XLON XLDV32.DLL Programmieranleitung

1 Inhaltsverzeichnis

XLON XLDV32.DLL Programmieranleitung

2 Allgemein

Die Schnittstellen der Gerätetreiber für LonTalk Adapter (LTA) unterscheiden sich je nach eingesetztem Betriebssystem. Ebenfalls konnte pro LonTalk Adapter immer nur eine Applikation exklusiv auf den Gerätetreiber zugreifen. Sollte aus einer Applikation auf mehr als einen LTA zugegriffen werden, so war zusätzlicher Implementierungsaufwand nötig.

Die ***XLON***® „xldv32.dll“ Software-Bibliothek wurde entwickelt, um alle diese Einschränkungen und Nachteile aufzuheben und dem Programmierer eine möglichst einfach zu programmierende Schnittstelle an die Hand zu geben. Der Programmierer soll sich auf das wesentliche konzentrieren können und nicht jedes Mal das Rad neu erfinden müssen, wenn der Zugriff auf einen LonTalk Adapter nötig ist.

3 Eigenschaften

Die ***XLON***® „xldv32.dll“ Software-Bibliothek bietet vor dem in Kapitel 2 erläuterten Hintergrund die folgenden Eigenschaften:

- Möglichst einfach zu benutzende Applikations-Schnittstelle (API) für den Zugriff auf LON Netzwerk-Interfaces, sogenannte LonTalk Adapter oder kurz LTAs.
- Größtmögliche Flexibilität bei der Erstellung eigener LON Applikationen.
- Möglichst große Zahl von Betriebssystemen, auf der alle Eigenschaften der Bibliothek genutzt werden können. (Plattformunahängigkeit)
- Einheitliche Applikationsschnittstelle unter verschiedenen Betriebssystemen für den Zugriff auf LonTalk Adapter.
  (Multi-OS Fähigkeit)
- Einfacher Zugriff auf LonTalk Adapter aus einer oder mehreren Applikationen.
  (Multi-Client Fähigkeit)
- Einfacher Zugriff von Applikationen auf ein oder mehrere LonTalk Adapter.
  (Multi-Interface Fähigkeit).
- Schaffen einer Alternative zur kostenpflichtigen „wldv32.dll“ bei der Verwendung von ***XLON***® LonTalk Adaptern.

Die ***XLON***® „xldv32.dll“ Software-Bibliothek wurde unter Microsoft Visual Studio 6.0 in C++ entwickelt. Die Aufrufkonvention der Funktionen der Applikations-Schnittstelle (API) entspricht der ANSI-C Syntax.

4 Unterstützte Betriebssysteme

Die XLON® „xldv32.dll“ Software-Bibliothek unterstützt derzeit die folgenden Betriebssysteme:

- 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

Die Unterstützung für folgende Betriebssysteme durch die XLON® „xldv32.dll“ Software-Bibliothek ist geplant:

- Windows CE 3.0
- Windows CE .NET
- Linux

5 Unterstützte XLON® LonTalk Adapter

Die ***XLON***® „xldv32.dll“ Software-Bibliothek unterstützt die folgenden ***XLON***® LTAs, sofern für das jeweilige Betriebssystem ein entsprechender 32-Bit Treiber verfügbar ist. Veraltete 16-bit MS-DOS Treiber für die Windows 9x Betriebssysteme werden nicht mehr unterstützt.

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

Zum Erwerb einer Lizenz für die Verwendung der ***XLON***® „xldv32.dll“ mit LonTalk Adaptern von Drittanbietern kontaktieren Sie bitte DH electronics.
Nähere Informationen erhalten Sie auf der ***XLON***® Homepage unter www.xlon.de oder per E-Mail unter info@xlon.de.

Die folgende Tabelle gibt einen Überblick über mögliche Gerätenamen der einzelnen ***XLON***® LTAs. Bitte beachten Sie, dass die Indizes am Ende der Gerätenamen je nach der Anzahl der verwendeten ***XLON***® LTAs variieren können.

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:

Übersicht über mögliche Gerätenamen für ***XLON***® *LTAs unter verschiedenen Betriebssystemen.*

XLON XLDV32.DLL Programmieranleitung

Um den korrekten Gerätenamen für den verwendeten ***XLON***® LTA zu bestimmen, wird auch auf die ausführliche Bedienungsanleitung für die ***XLON***® LTAs verwiesen. Diese können unter www.xlon.de heruntergeladen werden. Dort finden sich auch noch zahlreiche weitere Informationen zum jeweiligen ***XLON***® LTA.

Unter Windows 32 können die Gerätenamen der installierten ***XLON***® LTAs laut Echelon-Spezifikation unter dem folgenden Registrierungspfad ausgelesen werden:

- „HKEY_LOCAL_MACHINE\SOFTWARE\LonWorks\DeviceDrivers“

6 Kompatibilität zur wldv32.dll von Gesytec

Die Applikationsschnittstelle (API) der ***XLON***® „xldv32.dll“ Software-Bibliothek ist kompatibel zur „wldv32.dll“ Standard und „wldv32.dll“ Multi-Client Version der Firma Gesytec. Durch die „xldv32.dll“ können die Eigenschaften der „wldv32.dll“ Multi-Client Version nun auch unter Windows 9x und Windows CE genutzt werden.
Durch umbenennen der ***XLON***® „xldv32.dll“ in „wldv32.dll“ kann eine beliebige Version der „wldv32.dll“ völlig transparent ersetzt werden.
Da die ***XLON***® „xldv32.dll“ keine Windows NT spezifischen Dienste und Mechanismen wie z.B. „OLE Automation Server“, „EXE Server“ oder COM nutzt, kann der volle Funktionsumfang unter allen unterstützten Betriebssystemen, auch unter Windows 9x und Windows CE, genutzt werden, insbesondere sind dies die „Multi-Client“ und „Multi-Interface“ Fähigkeit. Als weiteren Vorteil des Verzichts auf diese plattformspezifischen Dienste kann genannt werden, dass keine Registrierung der ***XLON***® „xldv32.dll“ oder eine ihrer Komponenten nötig ist, d.h. die ***XLON***® „xldv32.dll“ kann beliebig kopiert, verschoben, gelöscht, ausgetauscht, umbenannt usw. werden.

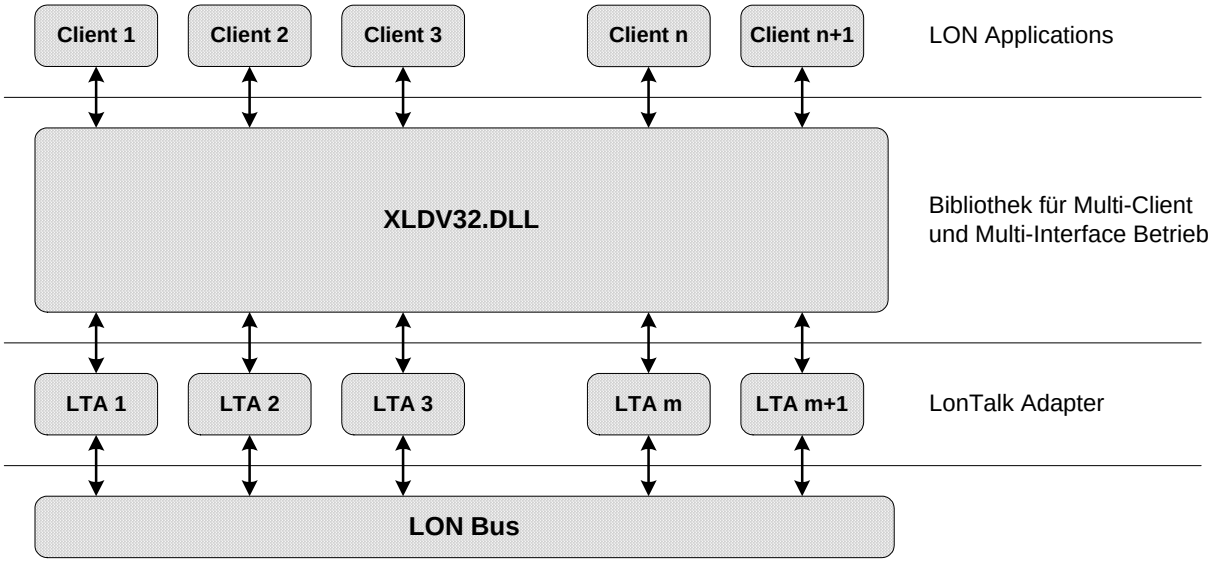
7 Multi-Client und Multi-Interface Betriebsart

Wird die „xldv32.dll“ in der Multi-Interface Betriebsart genutzt, so werden von einer Applikation (Client) mehrere ***XLON***® LTAs geöffnet.

Wird die „xldv32.dll“ in der Multi-Client Betriebsart genutzt, d.h. mehrere Applikationen (Clients) greifen auf einen ***XLON***® LTA zu, so werden alle empfangenen LON-Messages dieses LTAs an alle Clients weitergeleitet.

Selbstverständlich können in der Multi-Client Betriebsart auch mehrere ***XLON***® LTAs geöffnet werden, d.h. Multi-Client und Multi-Interface Betriebsart können beliebig kombiniert werden.

In der Multi-Interface- und Multi-Client-Betriebsart ist es zwingend erforderlich, eine Callback-Funktion wie in Kapitel 10.1.5 beschrieben zu registrieren.



8 Installation unter Windows Betriebssystemen

Eine spezielle Installation der ***XLON***® „xldv32.dll“ ist nicht nötig. Die ***XLON***® „xldv32.dll“ kann mit der zu installierenden LON Applikation mitinstalliert oder manuell auf ein Zielsystem kopiert werden. Soll die ***XLON***® „xldv32.dll“ mehreren Applikationen zur Verfügung stehen, so empfiehlt sich eine Installation bzw. das Kopieren in folgende Verzeichnisse:

- „Windows\System“ für Windows 9x basierende Betriebssysteme (Windows 95, 98, 98SE, ME).
- „Windows\System32“ für Windows NT basierende Betriebssysteme (Windows NT/2000/XP).

Soll die ***XLON***® „xldv32.dll“ nur von einer LON Applikation genutzt werden, so genügt es sie in das lokale Verzeichnis zu kopieren in der sich die LON Applikation befindet.
Sollen mehrere Applikationen auf die ***XLON***® „xldv32.dll“ zugreifen, so muss sichergestellt sein, dass jede Applikation die gleiche Instanz der ***XLON***® „xldv32.dll“ öffnet. Dies ist durch die Verwendung der oben vorgeschlagenen Verzeichnisse möglich.
Ebenso werden bei der Verwendung der ***XLON***® „xldv32.dll“ keine Registrierungsanträge angelegt bzw. benötigt. In der Praxis ein Punkt der nicht zu unterschätzen ist, da Zugriffe auf die Windows Registry eine häufige Fehlerquelle sind, vor allem wenn Produkte mehrfach (de)installiert werden müssen oder in verschiedenen Versionen auf einem System vorhanden sein müssen.

9 Verwendung in eigenen Applikationen

Die XLON® „xldv32.dll“ bei der Programmierung eigener Applikationen unter Microsoft Visual C++ 6.0 zu nutzen ist sehr einfach. Es sind lediglich die folgenden Schritte nötig:

- In alle Quellcode-Module, in denen Funktionen der XLON® „xldv32.dll“ genutzt werden sollen, muss das Headerfile „xldv32.h“ eingebunden werden.
- Die statische Bibliothek „xldv32.lib“ muss dem Linker bekanntgegeben werden und sorgt dafür, dass die „xldv32.dll“ automatisch zur Laufzeit geladen wird. Unter Microsoft Visual C++ 6.0 geschieht dies über den Menüpunkt „Projekt->Einstellungen“. Anschließend öffnet sich der Projekteinstellungs-Dialog. Auf der Registerkarte „Linker“ wird die „xldv32.lib“ nun zu den Bibliotheks-Modulen hinzugefügt.
- Die dynamische Bibliothek „xldv32.dll“ wird nun zur Laufzeit automatisch beim Starten der Applikation geladen.

10 Applikationsschnittstelle

10.1 Applikationsschnittstelle unter Windows Betriebssystemen

Mit der ***XLON***® „xldv32.dll“ stehen dem Programmierer zum Zugriff aus einer eigenen C/C++ Applikationen auf einen ***XLON***® LTA die folgenden Funktionen der Applikations-Schnittstelle (API) der ***XLON***® „xldv32.dll“ zur Verfügung:

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

In den folgenden Unterkapiteln wird ein Überblick über die einzelnen API-Funktionen gegeben.

10.1.1 ldv_open()

Prototyp:

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

Beschreibung:

Diese Funktion öffnet den durch „lpDeviceName“ spezifizierten ***XLON***® LTA. Um den korrekten Device Namen für den verwendeten ***XLON***® LTA zu bestimmen, wird auf die ausführliche Bedienungsanleitung für die ***XLON***® LTAs verwiesen. Diese können unter www.xlon.de heruntergeladen werden. Dort finden sich auch noch zahlreiche weitere Informationen zum jeweiligen ***XLON***® LTA. Eine Übersichtstabelle zu möglichen Gerätenamen findet sich im Kapitel 5 dieses Dokumentes. Aus Kompatibilitätsgründen kann vor dem eigentlichen Gerätenamen noch der Präfix „LDV/“ verwendet werden, dies ist allerdings nicht nötig und wird für neue Entwicklungsprojekte nicht mehr empfohlen.

Beispiel:

```
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
```

Rückgabewert:

Konnte die ***XLON***® „xldv32.dll“ den spezifizierten ***XLON***® LTA erfolgreich öffnen, wird ein Handle zurückgegeben, der die soeben eröffnete Kommunikationsverbindung zwischen Applikation und ***XLON***® LTA eindeutig kennzeichnet. Über diesen Handle können dann die weiteren Funktionen der Applikationsschnittstelle der ***XLON***® „xldv32.dll“ aufgerufen werden.
Im Fehlerfall wird ein Wert entsprechend der im Headerfile „xldv32.h“ definierten LDV-Codes zurückgeliefert. Detailliertere Informationen zur Fehlerursache können dann über den Aufruf der Funktion GetLastError() erlangt werden, welche in Kapitel 10.1.9 näher beschrieben ist.

Fehlerursachen:

Als wahrscheinliche Fehlerursachen für ein Fehlschlagen dieser Funktion sollten folgende Punkte in Erwägung gezogen werden:

- ***XLON***® LTA ist nicht vorhanden oder nicht angeschlossen.
- Gerätename ist falsch.
- Gerätetreiber ist nicht geladen oder bereits geöffnet.

10.1.2 ldv_close()

Prototyp:

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

Beschreibung:

Diese Funktion schließt die über den Handle „hLni“ spezifizierte Kommunikationsverbindung mit einem **XLON**® LTA. Als Funktionsparameter ist der bei erfolgreichem Aufruf der Funktion ldv_open() zurückgelieferte LNI-Handle zu übergeben.

Beispiel:

```
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()
```

Rückgabewert:

Konnte die Kommunikationsverbindung erfolgreich geschlossen werden, wird der Wert „LDV_OK“ zurückgegeben, ansonsten wird ein anderer der im Headerfile „xldv32.h“ spezifizierten LDV-Codes zurückgegeben. In letzterem Fall können detailliertere Informationen zur Fehlerursache über den Aufruf der Funktion GetLastError() erlangt werden. Nach einem erfolgreichen Aufruf von ldv_close() ist der übergebene LNI-Handle nicht mehr gültig und kann nicht mehr für den Aufruf von Funktionen der Applikations-Schnittstelle der **XLON**® „xldv32.dll“ genutzt werden

Fehlerursachen:

Als wahrscheinlichste Fehlerursachen für ein Fehlschlagen dieser Funktion sollten folgende Punkte in Erwägung gezogen werden:

- LNI-Handle ist ungültig.
- Kommunikationsverbindung mit diesem LNI-Handle wurde bereits geschlossen.

10.1.3 ldv_read()

Prototyp:

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

Beschreibung:

Mit dieser Funktion werden Daten durch eine Applikation von der ***XLON***® „xldv32.dll“ und somit von einem ***XLON***® LTA gelesen. Aufrufe dieser Funktion geschehen asynchron, d.h. die Funktion kehrt sofort zurück sobald die Daten von einem internen Puffer der ***XLON***® „xldv32.dll“ übernommen wurden. Falls keine Daten zur Verfügung stehen oder ein Fehler aufgetreten ist, kehrt diese Funktion ebenfalls sofort zurück, d.h. es wird nicht auf das Eintreffen von Daten von einem ***XLON***® LTA gewartet.

Der jeweilige Kommunikationskanal wird über seinen Handle „hLni“ spezifiziert. Der Zeiger „lpMsg“ muss auf einen Puffer zeigen, der in der Lage ist eine Datenstruktur vom Typ „APILNI_Message“ aufzunehmen. Ein solcher Puffer wird auch als Application Layer Buffer bezeichnet. Die Größe dieses Puffers ist variabel, sollte jedoch bei Leseoperationen immer auf den Maximalwert von 255 Bytes gesetzt werden, da die Anzahl der tatsächlich zu lesenden Daten beim Aufruf der Funktion noch nicht bekannt ist. Der Aufbau einer Datenstruktur vom Typ „APILNI_Message“ ist im Kapitel 11 dargestellt. Über den Parameter „uMsgLen“ wird die für den jeweiligen Aufruf gültige Größe dieser variablen Datenstruktur festgelegt.

Beispiel:

```
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) );
```

Rückgabewert:

Konnte die Leseoperation auf die ***XLON***® „xldv32.dll“ erfolgreich durchgeführt werden, wird der Wert „LDV_OK“ zurückgegeben, ansonsten wird ein anderer der im Headerfile „xldv32.h“ spezifizierten LDV-Codes zurückgegeben. In diesem Fall können detailliertere Informationen zur Fehlerursache über den Aufruf der Funktion GetLastError() erlangt werden.

Fehlerursachen:

Als wahrscheinlichste Fehlerursachen für ein Fehlschlagen dieser Funktion sollten folgende Punkte in Erwägung gezogen werden:

- Im Gerätetreiber stehen keine Daten zum Lesen bereit.
- LNI-Handle ist ungültig.
- Die im Parameter „uMsgLen“ übergebene Größe des »Application Layer Buffers« reicht nicht aus, um die bereitstehenden Daten aufzunehmen.

10.1.4 ldv_write()

Prototyp:

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

Beschreibung:

Mit dieser Funktion werden Daten von einer Applikation an die ***XLON***® „xldv32.dll“ und somit an einen ***XLON***® LTA übermittelt. Aufrufe dieser Funktion geschehen asynchron, d.h. die Funktion kehrt sofort zurück sobald die Daten in einen internen Puffer der ***XLON***® „xldv32.dll“ übernommen wurden oder dabei ein Fehler aufgetreten ist. Die Verarbeitung der Daten verläuft dann parallel zur Applikation im Hintergrund.

Der jeweilige Kommunikationskanal wird über seinen Handle „hLni“ spezifiziert. Der Zeiger „lpMsg“ muss auf eine Datenstruktur vom Typ APILNI_Message zeigen, die auch als Application Layer Buffer bezeichnet wird. Die Größe dieser Datenstruktur ist variabel, der Aufbau ist im Kapitel 11 dargestellt. Über den Parameter „uMsgLen“ wird die für den jeweiligen Aufruf gültige Größe dieser variablen Datenstruktur festgelegt.

Beispiel:

```
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) );
```

Rückgabewert:

Konnte die Schreiboperation auf die ***XLON***® „xldv32.dll“ erfolgreich durchgeführt werden, wird der Wert „LDV_OK“ zurückgegeben, ansonsten wird ein anderer der im Headerfile „xldv32.h“ spezifizierten LDV-Codes zurückgegeben. In diesem Fall können detailliertere Informationen zur Fehlerursache über den Aufruf der Funktion GetLastError() erlangt werden.

Fehlerursachen:

Als wahrscheinliche Fehlerursachen für ein fehlschlagen dieser Funktion kommen ein ungültiger LNI-Handle, ein fehlerhaft aufgebauter Application Layer Buffer oder ein außerhalb des gültigen Bereichs liegender bzw. nicht mit der tatsächlichen Länge des Application Layer Buffers übereinstimmender Parameter „uMsgLen“ in Frage.

10.1.5 ldv_register()

Prototyp:

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

Beschreibung:

Mit dieser Funktion kann eine Applikation für jeden von ihr verwendeten Kommunikationskanal eine Callback-Funktion bei der **XLON**® „xldv32.dll“ registrieren. Sobald für den jeweiligen Kommunikationskanal Daten zur Verfügung stehen, wird die registrierte Callback-Funktion automatisch von der **XLON**® „xldv32.dll“ aufgerufen. Dadurch kann der Applikation signalisiert werden, wann Daten mittels der API-Funktion „ldv_read()“ gelesen werden können. Die **XLON**® „xldv32.dll“ braucht dann nicht mehr von der Applikation mittels der API-Funktion „ldv_read()“ gepollt werden.
Der gewünschte Kommunikationskanal wird über seinen Handle „hLni“ spezifiziert. Der Zeiger „lpFunc“ muss auf eine gültige Funktion der Applikation zeigen. Diese muss vom Typ „void CallbackFunction(void)“ sein und der ANSI-C Konvention entsprechen.

Beispiel:

```
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 );
```

Rückgabewert:

Konnte die Registrierung einer Callback-Funktion bei der **XLON**® „xldv32.dll“ erfolgreich durchgeführt werden, wird der Wert „LDV_OK“ zurückgegeben, ansonsten wird ein anderer der im Headerfile „xldv32.h“ spezifizierten LDV-Codes zurückgegeben. In diesem Fall können detailliertere Informationen zur Fehlerursache über den Aufruf der Funktion GetLastError() erlangt werden.

Fehlerursachen:

Als wahrscheinlichste Fehlerursachen für ein Fehlschlagen dieser Funktion sollten folgende Punkte in Erwägung gezogen werden:

- LNI-Handle ist ungültig.
- Callback-Funktion ist bereits registriert.
- Allokieren von Ressourcen innerhalb der **XLON**® „xldv32.dll“ fehlgeschlagen.

10.1.6 ldv_get_version()

Prototyp:

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

Beschreibung:

Mit dieser Funktion kann eine Applikation die Versionskennung der verwendeten XLON® „xldv32.dll“ auslesen. Die Treiberversion ist in Form eines Strings vom Typ „TCHAR“ kodiert und bereits in einer für den Benutzer lesbaren Form. Der Inhalt dieses Strings könnte z.B. folgendermaßen aussehen: „XLON XLDV32.DLL Multi Client Version 1.2.0.0 (C) DH electronics GmbH 2003“.
Für den Aufruf dieser API-Funktion wird kein LNI-Handle benötigt, d.h. diese Funktion kann auch ohne vorherigen Aufruf von „ldv_open()“ verwendet werden. Die XLON® „xldv32.dll“ liefert als Rückgabewert einen Zeiger auf den Versionskennungs-String.

Beispiel:

```
LPCTSTR lpVersion;

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

Rückgabewert:

Konnte der String mit der Versionskennung erfolgreich ausgelesen werden, wird ein Zeiger auf diesen String zurückgegeben, andernfalls wird der Wert „NULL“ zurückgegeben. In diesem Fall können detailliertere Informationen zur Fehlerursache über den Aufruf der Funktion GetLastError() erlangt werden.

Fehlerursachen:

Der Versionsstring kann immer ausgelesen werden, sobald die DLL erfolgreich geladen wurde. Ein Fehlschlagen dieser Funktion ist deshalb sehr unwahrscheinlich. Sollte dies doch der Fall sein, ist von einem generellen, schwerwiegenden Problem auszugehen.

10.1.7 ldv_debugmode()

Prototyp:

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

Beschreibung:

Diese Funktion der ***XLON***® „xldv32.dll“ wurde nur aus Kompatibilitätsgründen bzw. für spätere Erweiterungen implementiert. Sie hat momentan keine Funktion. Die Parameter „lpFileName“ und „iMode“ haben keine Bedeutung.

Beispiel:

```
LDVCode ldv_code;
LPCTSTR lpFileName = {„C:\\Temp\\xldv32.log“};
INT iMode = 1;

ldv_code = ldv_debugmode( lpFileName, iMode );
```

Rückgabewert:

Wurde die Funktion erfolgreich aufgerufen, wird der Wert „LDV_OK“ zurückgegeben, ansonsten wird ein anderer der im Headerfile „xldv32.h“ spezifizierten LDV-Codes zurückgegeben. In diesem Fall können detailliertere Informationen zur Fehlerursache über den Aufruf der Funktion GetLastError() erlangt werden.

Fehlerursachen:

Diese Funktion ist momentan nur als Dummy-Funktion implementiert. Ein Aufruf der Funktion hat keine Auswirkungen. Es wird immer „LDV_OK“ zurückgeliefert.

10.1.8 ldv_neuronID()

Prototyp:

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

Beschreibung:

Diese Funktion der **XLON**® „xldv32.dll“ wurde nur aus Kompatibilitätsgründen bzw. für spätere Erweiterungen implementiert. Sie hat momentan keine Funktion. Der Parameter „lpNID“ hat keine Bedeutung.

Beispiel:

```
LDVCode ldv_code;
LPVOID lpFileName = NULL;

ldv_code = ldv_neuronID( lpNID);
```

Rückgabewert:

Wurde die Funktion erfolgreich aufgerufen, wird der Wert „LDV_OK“ zurückgegeben, ansonsten wird ein anderer der im Headerfile „xldv32.h“ spezifizierten LDV-Codes zurückgegeben. In diesem Fall können detailliertere Informationen zur Fehlerursache über den Aufruf der Funktion GetLastError() erlangt werden.

Fehlerursachen:

Diese Funktion ist momentan nur als Dummy-Funktion implementiert. Ein Aufruf der Funktion hat keine Auswirkungen. Es wird immer „LDV_OK“ zurückgeliefert.

10.1.9 GetLastError()

Prototyp:

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

Beschreibung:

Diese Funktion liefert detailliertere Informationen zur Fehlerursache, wenn eine der API-Funktionen „ldv_close()“, „ldv_read()“, „ldv_write()“, „ldv_register()“, „ldv_getversion()“, „ldv_debugmode()“, oder „ldv_neuronID()“ fehlschlägt. Ausführlichere Informationen zur Funktion GetLastError() sind der Windows Platform SDK Dokumentation zu entnehmen.

XLON XLDV32.DLL Programmieranleitung

Beispiel:

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

Rückgabewert:

Fehlercode der letzten Operation. Zu Beginn jeder Operation wird der Fehlercode auf den Wert „XLDV_OK“ gesetzt. Wird somit nach einer Operation die Funktion „GetLastError()“ aufgerufen, so muss der Rückgabewert stets „XLDV_OK“ sein, solange kein Fehler aufgetreten ist. Für die XLON® „xldv32.dll“ sind die folgenden Fehlercodes definiert:

Fehlercode	Wert	Beschreibung
XLDV_OK	0x20001000	Operation erfolgreich.
XLDV_NOT_FOUND	0x20002100	Ressource nicht gefunden.
XLDV_ALREADY_OPEN	0x20002200	Ressource bereits geöffnet.
XLDV_NOT_OPEN	0x20002300	Ressource nicht geöffnet.
XLDV_DEVICE_ERROR	0x20002400	Gerätefehler aufgetreten.
XLDV_NO_MSG_AVAIL	0x20002500	Keine Daten verfügbar.
XLDV_NO_RESOURCES	0x20002600	Nicht genügend Ressourcen verfügbar.
XLDV_OPEN_ERROR	0x2000A100	Fehler beim Öffnen.
XLDV_NO_LICENSE	0x2000A200	Keine Lizenz für diese Operation.
XLDV_NOT_CONNECTED	0x2000A300	Keine Verbindung vorhanden.
XLDV_NO_DRIVER	0x2000B100	Gerätetreiber nicht verfügbar.
XLDV_COMERROR	0x2000B200	Kommunikationsfehler aufgetreten.
XLDV_NO_THREAD	0x2000B300	Thread-Ressource nicht verfügbar.
XLDV_HANDLE_INVALID	0x2000B400	Ungültiger Handle.
XLDV_OTHERDEVICE	0x2000B500	Ressource von anderem Gerät belegt.
XLDV_NOPIPE	0x2000B600	Kommunikationskanal nicht verfügbar.
XLDV_READPIPE	0x2000B700	R/W Fehler auf Kommunikationskanal.
XLDV_NEURONID	0x2000B800	Fehler beim Lesen der Neuron-ID.

11 Aufbau eines Application Layer Buffer

Im Folgenden wird der Aufbau eines »Application Layer Buffer« erläutert, wie er als Parameter in den Funktionsaufrufen der API-Funktionen „ldv_read()„ und „ldv_write()“ der ***XLON***® „xldv32.dll“ benötigt wird. Darüber hinausgehende Informationen hierzu finden sich auch im „Echelon NSI Firmware User's Guide“ bzw. im „LonWorks Host Application Programmer's Guide“.

Application Layer Buffer:

Der folgende C-Code definiert den Datentyp „APILNI_Message“ für Application Layer Buffer, wie im Echelon NSI Firmware User's Guide spezifiziert. Der Aufbau des Strukturelements „ExpAppBuffer[]“ wird im „LonWorks Host Application Programmer's Guide“ ausführlich erläutert.

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

Die folgende Tabelle zeigt nochmals den Aufbau des Application Layer Buffer in grafischer Form.

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

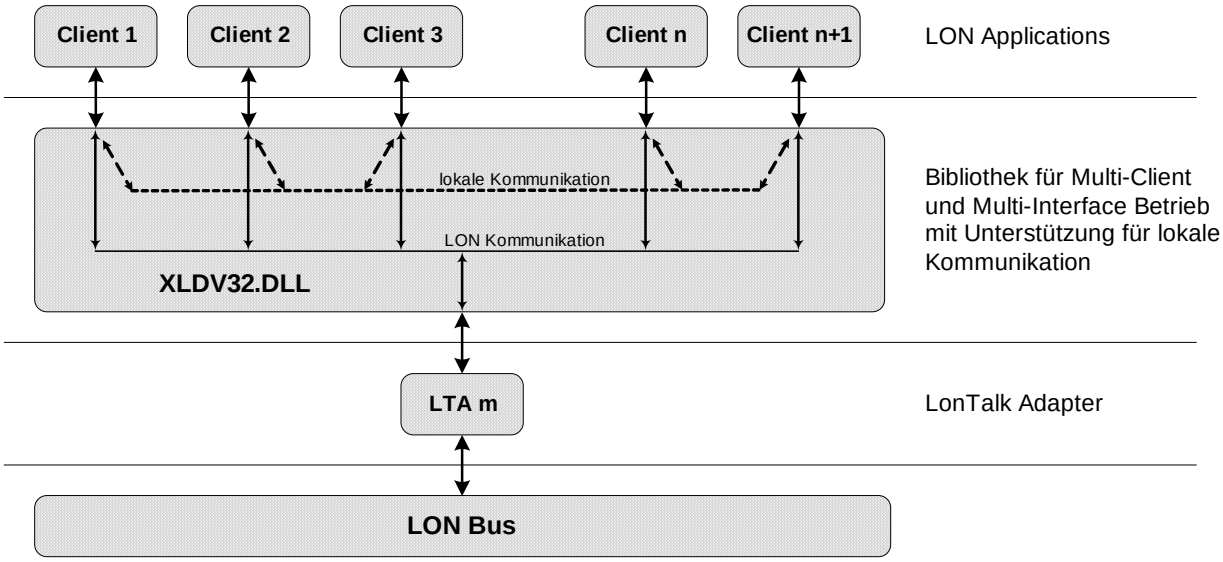
12 Unterstützung lokaler Kommunikation

Für manche Anwendungsfälle ist es nötig, dass die Clients der ***XLON***® „xldv32.dll“ untereinander kommunizieren. Verwenden die Clients unterschiedliche ***XLON***® LTAs (LonTalk Adapter), ist dies ohnehin implizit möglich, da die Kommunikation über das LON-Netzwerk stattfindet. Für Clients, die auf den gleichen ***XLON***® LTA registriert sind, ist dies unter bestimmten Voraussetzungen ebenfalls möglich. Die erste Voraussetzung ist, dass die Domaintabelle des ***XLON***® LTAs zuvor mit dem Netzwerk-Management-Kommando „UpdateDomain“ (Message-Code 0x63) initialisiert wurde, da die Programmlogik der ***XLON***® „xldv32.dll“ erst dann in der Lage ist, Nachrichten zu erkennen, die an Clients auf dem gleichen System adressiert sind. Die zweite Voraussetzung ist, dass die von den Clients versandten Nachrichten die Adressierungsarten „Subnet/Node“ bzw. „Broadcast“ verwenden und dass nur die Servicetypen „Acknowledged“, „Unacknowledged“ oder „Unacknowledged Repeated“ zum Einsatz kommen.

Sind die genannten Voraussetzungen erfüllt, so werden alle ausgehenden Nachrichten mit der Adressierungsart „Subnet/Node“, die als Empfängeradresse die Adresse des lokalen ***XLON***® LTAs eingetragen haben (wie zuvor mit „UpdateDomain“ programmiert), in eine eingehende Nachricht umgewandelt und an alle weiteren Clients weitergeleitet. Nachrichten der Adressierungsart „Broadcast“ werden sowohl auf das LON-Netzwerk als auch in umgewandelter Form als eingehende Nachricht an alle weiteren Clients geschickt. Die üblicherweise vom NSI auf ausgehende Nachrichten generierte Completion-Nachricht wird an alle Clients weitergeleitet, diese wird auch bei lokaler Kommunikation erzeugt. Der sendende Client erhält die von ihm gesendeten Nachrichten nicht selbst.

Die Felder „Retry Counter“, „Repetition Timer“ und „Transmission Timer“ einer ausgehenden, ausschließlich lokal adressierten Nachricht werden von der ***XLON***® „xldv32.dll“ zu „0“ gesetzt. Dies dient dazu, unnötige Wiederholungen und Wartezeiten auf dem LON-Bus zu vermeiden. Hintergrund ist, dass auch ausschließlich lokal adressierte Nachrichten stets zum Netzwerkinterface gesandt werden. Dies dient einerseits dazu, die vom sendenden Client erwartete Completion-Nachricht zu erzeugen und dient andererseits dazu, die lokalen Nachrichten auch in einem LON-Protokoll-Analysator darstellen zu können.

XLON XLDV32.DLL Programmieranleitung



13 Anwendungsbeispiel

Derzeit ist kein Anwendungsbeispiel verfügbar. Bitte kontaktieren Sie uns per Email unter info@xlon.de.

14 Version

Version	Datum	Änderung	Status	Autor	Bemerkung
0.1	01.10.2003	Neuerstellung	intern	RG	Doku für XLON® „xldv32.dll“.
0.2	14.10.2003	Überarbeitung	intern	SD	1. Korrektur der Doku von RG
0.3	30.03.2005	Erweiterung	freigegeben	RG	Multi-Client-Betrieb nur mit Callback.
0.4	03.08.05	Erweiterung	freigegeben	RG	Multi-Client-Betrieb ohne Callback, lokale Kommunikation wird unterstützt.
0.5	11.10.05	Erweiterung	freigegeben	RG	Erweiterung bzgl. Rpt_timer und Tx_timer bei lokaler Kommunikation. Neue Dokumentvorlage.