BiDiBLib V 0.0.1.0

Diese Library basiert auf den Definitionen des BiDiB®-Bussystems von Wolfgang Kufer.

http://www.bidib.org/

Version vom 30.10.2013

Autor: Andreas Tillner
Supportanfragen bitte über das OpenDCC-Forum
http://www.opendcc.de/forum.html
eMail: a.tillner@gmx.de

1 Inhalt

2 BiDiBLib.dll

Die Library BiDiBLib.dll umfasst eine Sammlung von Methoden, die folgende Funktionen abdecken:

- ✓ Initialisieren der seriellen Schnittstelle zum BiDiB-Interface.
- ✓ Erstellen der Messages (CRC-Berechnung, maskieren von 0xFE und 0xFD). Dabei muss nur die Knotenadresse, der Name der Message und die evtl. notwendigen Parameter übergeben werden.
- ✓ Senden der erstellten Messages an das Interface.
- ✓ Empfangen der Messages vom Interface.
- ✓ Die empfangenen Messages werden aus dem Datenstrom gefiltert und einzeln, nach CRC-Check, in einen FiFo (***MsgFiFo***) geschrieben.
- ✓ In der Variablen ***ProceedMSG*** kann eine Funktion hinterlegt werden, die die Weiterverarbeitung der Messages aus dem ***MsgFiFo*** übernimmt.

Der Quellecode liegt im ZIP-Archiv des Monitors unter `Sources\BiDiBLib-Source\BiDiBLib.cs` . Er basiert zum Teil auf den Beispielen von Wolgang Kufer (http://www.bidib.org/support/intro.html).
Da ich C# und die .NET-Umgebung noch nicht besonders gut kenne, sind Hinweise und Vorschläge zur Verbesserung des Codes immer willkommen.

2.1 Klassen der BiDiBLib.dll

```
public class MsgEventArgs : EventArgs
```

Eventklasse für PlugIn-Interface. Neu in V 0.0.0.6

Übergabe von BiDiB-Messages als Byte-Array.

```
byte[] _newmessage          Byte-Array der Message
byte[] _adr;                Byte-Array mit der Adresse des Knoten
```

```
public class TxtEventArgs : EventArgs
```

Eventklasse für PlugIn-Interface. Neu in V 0.0.0.6

Übergabe von Textmeldungen als String

```
string _newmessage;         String mit anzuzeigender Meldung
byte[] _adr;                Byte-Array mit der Adresse des Knoten
```

```
public class NodeFeature
```

Einheitliche Definition eines Node-Features für das PlugIn-Interface. Neu in V 0.0.0.6

```
byte FetID;                 Feature-ID
byte value;                 Wert des Features
```

```
public class BiDiBMsgHandler
```

Basis-Klasse dieser Library.

2.2 Variablen der Klasse BiDiBMsgHandler

```
public DataSet MessageList = new DataSet();
```
Enthält nach dem Aufruf von Init() eine Liste aller Messages aus BiDiB-Messages.xml
```
public byte[][] MsgFiFo = new byte[512][];
```
FiFo mit den eingehenden Messages. Es handelt sich hier um die reine Message. Schon ohne CRC-Byte und Maskierung von 0xFE und 0xFD.
```
public int MsgTail;
```
Zeiger auf die letzte, noch nicht weiter verarbeitete Message.
```
public int MsgHead;
```
Zeiger auf die zuletzt eingegangene Message.
```
public byte[][] InputFiFo = new byte[512][];
```
FiFo mit den eingehenden Bytes. Es handelt sich um den Datenstrom, der direkt vom Interface kommt. Dieser FiFo wird nur Innerhalb der Klasse für die Analyse der Messages verwendet und immer wieder ohne Prüfung überschrieben.
```
public int InputTail;
```
Zeiger auf das letzte, noch nicht verarbeitete Datenpaket.
```
public int InputHead;
```
Zeiger auf das, zuletzt eingegangene Datenpaket.
```
private int MsgCrc;
```
Merker für die CRC-Prüfung
```
private int MsgMi;
```
Merker des aktuellen Messageindex.
```
private bool escape_hot;
```
Merker ob ESC-Maskierung gefunden wurde
```
public int MsgFirst;
```
Merker für Begin der Kommunikation (erstes 0xFE)
```
public BiDiBSerialPort BiDiBSerPort = new BiDiBSerialPort();
```
Serielle Schnittstelle
```
public Form MainForm;
```
Aufrufende Windows-Forms-Anwendung (für BeginInvoke)
```
public delegate void ProceedMSGDe();
public ProceedMSGDe ProceedMSG;
```
Prozeduraufruf für die Weiterverarbeitung der Messages.
```
public bool ProceedMSGRunning;
```
Semaphore um zu verhindern, das ProceedMSG mehrfach aufgerufen wird. ProceedMSGRunning muss von ProceedMSG, nach abarbeiten des FiFo wieder auf False gesetzt werden.

Funktionen der Klasse BiDiBMsgHandler

2.2.1 Init ( )

```
/*****************************************************************************
 ** Name        : Init ( void )
 ** Parameter   : -
 ** Description : - Reading the Messagefile BiDiB-Messages.xml
 **               - Init the Message FiFo
 **               - Call Init(int, string) with the default size of 128 Byte
 **                 and the default Message-XML-File
 *****************************************************************************/
```

2.2.2 Init (string MsgXML)

```
/*****************************************************************************
 ** Name        : Init ( string MsgXML )
 ** Parameter   : string MsgXML = Full Path to the XML-File which contains
 **               the BiDiB-Messages
 ** Description : - Reading the Messagefile from MsgXML
 **               - Init the Message FiFo
 **               - Call Init(int, string) with the default size of 128 Byte
 **                 and the Message-XML-File MsgXML
*****************************************************************************/
```

2.2.3 Init (int msgsize, string MsgXML)

```
/*****************************************************************************
 ** Name        : Init ( int msgsize, string MsgXML )
 ** Parameter   : int msgsize   = max. size of a single message in byte
 **               string MsgXML = Full Path to the XML-File which contains
 **               the BiDiB-Messages
 ** Description : - Reading the Messagefile BiDiB-Messages.xml
 **               - Init the Message FiFo
 ** Return      : 1 >> OK
 **               2 >> No Message found in Message-XML-File
 *****************************************************************************/
```

2.2.4 InitSerPort(string comport, int baud)

```
/*****************************************************************************
 ** Name        : InitSerialPort(string comport, int baud)
 ** Parameter   : string comport : Comport as a string. COM5
 **             : int baud       : baudrate. 1000000, 115200 or 19200 alowed
 ** Description : Set the parameter of the serial port.
 ** Return      : string
 **               %%* = if the returnstring starts with %%, this is an error string
 **               All other strings are information or log.
 *****************************************************************************/
```

Diese Funktion initialisiert die serielle Schnittstelle. Als Baudrate sind nur die Werte 1000000, 115200 und 19200 erlaubt. Alle anderen ergeben eine Fehlermeldung bzw. wird ein Error-String zurückgegeben.

2.2.5 InitSerialPortReg(string RegSubKey)

```
/*************************************************************************
 ** Name        : InitSerialPortReg(string RegSubKey)
 ** Parameter   : string RegSubKey : Subkey in Registry under HKEY_LOCAL_USER
 ** Description : Read the config from Registry and set the
 **               parameter of the serial port.
 ** Return      : %%* = if the returnstring starts with %%, this is an error string
 **               All other strings are information or log.
 *************************************************************************/
```

Liest den Comport und die Baudrate aus der Registry des Computers. Der Parameter RegSubKey gibt dabei den Pfad in der Registry unterhalb von HKEY_CURRENT_USER an. Sind die Schlüssel „Comport“ und „Baudrate“ nicht vorhanden, werden sie angelegt und mit den Werten COM1 und 2400 gefüllt.

Beispiel: `InitSerialPortReg("Software\\BiDiBLib\\Beispiel")`

```
[HKEY_CURRENT_USER\Software\BiDiBLib\Beispiel]
"Comport"="COM5"
"Baudrate"="115200"
```

2.2.6 ExtractMsg(byte[] msg)

```
/*************************************************************************
 ** Name        : ExtractMsg(byte[] msg)
 ** Parameter   : byte[] msg   : array of bytes contains the data from serial port
 ** Description : build single messages from the datastream and save them
 **               to the Message FiFo
 ** Return      : -
 *************************************************************************/
```

Diese Funktion wird eigentlich nur intern verwendet. Sie ist aber public, um bei Simulation den Eingang von Messages zu testen.

2.2.7 SendMsgNoParam(byte[] adr, string bm, bool log, byte nu, bool sim)

```
/*************************************************************************
 ** Name        : SendMsgNoParam(byte[] adr, string bm, bool log, byte nu, bool sim)
 ** Parameter   : byte[] adr   : array of bytes contains the adress of the node
 **               string bm    : Message in plain text like MSG_BM_OCC
 **               bool log     : true = returns logoutput, false: returns status
 **               byte nu      : messagenumber
 **               bool sim     : true = simulation on, false = simulation off
 ** Description : build a message with one parameter and send it to the interface
 ** Return      : %%* = if the return string starts with %%, it is an error message
 **               All other strings are information or log.
 *************************************************************************/
```

Parameter	Wert	Bedeutung
Log	True	Wenn die Message gesendet wurde, wird ein Text der Form „OUT -- : MSG_SYS_PING FE 04 00 01 07 EE 2C FE“ zurückgegeben.
Log	False	Es werden nur die normalen Statuswert zurückgegeben „%%*“ für Fehler, „OK“ wenn Message gesendet wurde.
Sim	True	Die Message wird nicht an die Schnittstelle geschickt.
Sim	False	Die Message wird an die Schnittstelle geschickt.
Nu		Durchlaufende Nummer der Message bezogen auf den Knoten der angesprochen wird.
bm		Sprechender Name der Message. Z.B. MSG_SYS_PING
adr		Adresse des Knotens der angesprochen wird. Z.B. 0x00 0x00 0x00 0x04

2.2.8 SendMsgOneParam(byte[] adr, string bm, byte param1, bool log, byte nu, bool sim)

```
/**************************************************************************
 ** Name        : SendMsgOneParam(byte[] adr, string bm, byte param1, bool log, byte
nu, bool sim)
 ** Parameter   : byte[] adr   : array of bytes contains the adress of the node
 **               string bm    : Message in plain text like MSG_BM_OCC
 **               byte param1  : Byte of first parameter
 **               bool log     : true = returns logoutput, false: returns status
 **               byte nu      : messagenumber
 **               bool sim     : true = simulation on, false = simulation off
 ** Description : build a message with one parameter and send it to the interface
 ** Return      : %%* = if the returnstring starts with %%, this is an error string
 **               All other strings are information or log.
 **************************************************************************/
```

Bedeutung der Parameter wie unter 2.2.7

2.2.9 SendMsgTwoParam(byte[] adr, string bm, byte param1, byte param2, bool log, byte nu, bool sim)

```
/**************************************************************************
 ** Name        : SendMsgTwoParam(byte[] adr, string bm, byte param1, byte param2,
bool log, byte nu, bool sim)
 ** Parameter   : byte[] adr   : array of bytes contains the adress of the node
 **               string bm    : Message in plain text like MSG_BM_OCC
 **               byte param1  : Byte of first parameter
 **               byte param2  : Byte of second parameter
 **               bool log     : true = returns logoutput, false: returns status
 **               byte nu      : messagenumber
 **               bool sim     : true = simulation on, false = simulation off
 ** Description : build a message with two parameters and send it to the interface
 ** Return      : %%* = if the returnstring starts with %%, this is an error string
 **               All other strings are information or log.
 **************************************************************************/
```

Bedeutung der Parameter wie unter 2.2.7

2.2.10 SendMsgMultiParam(byte[] adr, string bm, int bnum, byte[] param, bool log, byte nu, bool sim)

```
/**************************************************************************
 ** Name        : SendMsgMultiParam(byte[] adr, string bm, int bnum, byte[] param,
bool log, byte nu, bool sim)
 ** Parameter   : byte[] adr   : array of bytes contains the adress of the node
 **               string bm    : Message in plain text like MSG_BM_OCC
 **               int bnum     : number of bytes of the parameter array
 **               byte[] param : array of byte with the paramters
 **               bool log     : true = returns logoutput, false: returns status
 **               byte nu      : messagenumber
 **               bool sim     : true = simulation on, false = simulation off
 ** Description : build a message with bnum parameters and send it to the interface
 ** Return      : %%* = if the returnstring starts with %%, this is an error string
 **               All other strings are information or log.
 **************************************************************************/
```

Bedeutung der Parameter wie unter 2.2.7

2.2.11 SendMSGToSerial(byte[] msgb, string MB, bool log, bool sim)

```
/************************************************************************
 ** Name        : SendMSGToSerial(byte[] msgb, string MB, bool log, bool sim)
 ** Parameter   : byte[] msgb : array of bytes contains the message
 **               string bm   : Message in plain text like MSG_BM_OCC
 **               bool log    : true: returns logoutput, false: returns status
 **               bool sim    : true = simulation mode on, false = simulation mode off
 ** Description : Find a Message in DataSet and return the Msg.-Number as Byte
 ** Return      : -       = Message not send
 **               OK      = Message was send
 **               logtext = Text for Logfile
 ************************************************************************/
```

Maskierung der Zeichen 0xFE und 0xFD und senden an die serielle Schnittstelle.

2.2.12 GetMessageNum(string BM)

```
/************************************************************************
 ** Name        : GetMessageNum(string bm)
 ** Parameter   : string bm
 **               bm = Message in plain text like MSG_BM_OCC
 ** Description : Find a Message in DataSet and return the Msg.-Number as Byte
 ** Return      : 255 = Message not found
 **               0 - 254 = messagenumber as a byte
 ************************************************************************/
```

Sucht in dem DataSet ***MessageList*** nach dem Text der Nachricht und gibt den Bytecode zurück.

2.2.13 GetMsgString(byte BN)

```
/************************************************************************
 ** Name        : GetMsgString(byte BN)
 ** Parameter   : string bm
 **               bm : Messagenumber in byte
 ** Description : Find a Msg.-Bumber in DataSet and return the Message as String
 ** Return      : 255 = Message not found
 **               0 - 254 = messagenumber as a byte
 ************************************************************************/
```

Sucht in dem DataSet ***MessageList*** nach dem Bytecode der Message und gibt die Textbezeichnung zurück.

2.2.14 BuildMsgMultiParam(byte[] adr, string bm, int bnum, byte[] param, byte nu)

```
/************************************************************************
** Name        : BuildMsgMultiParam(byte[] adr, string bm, int bnum, byte[] param,
byte nu )
** Parameter   : byte[] adr   : array of bytes contains the adress of the node
**               string bm    : Message in plain text like MSG_BM_OCC
**               int bnum     : number of bytes of the parameter array
**               byte[] param : array of byte with the paramters
**               byte nu      : messagenumber
** Description : build a message with bnum parameters and return the byte-array with
the message
** Return      : byte[] msgb : 0 = number of bytes, incl. 0,1 and 2
**                             1 = position of msg-counter
**                             2 = position of msg-number
**                             3 - n = Messagebytes
**               The build message is only valid, if msgb[0] > 0
************************************************************************/
```

2.2.15 SendByteMsg(byte[] msg, bool log, byte nu, bool sim)

```
/*************************************************************************
** Name        : SendByteMsg(byte[] param, bool log, byte nu, bool sim)
** Parameter   : byte[] param : array of byte with the Message
**                              0 = number of bytes, incl. 0 and 1
**                              1 = position of msg-number
**                              2 - n = Messagebytes
**               bool log     : true = returns logoutput, false: returns status
**               byte nu      : messagenumber
**               bool sim     : true = simulation mode on, false = simulation mode off
** Description : build a message with bnum parameters and send it to the interface
** Return      : string
**               %%* = if the returnstring starts with %%, this is an error string
**               All other strings are information or log.
*************************************************************************/
```

2.3 Beispielanwendung

Die Library liegt im Verzeichnis vom BiDiB-Monitor als Datei BiDiBLib.dll.

Die Library wird per `using BiDiBLib;` eingebunden.

Nach der Deklaration von
`BiDiBMsgHandler BiDiBMsg = new BiDiBMsgHandler();`

Müssen zwingend folgende Schritte durchgeführt werden:

1. Per `BiDiBMsg.Init();` werden die FiFo's initialisiert und das XML-File mit den Messages eingelesen.
2. Die aufrufende Form wird per `BiDiBMsg.MainForm = <Form1>;` bekannt gemacht.
3. Mit `BiDiBMsg.ProceedMSG = <Auswertung der Messages>;` wird die Funktion übergeben, die den FiFo der eingehenden Messages abarbeitet.
4. `BiDiBMsg.InitSerialPortReg("Software\\BiDiBLib\\Beispiel");` oder `BiDiBMsg.InitSerialPortReg("COM5", 115200);` initialisiert die Schnittstelle.

Danach kann mit `BiDiBMsg.BiDiBSerPort.BiDiBPort.Open();` die Schnittstelle geöffnet werden. Eingehende Daten lösen einen Event aus und werden an `BiDiBMsg.ExtractMsg` übergeben. Hier wird der Datenstrom in die einzelnen Messages zerlegt und diese in `MsgFiFo` gespeichert. Danach ruft `BiDiBMsg.ExtractMsg` die Funktion auf, die vorher in `BiDiBMsg.ProceedMSG` übergeben wurde.
Hier kann der Programmierer bestimmen, wie die Messages verarbeitet werden.

Dem BiDiB-Monitor liegt im Verzeichnis **sources** eine C#-Beispielanwendung bei.
Diese macht nichts anderes, als nach dem Klick auf den Button „Connect" eine Verbindung mit dem BiDiB-Interface aufzubauen und alle eingehenden Messages anzuzeigen.