

EasyLevelIX

for

LabView™



Company of the ACTIA group



Systems, Support & More

Content

- 1 EasyLevelX..... 3
- 1.1 Overview 3
- 2 Installation..... 4
- 2.1 Necessary requirements: 4
- 2.2 Content of File EasyLevelX.ini 4
- 3 Interface description..... 5
- 4 How to use this Interface DLL with Labview 7
- 4.1 Hint 8

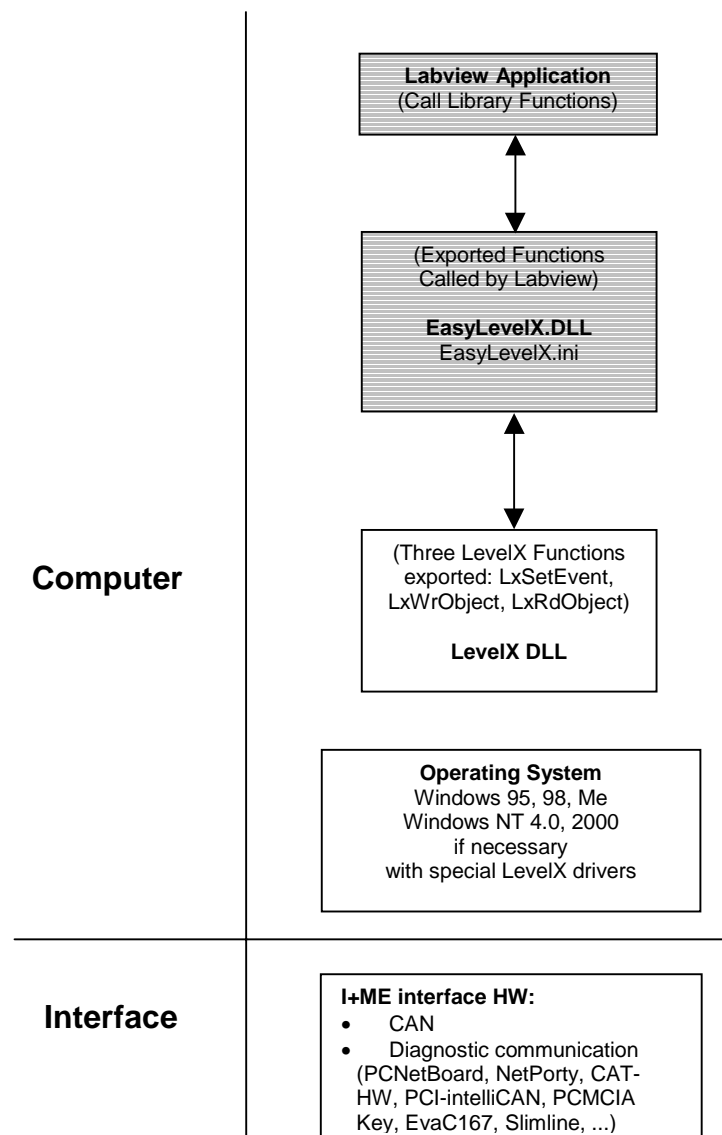
1 EasyLevelX

a “easy to use” interface DLL between LevelX and applications like LabVIEW™

1.1 Overview

LabVIEW applications can communicate with inputs/outputs by calling DLL functions. We have designed an interface library (EasyLevelX.DLL) which provides LabVIEW™ and other applications to use all the functions needed to drive an I+ME hardware.

The following diagram shows the different layer between a LabVIEW™ application and a I+ME hardware.



2 Installation

2.1 Necessary requirements:

- A LevelX interface driver and a LevelX hardware must be installed.
- LabVIEW™ must be installed (The Evaluation Package works very well).

Copy the following files

- LVdemoX.vi,
- EasyLevelX.dll
- EasyLevelX.ini

in any directory you want.

2.2 Content of File EasyLevelX.ini

[EasyLevelX]

LevelXDllName=Ixn4py2s

- define the necessary Interface DLL to links to the I+ME ACTIA hardware.
e.g.: Ixn4py2s.dll for the NetPorty and System Windows NT4.0

DriverOpenBoardNumber=1

- define the I+ME ACTIA hardware to be used by the tool. If installed e.g. 2 PCI-boards the 1st is number 0 and the 2st is number 1. For NetPorty it is the number of the serial communication port e.g. COM 1 is 1.

CanL2Channel=0

- define the number of the used CAN Channel. For CAN channel 1 is the number 0 and for CAN channel 2 is the number 1.
- e.g. PCI-boards have 2 CAN channels.

CanL2PhysicalDriver

- selects the physical Line Driver für CAN Communication
- 0 – Single Wire CAN Lowspeed
- 1 – Single Wire CAN Highspeed
- 2 – Dual Wire CAN Midspeed
- 3 – Dual Wire CAN Highspeed, (normal Mode)
- If the CanL2PhysicalDriver = 255 the option is disabled.

CanL2SpeedCode=9

- in following speed table you can see the number for the transfer speed. If the number of the CanL2SpeedCode = 255 are the entrys CanL2BTR0 and CanL2BTR1 used.

| Number | Speed |
|--------|------------|
| 0 | 10 kbps |
| 1 | 20 kbps |
| 2 | 50 kbps |
| 3 | 62.5 kbps |
| 4 | 75 kbps |
| 5 | 83.33 kbps |

| | |
|----|-----------|
| 6 | 100 kbps |
| 7 | 125 kbps |
| 8 | 250 kbps |
| 9 | 500 kbps |
| 10 | 800 kbps |
| 11 | 1000 kbps |

CanL2BTR0 and CanL2BTR1

- only active when CanL2SpeedCode=255
- to declared the value for BTR0 and BTR1 for user specifically definition of the bit timing. The value of BTR0 and BTR1 have to be calculated in accordance to the protocol chip specification. The used values are decimal.
- the CAN Bus speed is defined by using the BTR0 and BTR1 in the next line.

CanL2ConfReg=250

- the value is depend on the used hardware. In the registry under the Key HKEY_LOCAL_MACHINE\SOFTWARE\I+ME ACTIA GmbH\levelX\Interfaces\

3 Interface description

The file EasyLevelX.ini must be located in the EasyLevelX.DLL directory. The LevelX-DLL location is searched first over the registry and then in the EasyLevelX.DLL directory.

At this time only one CAN channel of a LevelX hardware can be used.

On many problems you get windows message boxes with a problem description from the LevelX interface.

The DLL functions calling conventions are “**stdcall**”.

ulnt8 **LevelXStart**(void);

function **LevelXStart**: byte;

Reads the EasyLevelX.ini, opens the LevelX-DLL, setups the CAN interface and starts the CAN bus (goes online).

– return: 0 – all ok; 1 – error, LevelX is not running

void **LevelXClose**(void);

procedure **LevelXClose**;

Closes all used resources.

void **LevelXTxData**(ulnt8 IdTypeXtd, ulnt32 Id, ulnt8 Rtr, ulnt8 DataLen, ulnt8 *pTxData);

procedure **LevelXTxData** (IdTypeXtd: byte; Id: DWORD; Rtr: byte; DataLen: byte; pTxData: tpLxData);

Transmits a CAN object.

- IdTypeXtd = CAN identifier type: 0 – standard; 1 – extended
- Id = CAN identifier: 11bit (std ID) or 29bit (xtD ID) value
- Rtr = RemoteRequestFlag: 0 – normal; 1 – RTR frame
- DataLen: 0..8
- pTxData = pointer on TxData byte array: the data to transmit are read from this address

Delphi: type tLxData = Array[0..7] of byte; tpLxData = ^tLxData;

C: typedef unsigned char tLxData[8]; typedef tLxData *tpLxData;

void **LevelXRxSetup**(ulnt8 RxNumber, ulnt8 Enable, ulnt8 IdTypeXtd, ulnt32 Id, ulnt8 Rtr);

procedure **LevelXRxSetup** (RxNumber: byte; Enable: byte; IdTypeXtd: byte; Id: DWORD; Rtr: byte);

Defines a CAN receive object in one of the in EasyLevelX available 16 receive buffers.

- RxNumber = the number of the receive buffer: 0..15
- Enable = switch ON or OFF the receive buffer: 0 – off; 1 – on
- IdTypeXtd = CAN identifier type: 0 – standard; 1 – extended
- Id = CAN identifier: 11bit (std ID) or 29bit (xtD ID) value
- Rtr = RemoteRequestFlag: 0 – normal; 1 – RTR frame

void **LevelXRxData**(ulnt8 RxNumber, ulnt8 *pDataLen, ulnt8 *pRxData, ulnt32 *pTimeStamp);

procedure **LevelXRxData** (RxNumber: byte; pDataLen: pbyte; pRxData: tpLxData; pTimeStamp: tpTimeStamp); Stdcall;

Reads the CAN receive object informations from one of the in EasyLevelX available 16 receive buffers.

- RxNumber = the number of the receive buffer: 0..15
- pDataLen = pointer on the unsigned 8bit DataLen variable: 0..8
- pRxData = pointer on RxData byte array: the received data are written to this address

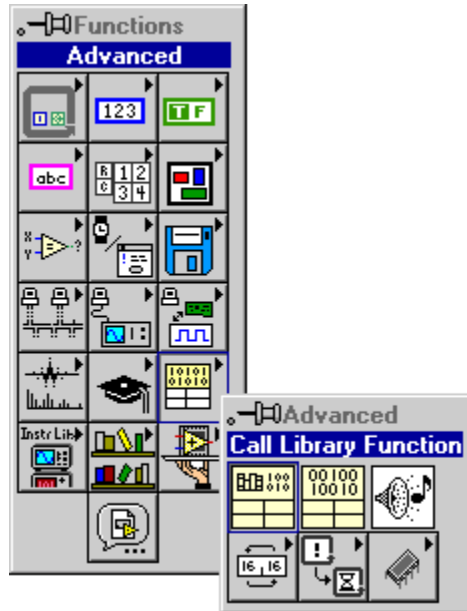
Delphi: type tLxData = Array[0..7] of byte; tpLxData = ^tLxData;

C: typedef unsigned char tLxData[8]; typedef tLxData *tpLxData;

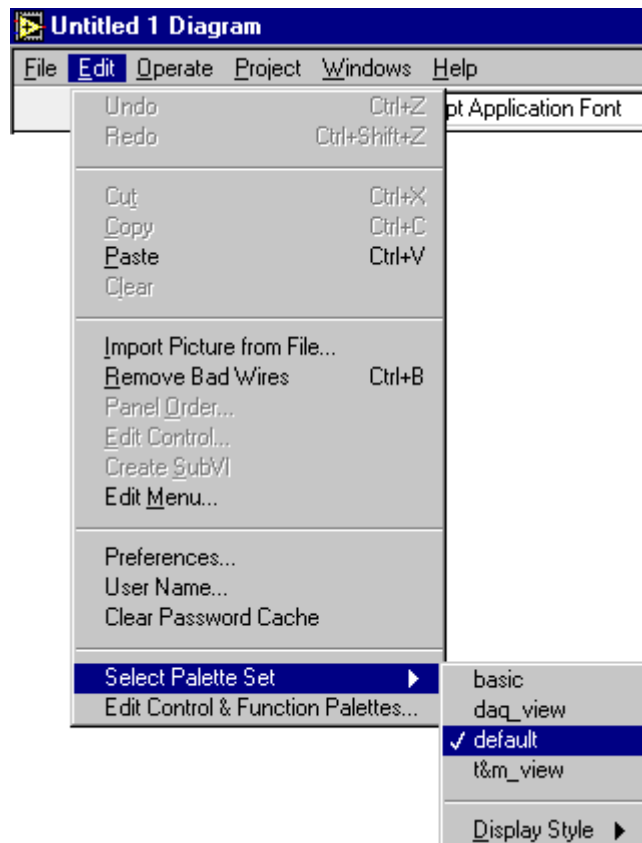
pTimeStamp = pointer on the unsigned 32bit TimeStamp variable: 0... (the LevelX system time)

4 How to use this Interface DLL with Labview

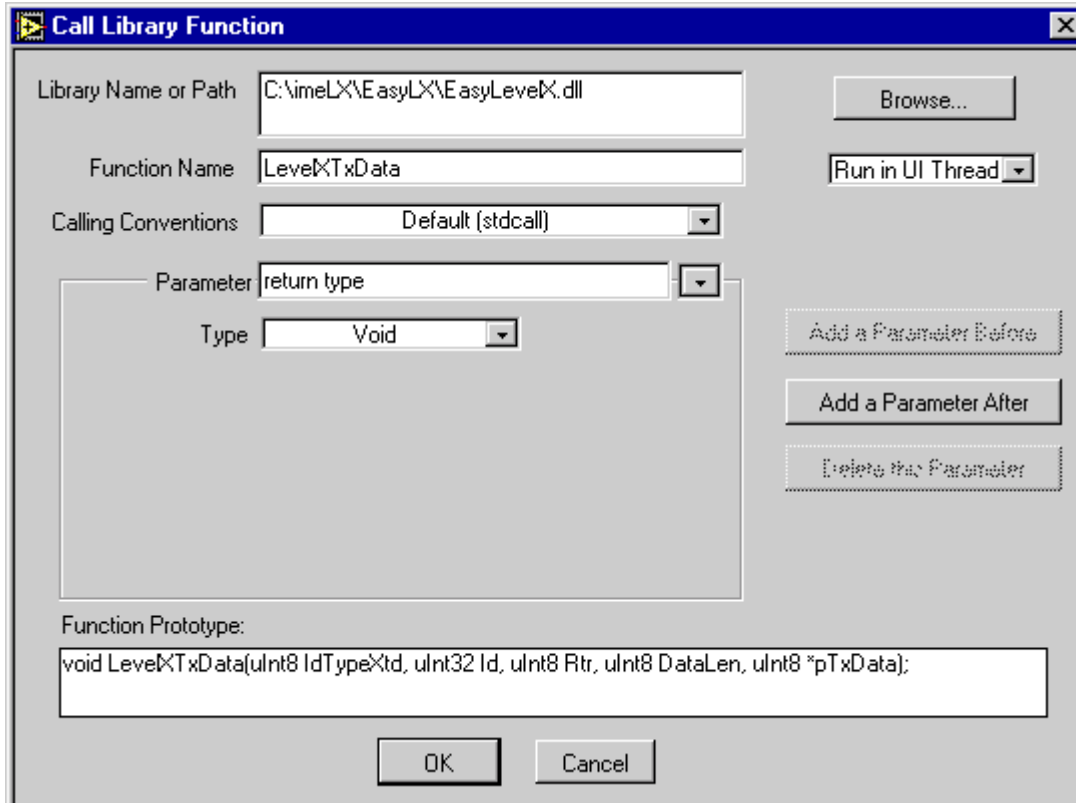
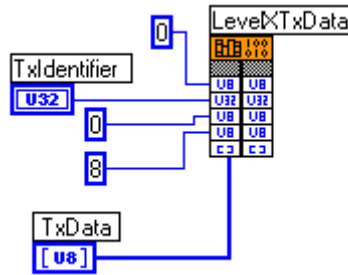
In order to drive a I+ME LevelIX hardware with a Labview application, you need to use the LabView™ function “Call Library Function”.



If you don't have the Advanced Option select under the menu Edit – Select Palette Set – default



To call functions described above and define name parameter linke described in the following screen:



4.1 Hint

When you work with Labview and the interface DLL, you must call always at first in the VI the function **LevelXStart** and at last always the function **LevelXClose**.

Note: To stop the VI do not use the stop button in the toolbar.

You can disable the stop and continuous button under vi setup with the right mouse button (see below).

