ and „xlon_usb.dll“ have 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 XLON® USB 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 files „xlon_usbl.dll“ and „xlon_usb.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“
- 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_usb.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 „XLON® USB“ 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 XLON® USB 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_usb.reg“. The content is explained below.

[HKEY_LOCAL_MACHINE\Drivers\USB\LoadClients]
The registration code HKEY_LOCAL_MACHINE\Drivers\USB\LoadClients contains subcodes defining which device driver the USB subsystem loads for which XLON® USB adapter. If you plug in a device and an associated registration entry is found, then the associated driver is loaded, otherwise the user is requested for the input of a device driver.

The subkeys of „LoadClients“ have the form „Group1_ID\Group2_ID\Group3_ID\ Driver Name“. The character string with the name of the device driver can be found under the subkey „DriverName“. For the XLON® USB adapter only the subkey „Group1_ID“ is important, because there is the Vendor- and Product-ID specified. Both ID’s are seperated with an underscore. The „Group2_ID“ specifies the Device class, the „Group3_ID“ the Interface class which is supported by the device driver. The XLON® USB adapter is clearly characterized with „Group1_ID“, so „Group2_ID“ and „Group3_ID“ is set to „Default“. This means the driver will be loaded if a XLON® USB adapter with the right Vendor- and Product-ID is connected. The subkey „DriverName“ carries the names of the respective device driver and contains the entries described in the following table.

Value Name	Term	Remark
Dll	**REG_SZ**	This required entry specifies the file name for a driver DLL which is loaded by the Device Manager, e.g. xlon_usbl.dll or xlon_usb.dll

3.3.2.4 Hardware Configuration

With the Windows CE 3.0 device driver you can use up to 127 XLON® USB adapters in one system. Because of the Hot Plug and Play functionality of the USB system, no additional hardware configuration is neccessary.

3.3.3 Initial Installation under Linux

Currently there is no device driver for Linux for the XLON® USB adapter available.

3.3.4 Driver update

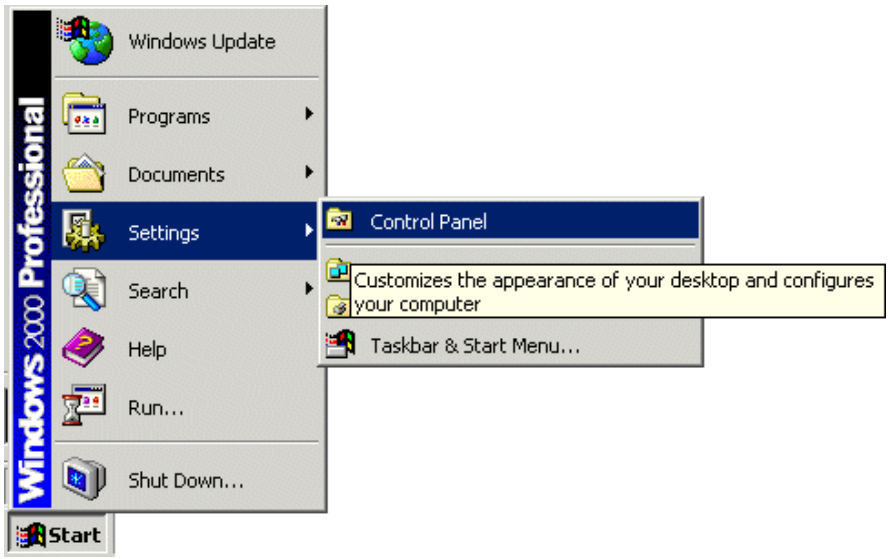
☞ Find the current drivers for download under:

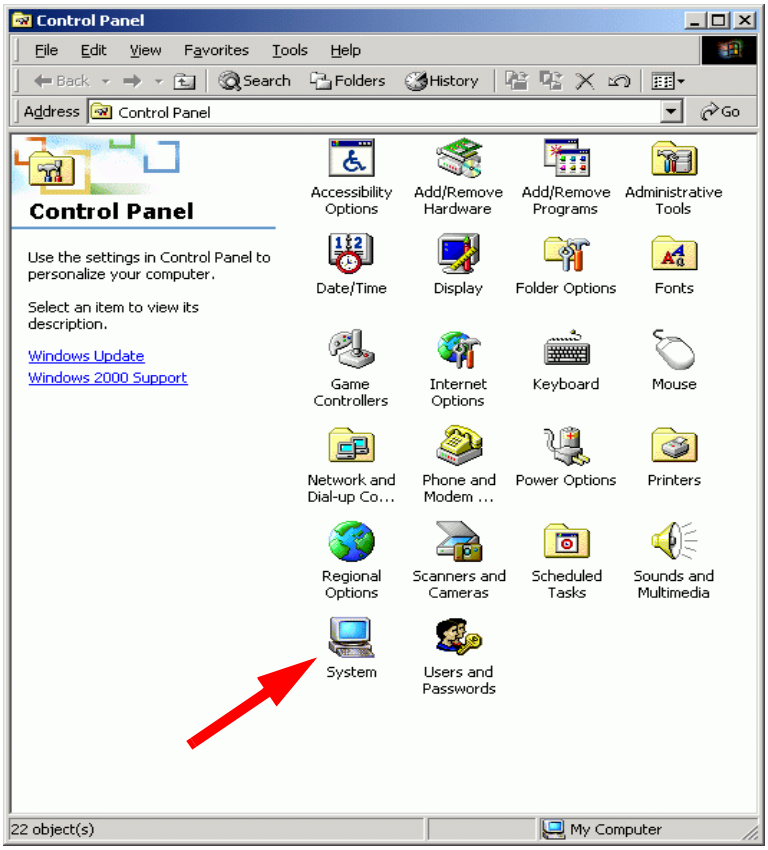
www.xlon.de

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

3.3.4.1 Windows

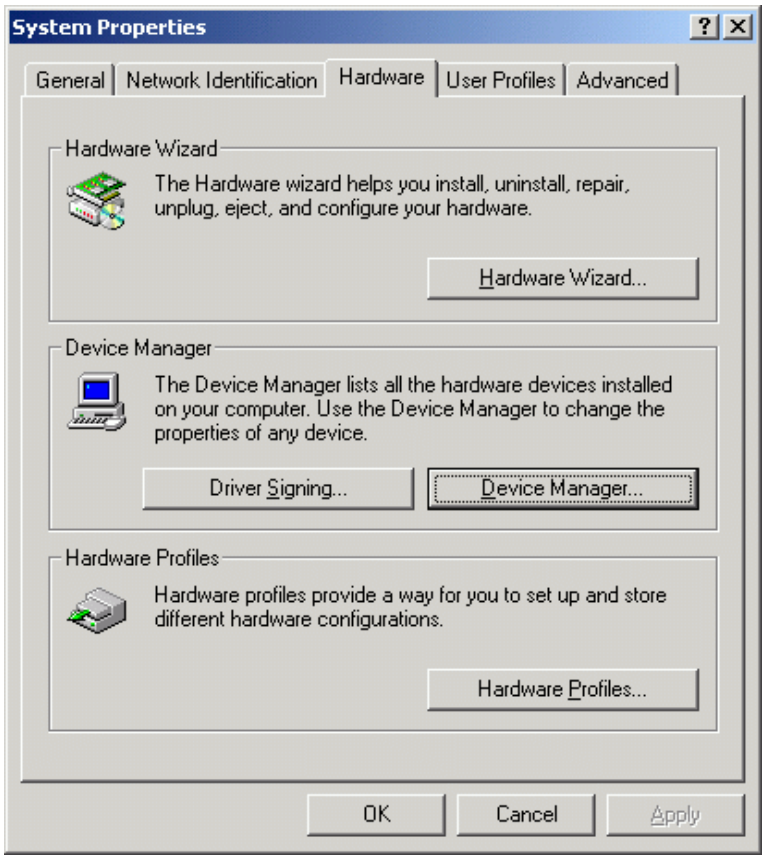
- Open the Control Panel
  - Click on „Start“
  - Go to „Settings“ with the mousepointer
  - Click on „Control Panel“



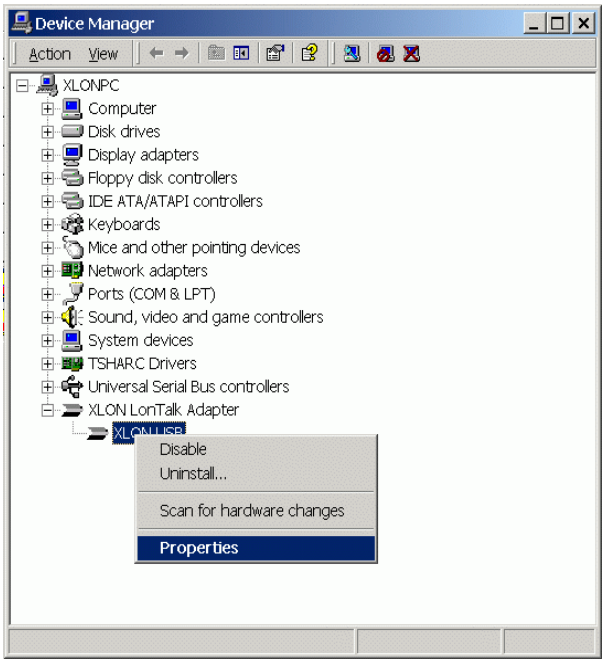


- Select „System“ by doubleclick

- Go to tab „Hardware“

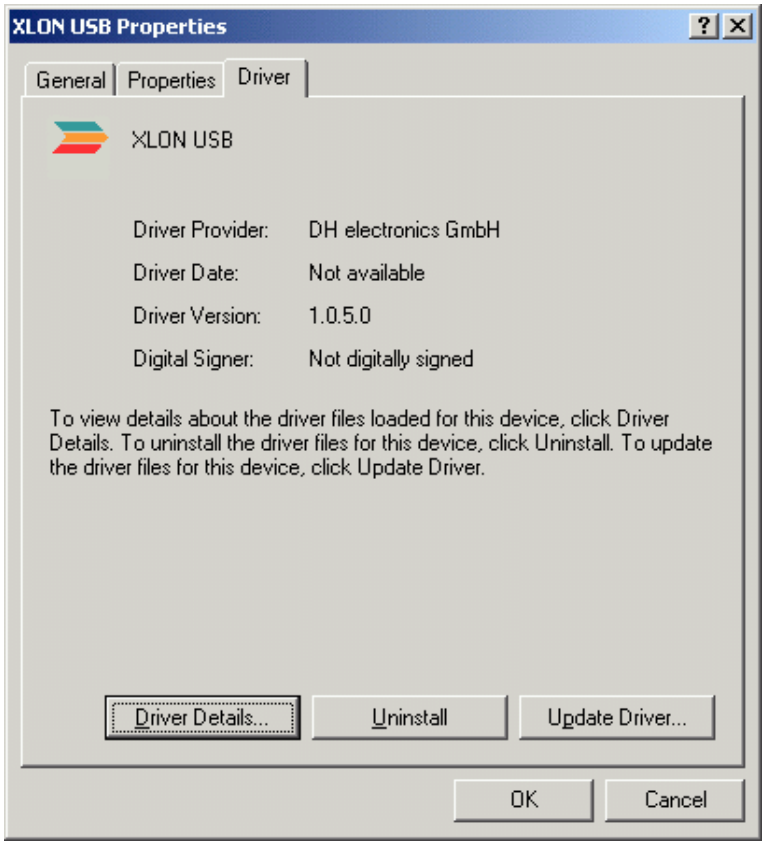


- Open the „Device Manager...“ by clicking the mouse



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

- Click on tab „Driver“

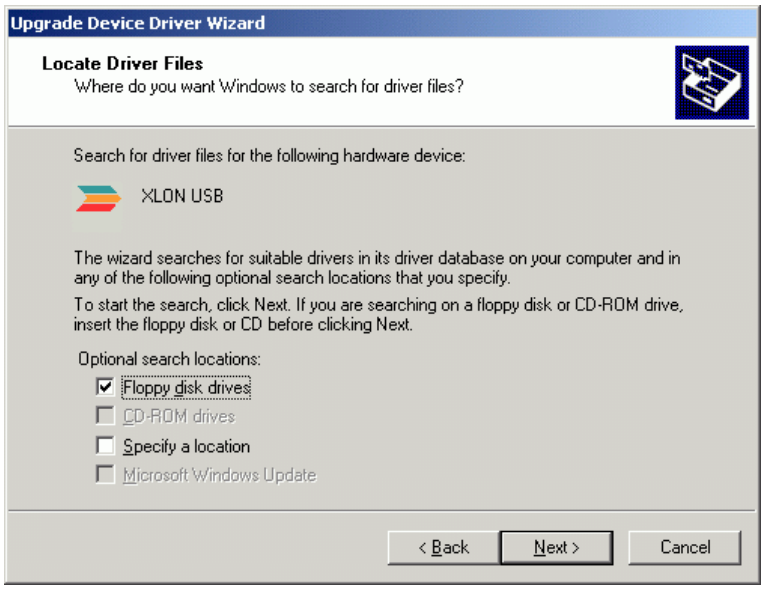


- Check if the driver version installed on your system is lower than the version you are intending to install.
- Click on „Update Driver...“

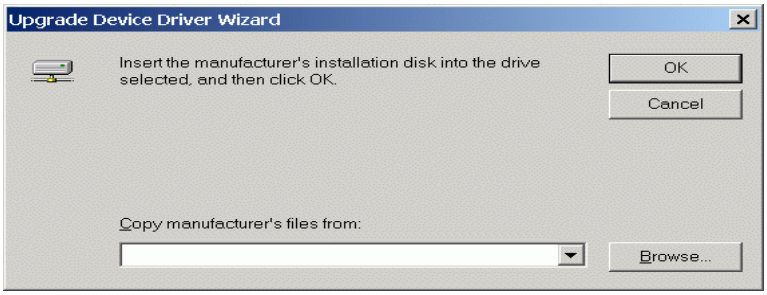
- The wizard for driver upgrade opens
- Follow the instructions displayed on the screen

- Click on „Next>“

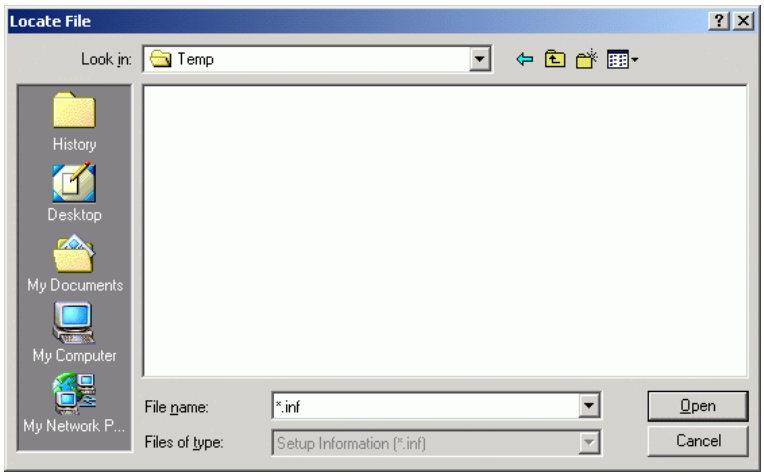
- Click on „Next>“



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



- Click on „Browse“



- 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

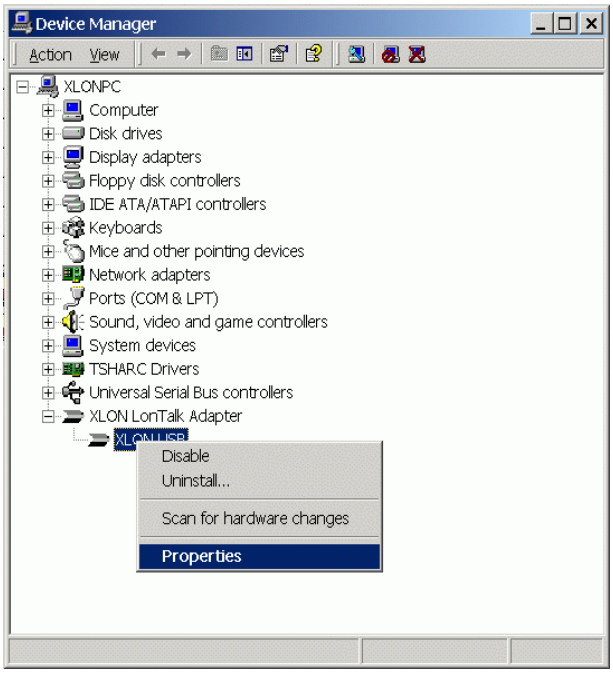
Currently there is no device driver for Linux for the XLON® USB adapter available.

3.4 Software De-installation

De-installation of software is not necessary. Simply disconnect the XLON® USB adapter from the USB port. Find information in chapter 3.2.

4 Configuring and Testing

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

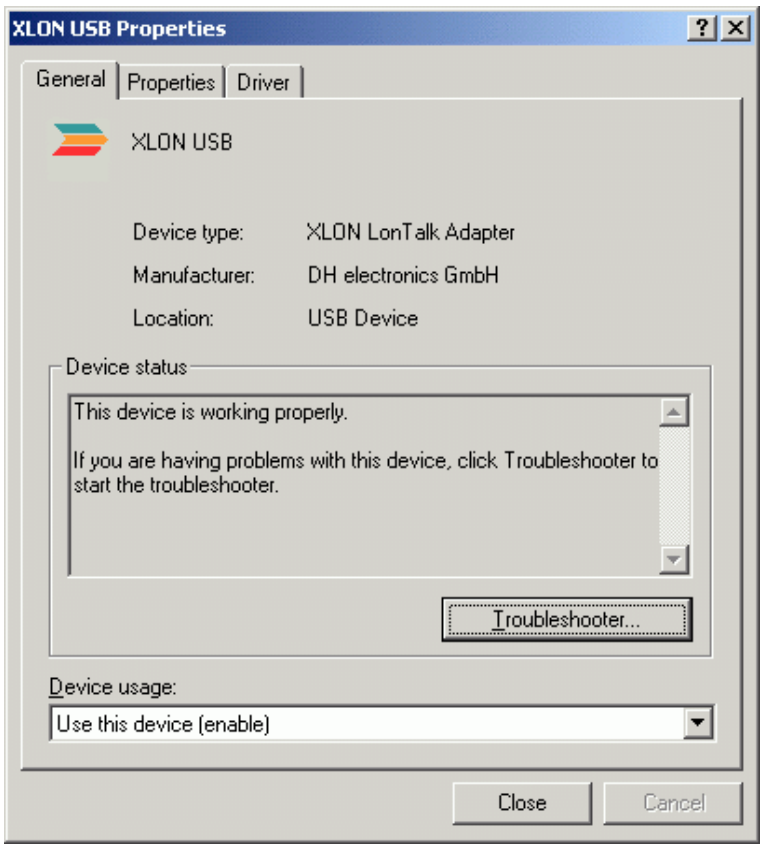


- The „Properties“ window opens

4.1 Check of settings under Windows

4.1.1 General settings

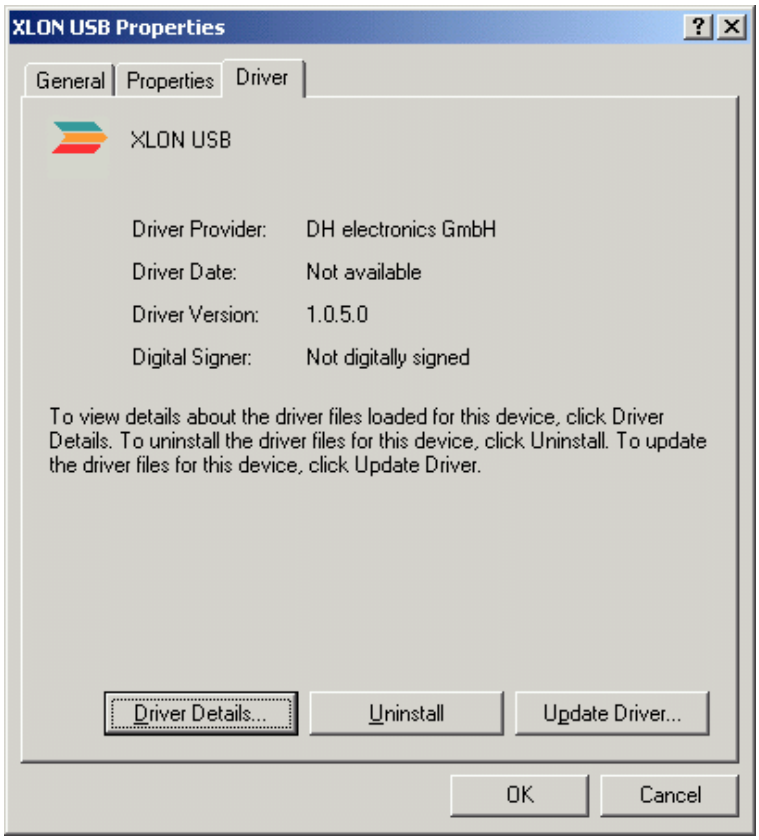
Tab General:



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

4.1.2 Driver information

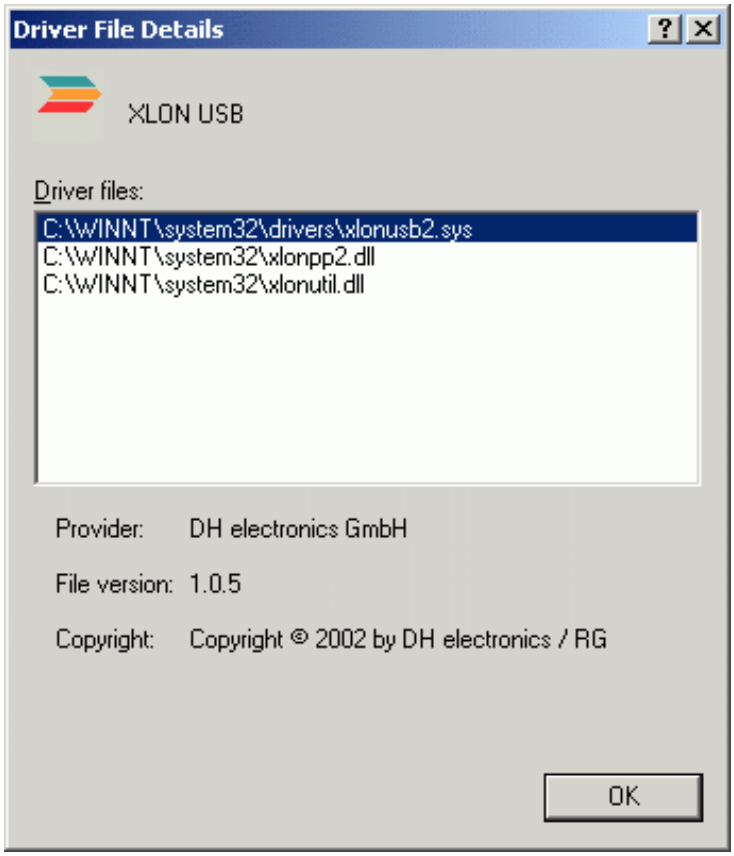
- Click on tab „Driver“



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...“



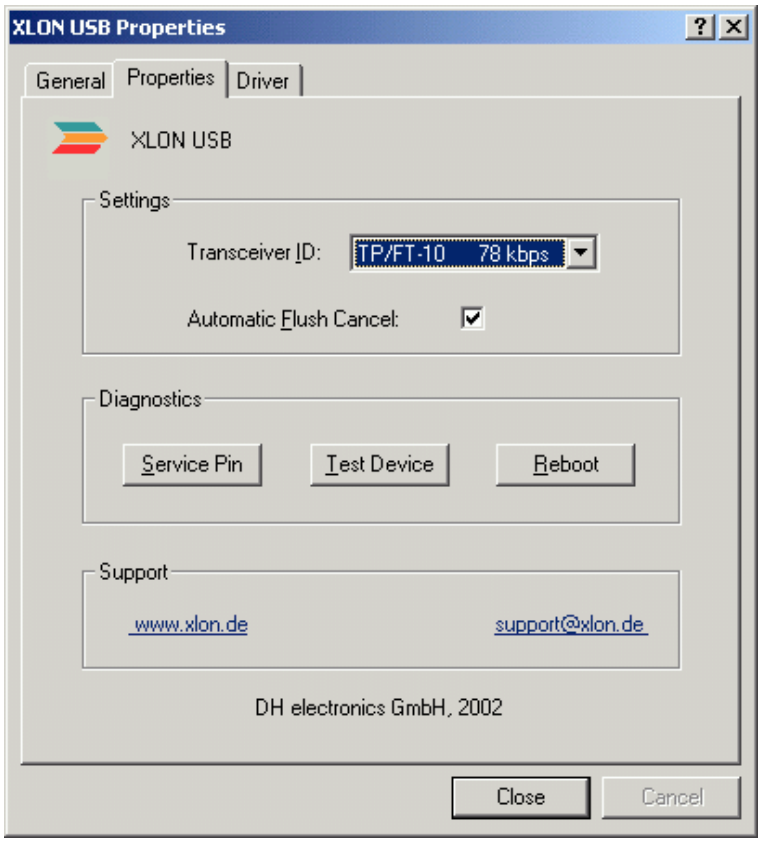
Selecting one of the indicated files you get information about the Provider, File version and Copyright.

☞ **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 Properties of the XLON® USB adapter

- Click on tab „Properties“



☞ The current configuration of the XLON® USB 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® USB 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.

Actvating/De-activating Automatic Flush Cancel

After each reset of the XLON® USB 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® USB adapter.

If the Automatic Flush Cancel function is activated the device driver sends a Flush Cancel command to the XLON® USB 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® USB adapter!**

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

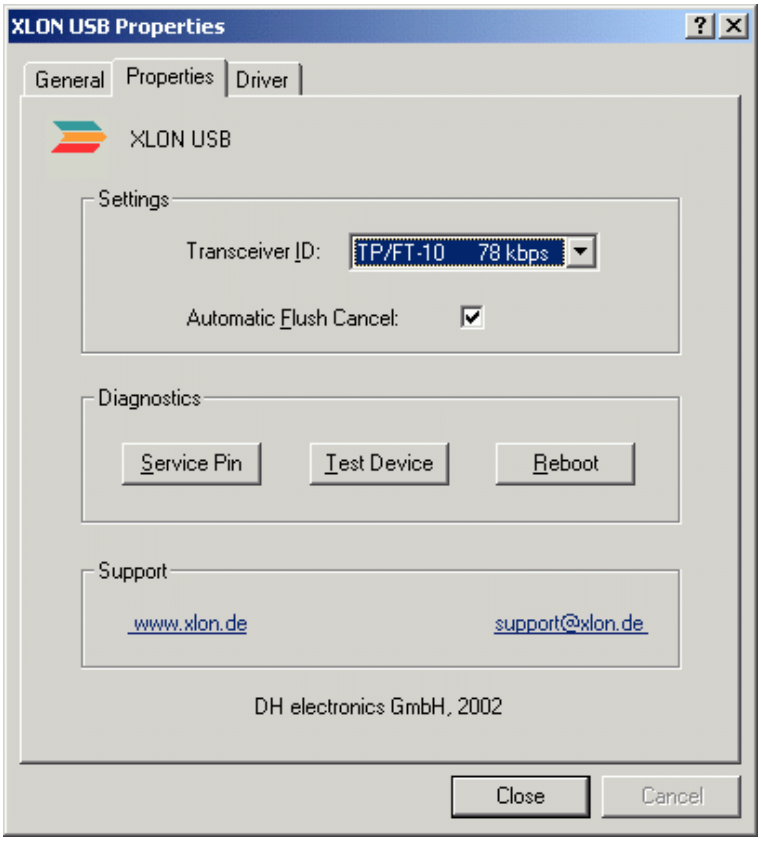
The function of the buttons „Service Pin“, „Test Device“ and „Reboot“ is explained in chapter 4.2.

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

4.2 Test of the XLON® USB adapter under Windows

4.2.1 Diagnosis by Software

- Open the „Properties“ window of the XLON® USB 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® USB 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® USB 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® USB adapter will be indicated.

Reboot

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

4.2.2 Diagnosis by LED

As described in chapter 3.2 the XLON® USB adapter is fitted with two LED's for visualisation. A green State LED and a yellow Service-Pin LED.

Green State LED:

The green LED indicates if the device is ready for use or not. If the LED is on the device is ready to use and the device driver is installed correct.

☞ If the LED is off, the device doesn't work correct, or no device driver is installed or loaded. Another possibility is, that the XLON® USB adapter is in the Suspend mode.

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 XLON® USB adapter is made from an application the LED is de-activated.
- If the LED blinks with a frequency oft ½ Hz the state of the XLON® USB 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 XLON® USB 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	Describtion
Constantly off	- Successful installation of the XLON® USB adapter if green LED is constantly on or - Faulty installation of the XLON® USB adapter if green LED is constantly off
Constantly on	- Faulty hardware of the XLON® USB adapter
Flash frequency1,25 Hz	- No application has accessed the XLON® USB adapter
Flash frequency ½ Hz	- State of the XLON® USB adapter is „Unconfigured", this means the XLON® USB adapter has no network address
Flashes for a moment	- A „Reset" to the XLON® USB adapter has been made or - A device test has been made

5 Technical Specification

5.1 Hardware

5.1.1 General Information

Bus Interface	USB conform, in accordance with USB specification Revision 1.1, 12 MBit/s
Network Connection FTT-10A: RS485:	2-conductor Weidmueller connector with tension clamp connection and screw flanges 8-pole Western Modular connector
Power Supply	Via the USB
Service Pin Function	Controlled by host or Service button
Configuration State	Displayed on host as well as via Service and State 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, ±5%, 100 mA typical
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	123 x 66 x 30 mm (4.84“ x 2.68“ x 1.18“) (lenght x width x height)
Weight	100 g

The hardware of the XLON® USB adapter supports up to 127 adapters per Universal Serial Bus in the PC (Multiple Device Support).

5.1.2 Plug Connection

As plug connection for the LonWorks®-connection of the XLON USB adapter a Weidmueller plug connection (Series BL3.5) is used in the FTT-10A version. In the RS485 version a Western Modular (RJ45) connector is used.

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

The cross-section of the line which can be clamped at maximum is 1.5 mm^2.

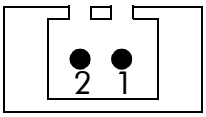
Find order information as well as further detailled information about this connector under:

www.weidmueller.de

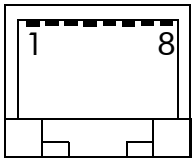
Plug Allocation:

Pin allocation LonWorks® connector

FTT-10A

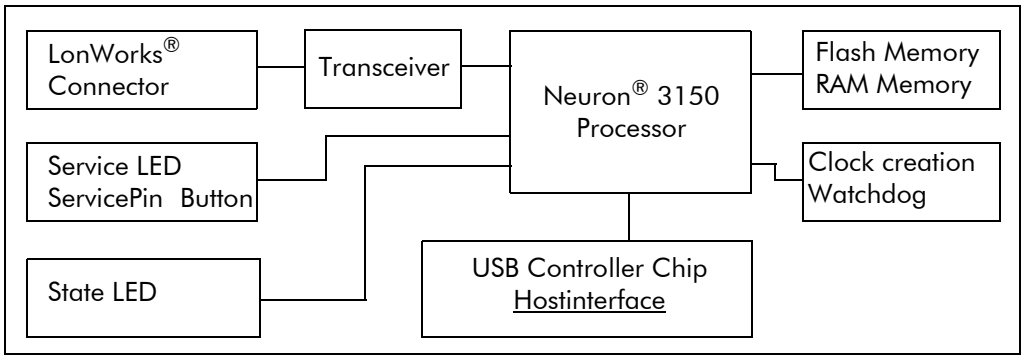


RS485



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

5.1.3 Block model



5.1.4 Technical Details of the Hardware

Connection to the Host System via USB:

Connection to the Universal Serial Bus follows USB-Specification Revision 1.1. The transmission rate is 12 MBit/s.

The XLON® USB adapter is completely compatible to Plug&Play. Within the system it is identified by the DH electronics GmbH Vendor ID ‚0x0916' which is given by the ‚USB Implementers Forum' and the Device ID's ‚0x0001' and ‚0x0002'.

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 (Western Modular connector). The transmission rate of the FTT-10A Transceiver is 78,5 kBit/s. If the XLON® USB 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. The isolation voltage for the RS485 version is 1.5 kV (UL rated).

Neuron Processor Core:

A 3150® Neuron Processor with external storage interface is used. A reprogramable Flash memory is used as program memory. A SRAM memory is used as data memory. The Neuron Processor is connected to the USB-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 XLON® USB 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® USB adapter via a LNS application as network interface you only have to indicate the XLON® USB adapter you are intending to use. The device driver supports up to 127 XLON® USB adapters per PC-System, this means that several XLON® USB 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.

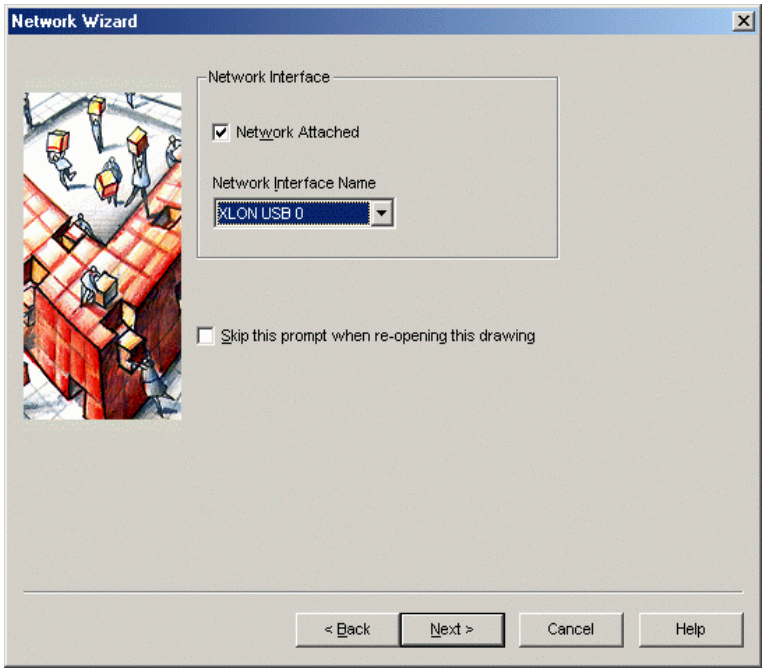
The LNS-application showes a „Select" menu for the Network Interface Name. Find the XLON® USB adapter under the Network Interface Name:

XLON USB x

x: XLON® USB Network Interface Number

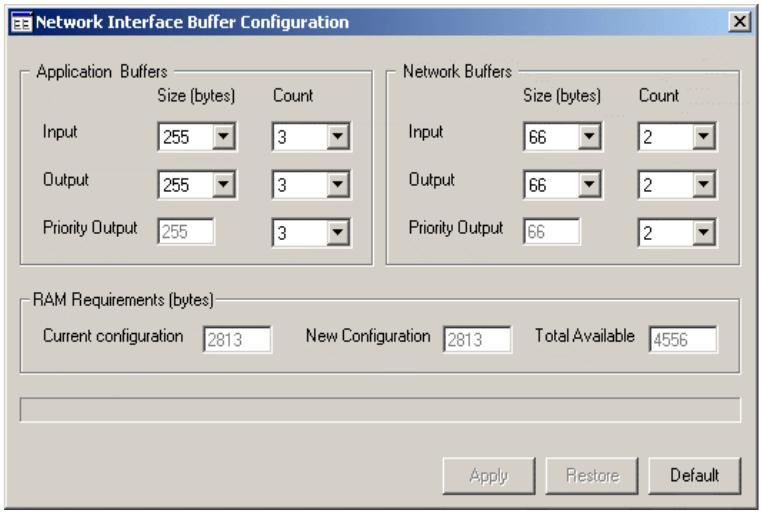
For example: XLON USB 0 => first XLON® USB adapter installed on the system.

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



6.1.2 Configuration of the Network Interface Buffer

The Network- and Applicaton 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 XLON® USB 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 XLON® USB adapter is: ‚\\.\xlonusb0'

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® USB adapter under the Alias Name: XLON USB x

x: XLON® USB Network-Interface-Number

For example: XLON USB 0 => first XLON® USB 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 XLON® USB adapter and passes back a handle for access to the device driver. ‚\\.\xlonusb0' has to be used as name for the device driver.

2.) *pHandle = CreateFile("\\\\.\\xlonusb0", 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 XLON® USB adapter and passes back a handle for access to the device driver. ‚\\.\xlonusb0' has to be used as name for the device driver.

2.) *pHandle = CreateFile("\\\\.\\xlonusb0", 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 XLON® USB adapter a Program ID has to be programed 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 XLON® USB 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 4.1.3) 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 LastError(). 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 XLON®-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 occured. Processing of data is then running in parallel to the

LON application in the background.

Each XLON®-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 XLON®-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 XLON®-Network Interface is not being waited for.

The XLON®-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 „nNumber ofBytesToRead“ 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 lni_msg                // Application Layer Buffer for message from NSI

bResult = ReadFile(          myHandle,
                             (LPCVOID)&lni_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 DeviceIo Control()

Prototype:

```
BOOL DeviceIoControl(   HANDLE hFile,
                        DWORD dwIoControlCode
                        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 DeviceIoControl() 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 XLON®-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 XLON®-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 XLON®-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 XLON®-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 lni_msg              // Application Layer Buffer for message from NSI

bResult = DeviceIoControl(     myHandle,
                               IOCTL_XLON_READWAIT,
                               &dwTimeout,
                               sizeof( dwTimeout ),
                               (LPVOID)&lni_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

Currently there is no device driver for Linux for the XLON® USB adapter available.

7 Appendix

7.1 Declaration of Conformity

The Product

Product	XLON® USB
Model Number	USB4-WM-FTT, USB4-RJ-485 und 229838 Interface LON-PC USB
	is developed, constructed and produced in accordance with EC-guidelines 89/336/EG, altered 92/31/EG in exclusive responsbility of
Manufacturer	DH electronics GmbH Am Anger 8 83346 Bergen Germany

Standards to which Confirmity 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, guidlines 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	Describtion	State	By	Comments
1.0	07/03/03	Initial version	released	StS	