XLON XLDV32.DLL Programmer's Guide

1 Table of Contents

XLON XLDV32.DLL Programmer's Guide

2 Introduction

The device driver interface of the LonTalk adapter (LTA) depends on the operating system being used. Furthermore only one application could access the device driver. When the programmer needs to access more than one physical LTA he has to handle this by software.

To get around with all these limitations and disadvantages the ***XLON***® „xldv32.dll" software library has been developed. So the programmer doesn't need to handle these low level tasks within his application anymore.

3 Features

Based on the background of chapter 2 the ***XLON***® „xldv32.dll" has the following features:

- Easy application programming interface (API) for accessing the LonTalk adapter
- Highest flexibility when programming LON applications
- All features of the library can be used on all different operating systems (platform independency)
- A common API for accessing the LonTalk adapter on different operating systems (multi-OS support)
- Easy access of one or more applications to one physical LonTalk adapter (multi-client support)
- Easy access of one or more applications to one or more physical LonTalk adapters (multi-interface support)
- Low-cost alternative for Gesytec's „wldv32.dll"

The ***XLON***® „xldv32.dll" software library has been developed by using Microsoft Visual Studio 6.0 C++. The functions calls of the API are according to ANSI-C.

4 Supported Operating Systems

The following operating systems are supported by the actual version of the XLON® „xldv32.dll" software library:

- Windows 95
- Windows 98
- Windows 98 second edition
- Windows ME
- Windows NT 4.0
- Windows 2000 Home Edition
- Windows 2000 Professional Edition
- Windows XP Home Edition
- Windows XP Professional Edition
- Windows 2000 Server
- Windows Server 2003

The following operating systems are planned to be supported by the XLON® „xldv32.dll" software library:

- Windows CE 3.0
- Windows CE .net
- Linux

5 Supported XLON® LonTalk adapters

The following ***XLON***® LonTalk adapters are supported by the ***XLON***® „xldv32.dll" software library (if there is a 32 bit driver available for the OS):

- XLON PCI
- XLON USB
- XLON DONGLE
- XLON PC/104
- XLON PC
- XLON RNI

Obsolete 16 bit MS-DOS drivers for Windows 9x operating systems are not anymore supported.

Using the ***XLON***® „xldv32.dll" software library with ***XLON***® LonTalk adapters is for free. When using it with 3rd party LTAs you have to pay a license fee, please contact DH electronics. Further informations can be found on our website www.xlon.de or contact us by eMail under info@xlon.de

The following table gives an overview of possible device driver names of our ***XLON***® LTAs (please note that the index at the end of the device driver names could differ depending on the number of used LTAs):

XLON®	Windows 95/98/ME	Windows NT/2000/XP	Windows CE
PCI	\\.\xlonpci0	\\.\xlonpci0	LON1:
USB	\\.\xlonusb0	\\.\xlonusb0	LON1:
Dongle	\\.\london	\\.\dhlon10	LON1:
PC/104	\\.\xlonpc104	\\.\dhlon10	LON1:
PC	\\.\xlonpc	\\.\dhlon10	LON1:
RNI	\\.\xlonrni0	\\.\xlonrni0	LON1:

*Overview of possible device driver names of **XLON**® LTAs under different operating systems.*

XLON XLDV32.DLL Programmer's Guide

Please read the corresponding Comprehensive User's Guide of the XLON® LTA, to get detailed informations about the device driver name. You can download this document from our website www.xlon.de.

According to Echelon's specification, you can get the device driver name on Microsoft Windows 32 operating systems under the following registry path:

- „HKEY_LOCAL_MACHINE\SOFTWARE\LonWorks\DeviceDrivers"

6 Compatibility to WLDV32.DLL from Gesytec

The application programming interface (API) of the ***XLON***® „xldv32.dll" is compatible with the following products of Gesytec:

- Standard version of the WLDV32.DLL
- Multi-Client version of the WLDV32.DLL

When using the ***XLON***® „xldv32.dll" you can now also use the features of the Multi-Client version of the "wldv32.dll" on all Windows 9x and Windows CE operating systems.

By renaming the ***XLON***® „xldv32.dll" into „wldv32.dll" any version of the „wldv32.dll" could be easily replaced. In contrast to Gesytec's solution the ***XLON***® „xldv32.dll" does not use any Windows NT specific services and mechanisms like *OLE Automation Server*, *EXE Server* or *COM*.
Therefore all features of the ***XLON***® „xldv32.dll" can also be used on Windows 9x and Windows CE operating systems. This is a very important feature, particular when using the „Multi-Client" and „Multi-Interface" features. An additional advantage is that no components have to be registered. Therefore the ***XLON***® „xldv32.dll" can be easily copied, moved, deleted or replaced.

XLON XLDV32.DLL Programmer's Guide

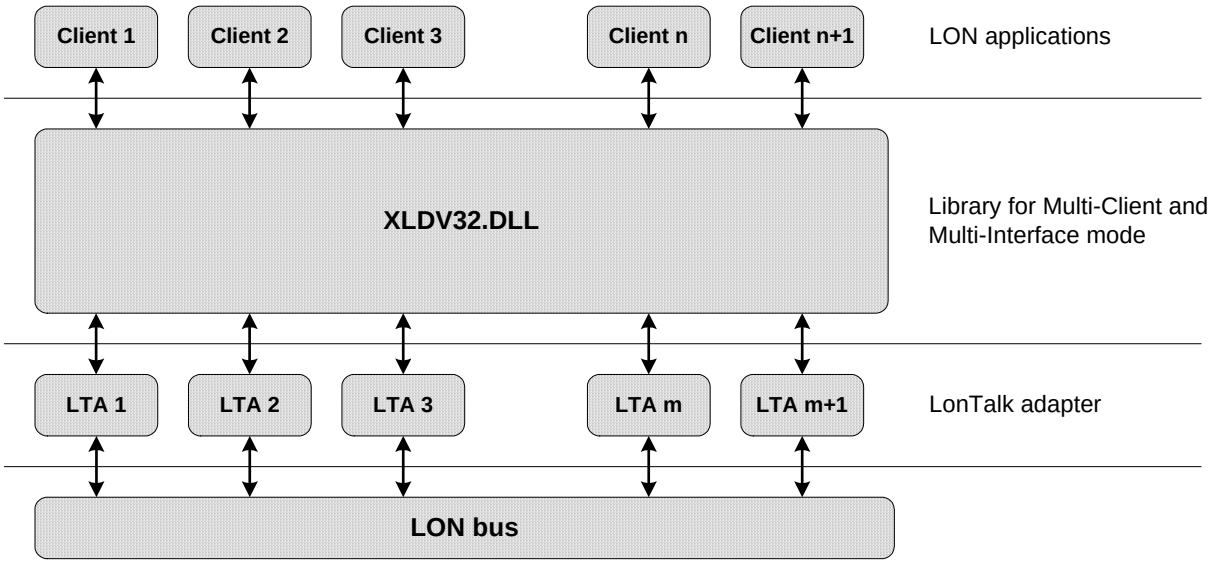
7 Multi-Client and Multi-Interface mode

When using the „xldv32.dll" in the Multi-Interface mode the application software (client) has the ability to access several ***XLON***® LTAs.

When using the „xldv32.dll" in the Multi-Client mode several applications (clients) have the ability to access one common ***XLON***® LTA. In Multi-Client mode all received LON messages are forwarded to all clients.

Of course a mixture of Multi-Client and Multi-Interface mode is also possible. So several applications (clients) have the ability to access several ***XLON***® LTAs.

For Multi-Interface and Multi-Client mode, it is required to register a callback-function as described in chapter 10.1.5.



8 Installation on Windows systems

No specific install program is required for installation of the ***XLON***® „xldv32.dll". The ***XLON***® „xldv32.dll" could easily be installed with the corresponding LON application or simply copied by hand onto the system.

If the ***XLON***® „xldv32.dll" should be shared by more than one windows application it makes sense to install or copy the file into one of the following paths:

- „Windows\System" for Windows 9x based operating systems (Windows 95, 98, 98SE, ME).
- „Windows\System32" for Windows NT based operating systems (Windows NT/2000/XP/XP embedded).

It is sufficient to copy the ***XLON***® „xldv32.dll" into the local application directory when being used by one application only.
If more applications need to access the ***XLON***® „xldv32.dll" it must be considered that every application opens the same instance of the ***XLON***® „xldv32.dll". This will be guaranteed by the above paths.
Like already mentioned no registry entries are registered during installation process or are necessary for a correct behaviour of the ***XLON***® „xldv32.dll".

This is practice-oriented as the most common problems with the Windows registry are mostly

- obsolete,
- only partly deleted or
- wrong Windows registry entries and very often lead to problems when distributing software.

9 How to implement XLDV32.DLL?

Programming an application on base of the XLON® „xldv32.dll" under Microsoft Visual C++ 6.0 is very easy.

Only the following steps are necessary:

- Every source code module which uses the XLON® „xldv32.dll" must include the header file „xldv32.h"
- The static library „xldv32.lib" must be known by the linker, thus it is loaded automatically during runtime. Under Microsoft Visual C++ 6.0 this is done in the following way: Click on *Project-Settings* – now the project dialog window opens – under the tab *linker* you can add the „xldv32.lib".
- Now the dynamic link library „xldv32.dll" will be loaded automatically when starting the application.

10 Application Programming Interface

10.1 Application Programming Interface on Windows systems

The ***XLON***® „xldv32.dll" provides the application programmer with an interface (API) for accessing one or more ***XLON***® LTAs.

The following functions are part of the API:

ldv_open()	ldv_register()
ldv_close()	ldv_get_version()
ldv_read()	ldv_debugmode()
ldv_write()	ldv_neuronID()

In the following chapters you will find a detailed description of the API functions.

10.1.1 ldv_open()

Prototype:

LNI **ldv_open**(LPCTSTR *lpDeviceName*);

Description:

This function opens the device driver of the ***XLON***® LTA which is specified by „lpDeviceName". In order to get the correct device driver name, please read the Comprehensive User's Guide of the ***XLON***® LTA. You can download this document from www.xlon.de. Here you will find further informations.
To get an overview, please have a look at the table in chapter 5 of this document. Due to compatibility reasons you can also use the prefix „LDV/" in front of the device driver name. This is not a must and is not recommended for new development projects.

Example:

```
LNI hLni, hLni1;          // handle that specifies a communication channel
                          // LNI means Lon Network Interface, a synonym for LTA
hLni = ldv_open( _TEXT( "LDV/\\.\xlonusb0" ) );      // device name with prefix „LDV/"
hLni1 = ldv_open( _TEXT( "\\.\xlonpci1" ) );         // device name without any prefix
```

Return value:

If the device driver has been opened successfully a handle is passed back and is used as an identifier for the just opened communication channel between the application and the ***XLON***® LTA. Use this handle for further functions of the API.
In case of an error a value according the LDV-codes specified in the „xldv32.h" header file is returned. More detailed information about the error cause can be passed to the application by calling the GetLastError() function which is described in chapter 10.1.9.

Error cause:

Possible error causes for a failure of this function are:

- ***XLON***® LTA is not available or not installed
- False name of the device driver
- Device driver has not been loaded or is already opened

10.1.2 ldv_close()

Prototype:

LDVCode **ldv_close(** LNI *hLni*);

Description:

This function closes the communication channel to an **XLON**® LTA which is specified by the handle „hLni". As function parameter the handle „hLni" passed back after successfully calling the function ldv_open() has to be used.

Example:

```
LDVCode ldv_code;                 // return code of API functions in xldv32.dll
ldv_code = ldv_close( hLni );     // hLni was initialized in the sample under ldv_open()
```

Return value:

If the communication channel has been closed successfully the value „LDV_OK" is passed back. In case of an error a value according the LDV-codes specified in the „xldv32.h" header file is returned. More detailed information about the error cause can be passed to the application by calling the GetLastError() function which is described in chapter 10.1.9.
After a successful call of the ldv_close() function the handle that has been passed is not valid any longer and can no longer be used for calling other functions of the API.

Error cause:

Possible error causes for a failure of this function are:

- Invalid LNI-handle
- Communication channel specified by this LNI-handle has already been closed

10.1.3 ldv_read()

Prototype:

LDVCode **ldv_read(** LNI *hLni*, LPCVOID *lpMsg*, UINT *uMsgLen*);

Description:

By this function data is read from the **XLON**® „xldv32.dll" and consequently from the **XLON**® LTA by an application. Calls made by this function are asynchronous. This means that the function returns as soon as data from the internal buffer of the **XLON**® „xldv32.dll" is passed back. If no data is available or if there has been an error, the function also returns immediately. This means that there is no waiting for data from the **XLON**® LTA.

The respective communication channel is specified by its handle „hLni". The pointer „lpMsg" must point to a data structure of the type „APILNI_Message" which is also noted as Application Layer Buffer. Please have a look at chapter 11 for more information about this buffer. The size of this data structure is variable. However during a read operation it should always be set to the maximum value of 255 bytes as the real size of incoming data is not known at that time in most cases. For each specific call the size of this variable data structure is set by the parameter „uMsgLen".

Example:

```
LDVCode ldv_code;
APILNI_Message lni_msg;              // application layer buffer for message to receive
                                     // hLni was initialized in the sample under ldv_open()
ldv_code = ldv_read( hLni, (LPCVOID)&lni_msg, sizeof(APILNI_Message) );
```

Return value:

If the reading operation on the **XLON**® „xldv32.dll" has been successful the value „LDV_OK" is passed back. In case of an error a value according the LDV-codes specified in the „xldv32.h" header file is returned. More detailed information about the error cause can be passed to the application by calling the GetLastError() function which is described in chapter 10.1.9.

Error cause:

Possible error causes for a failure of this function are:

- No data available
- Invalid LNI-handle
- The selected size in parameter „uMsgLen" is too small for the data to be read

10.1.4 ldv_write()

Prototype:

LDVCode **ldv_write** (LNI *hLni*, LPCVOID *lpMsg*, UINT *uMsgLen*);

Description:

By this function data from an application is written to the **XLON**® „xldv32.dll" and consequently on the **XLON**® LTA. Calls made by this function are asynchronous. This means that the function returns as soon as data has been taken info the internal buffer of the **XLON**® „xldv32.dll" or an error has occurred. Processing of data is then running in parallel to the application in the background.
The respective communication channel is specified by ist handle „hLni". The pointer „lpMsg" must point to a data structure of the type „APILNI_Message" which is also noted as Application Layer Buffer. Please have a look at chapter 11 form more information about this buffer. The size of this data structure is variable. For each specific call the size of this variable data structure is set by the parameter „uMsgLen".

Example:

```
LDVCode ldv_code;
APILNI_Message lni_msg = { niRESET, 0x00 }; // sending this message resets the LTA
                                            // hLni was initialized in the sample under ldv_open()
 ldv_code = ldv_write( hLni, (LPCVOID)&lni_msg, sizeof(APILNI_Message) );
```

Return value:

If the writing operation on the **XLON**® „xldv32.dll" has been successful the value „LDV_OK" is passed back. In case of an error a value according the LDV-codes specified in the „xldv32.h" header file is returned. More detailed information about the error cause can be passed to the application by calling the GetLastError() function which is described in chapter 10.1.9.

Error cause:

Possible error causes for a failure of this function are:

- Invalid LNI-handle
- Wrong format of the Application Layer Buffer
- The selected value in parameter „uMsgLen" does not match the actual length of the Application Layer Buffer or is higher than maximum size.

10.1.5 ldv_register()

Prototype:

LDVCode **ldv_register(** LNI *hLni*, LPCVOID *lpFunc*);

Description:

By this function an application can register a callback function for every communication channel at the **XLON**® „xldv32.dll". As soon as data is available the registered callback function is automatically called by the **XLON**® „xldv32.dll". By that way the application is notified when uplink data is available and could be read by the API function „ldv_read()". So the **XLON**® „xldv32.dll" must not be polled by the API function „ldv_read()".
The respective communication channel is specified by its handle „hLni". The pointer „lpFunc" must point on a valid callback function of the application. This function must be of the type „void CallbackFunction(void)" and meet the ANSI-C conventions.

Example:

```
void CbFunc( void ) {
...                          // Initiate a read from „xldv32.dll" in this function
}
...
LDVCode ldv_code;
                             // hLni was initialized in the sample under ldv_open()
ldv_code = ldv_register( hLni, (LPCVOID)CbFunc );
```

Return value:

If the registration of the callback function at the **XLON**® „xldv32.dll" has been successful the value „LDV_OK" is passed back. In case of an error a value according the LDV-codes specified in the „xldv32.h" header file is returned. More detailed information about the error cause can be passed to the application by calling the GetLastError() function which is described in chapter 10.1.9.

Error cause:

Possible error causes for a failure of this function are:

- Invalid LNI-handle
- Callback function already registered
- Error condition when allocating resources inside the **XLON**® „xldv32.dll"

10.1.6 ldv_get_version()

Prototype:

LPCTSTR **ldv_get_version**(*void*);

Description:

By using this function an application can read the version of the actual used ***XLON***® „xldv32.dll".
The driver version is coded in a string format of the type „TCHAR" and is already in a readable format.
Here is an example of an possible content of that string:
„XLON XLDV32.DLL Multi Client Version 1.2.0.0 (C) DH electronics GmbH 2003".

When using that API function, no LNI-handle is required, thus it can be used without calling the „ldv_open()" API function before. The ***XLON***® „xldv32.dll" returns a pointer to the version string.

Example:

```
LPCTSTR lpVersion;
lpVersion = ldv_get_version();      // request version string
_tprintf( lpVersion );              // Output version string
```

Return value:

If reading the version string has been successful a pointer to the version string is returned. In case of an error a NULL pointer is returned. In such a case more detailed information about the error cause can be passed to the application by calling the GetLastError() function which is described in chapter 10.1.9.

Error cause:

If the ***XLON***® „xldv32.dll" has been successfully loaded the version string can always be readout. A fail of the function therefore is very unlikely. Nevertheless in such a case there must be a serious problem.

10.1.7 ldv_debugmode()

Prototype:

LDVCode **ldv_debugmode**(LPCTSTR *lpFileName*, INT iMode);

Description:

This function of the XLON® „xldv32.dll" has only been implemented for compatibility reasons or rather for a later implementation if necessary. It has no function yet. The parameters „lpFileName" and „iMode" have no meaning yet.

Example:

```
LDVCode ldv_code;
LPCTSTR lpFileName = {„C:\\Temp\\xldv32.log"};
INT iMode = 1;
ldv_code = ldv_debugmode( lpFileName, iMode );
```

Return value:

If calling of the function has been successful the value „LDV_OK" is passed back. In case of an error a value according the LDV-codes specified in the „xldv32.h" header file is returned. More detailed information about the error cause can be passed to the application by calling the GetLastError() function which is described in chapter 10.1.9.

Error cause:

As this function is implemented only as a „dummy function" it has no effect on the application. It always returns „LDV_OK".

10.1.8 ldv_neuronID()

Prototype:

LDVCode **ldv_neuronID**(LPVOID *lpNID*);

Description:

This function of the XLON® „xldv32.dll" has only been implemented for compatibility reasons or rather for a later implementation if necessary. It has no function yet. The parameter „lpNID" has no meaning yet.

Example:

```
LDVCode ldv_code;
LPVOID lpFileName = NULL;
ldv_code = ldv_neuronID( lpNID);
```

Return value:

If calling of the function has been successful the value „LDV_OK" is passed back. In case of an error a value according the LDV-codes specified in the „xldv32.h" header file is returned. More detailed information about the error cause can be passed to the application by calling the GetLastError() function which is described in chapter 10.1.9.

Error cause:

As this function is implemented only as a „dummy function" it has no effect on the application. It always returns „LDV_OK".

10.1.9 GetLastError()

Prototype:

DWORD **GetLastError(** *void*);

Description:

This function gives detailed information about error cause if there is a failure in the functions „ldv_close()", „ldv_read()", „ldv_write()", „ldv_register()", „ldv_getversion()", „ldv_debugmode()", or „ldv_neuronID()". Find detailed information about the function GetLastError() in the Windows Platform SDK documentation.

Example:

```
DWORD dwLastError;
dwLastError = GetLastError();
```

Return value:

Error code of the last operation. At the beginning of each API function the error code is set to „XLDV_OK". When calling the „GetLastError()" function it must always return „XLDV_OK" as long as no error occured. The following error codes are defined inside the DLL:

Error code	Value	Description
XLDV_OK	0x20001000	Operation successful.
XLDV_NOT_FOUND	0x20002100	Ressource not found.
XLDV_ALREADY_OPEN	0x20002200	Ressource already opened.
XLDV_NOT_OPEN	0x20002300	Ressource not opened.
XLDV_DEVICE_ERROR	0x20002400	Device error occured.
XLDV_NO_MSG_AVAIL	0x20002500	No data available.
XLDV_NO_RESOURCES	0x20002600	Not enough resources available.
XLDV_OPEN_ERROR	0x2000A100	Error while opening.
XLDV_NO_LICENSE	0x2000A200	No license for this operation.
XLDV_NOT_CONNECTED	0x2000A300	No connection available.
XLDV_NO_DRIVER	0x2000B100	Device driver not available.
XLDV_COMERROR	0x2000B200	Communication error occured.
XLDV_NO_THREAD	0x2000B300	Thread ressource not available.
XLDV_HANDLE_INVALID	0x2000B400	Invalid handle.
XLDV_OTHERDEVICE	0x2000B500	Ressource occupied by another device.
XLDV_NOPIPE	0x2000B600	Communication channel not available.
XLDV_READPIPE	0x2000B700	R/W error on Communication channel.
XLDV_NEURONID	0x2000B800	Error while reading Neuron ID.

11 Application Layer Buffer format

In the following you will find a description of the structure of the »Application Layer Buffer« which is used as a parameter for the API functions „ldv_read()„ and „ldv_write()“ of the ***XLON***® „xldv32.dll“. You will find additional informations in Echelon's „NSI Firmware User's Guide“ and „LonWorks Host Application Programmer's Guide“.

Application Layer Buffer:

The following C-code defines the structure „APILNI_Message“ as Application Layer Buffer according to Echelon's „NSI Firmware User's Guide“. Please refer to Echelon's „LonWorks Host Application Programmer's Guide“ for a detailed description of the „ExpAppBuffer[]“.

```
#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 following table shows the structure of the Application Layer Buffer graphically.

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

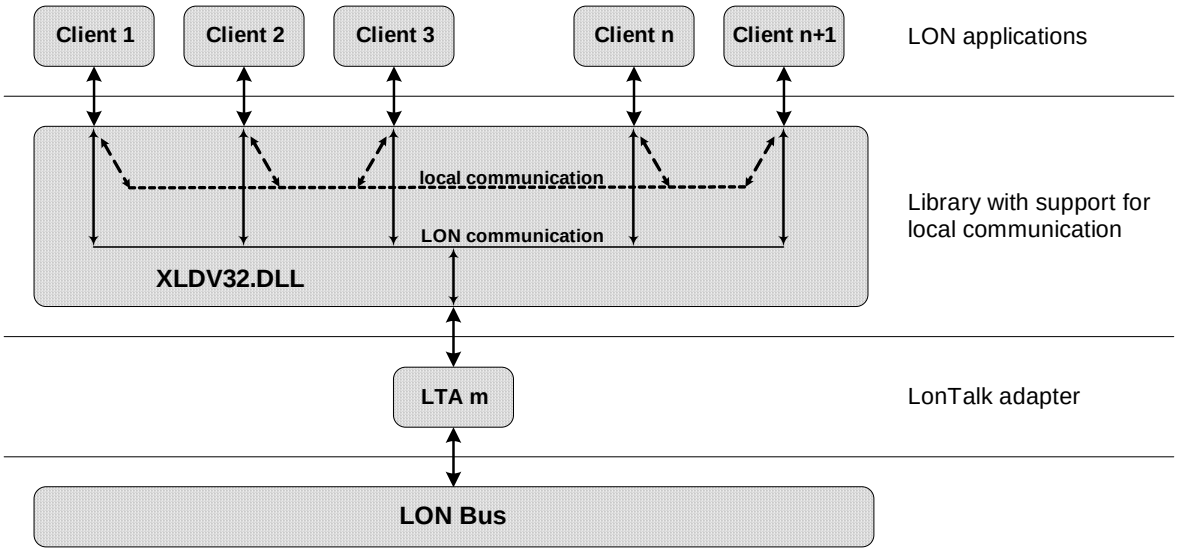
12 Local communication support

In special cases, it is required that clients of the EXLON® „xldv32.dll" communicate to each other. For clients using different EXLON® LTAs (LonTalk adapter), this is implicitely possible, because the communication goes over the LON network. For clients using the same EXLON® LTA, this is possible as well under specific preconditions. The first precondition is, that the domain table of the EXLON® LTA has been initialized before by using the network-management-command "UpdateDomain" (message-code 0x63). This enables the program-logic of the EXLON® „xldv32.dll", to recognize messages which are addressed to another client on the same system. The second precondition is, that the clients sending such messages use only addressing modes "Subnet/Node" or "Broadcast" and service types „Acknowledged", „Unacknowledged" or „Unacknowledged Repeated".

If all preconditions are fulfilled, all outgoing messages using addressing mode "Subnet/Node", which specifying the address of the local EXLON® LTA (as previously set by "UpdateDomain") as receiver address, are transformed into incoming messages and distributed to all other clients. Outgoing messages with addressing mode "Broadcast" are sent over LON as well as they are transformed into incoming messages and distributed to all other clients. Completion-messages, as usually generated by the NSI, are forwarded to all clients, even on local communication. The sender client does not receive it's own messages sent.

The fields „Retry Counter", „Repetition Timer" and „Transmission Timer" of an outgoing, exclusively local addressed message are set to „0" by the EXLON® „xldv32.dll". This serves the purpose to avoid unnecessary repetitions and idle times on the LON-Bus. To understand the background of this, it must be mentioned that even exclusively local addressed messages are sent to the network interface. On the one hand, this serves to generate completion-messages as expected by the sending client. On the other hand, this serves to be able to show local messages within a LON-protocol-analyzer.

XLON XLDV32.DLL Programmer's Guide



13 Programming Example

There is no programming example available yet. Please contact us by email info@xlon.de.

14 Version

Version	Date	Changes	Status	Author	Remarks
0.1	01.10.2003	Creation	internal	RG	Documentation for XLON® „xldv32.dll".
0.2	11.10.2003	English translation	internal	SD	
0.3	30.03.2005	Extension	released	RG	Callback in multi-client mode required.
0.4	03.08.2005	Extension	released	RG	Callback in multi-client mode no longer required, local communication supported.
0.5	11.10.2005	Extension	released	RG	Extensions regarding Rpt_timer and Tx_timer when local communication is used. New document template.