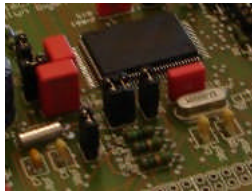


Hart Tools

TinyHartDLL 7.0

Software Documentation

Revision: 1
Date: 7.3.2010



Connecting Windows to the
World of Embedded Systems.

Borst Automation
Im Wingert 4
DE-65626 Fachingen
GERMANY

Fon: +49 (0)6432 989176
Fax: +49 (0)6432 989129

<http://borst-automation.com>

info@borst-automation.de

Borst Automation
Embedded Solutions

Copyright© 2006-2010 Borst Automation, Walter Borst, Fachingen, GERMANY

Hart® is a registered trademark of the Hart Communication Foundation
Windows® is a registered trademark of Microsoft Corporation

Contents

Overview	1
Typical Service Processing Program Flow	1
Architecture	2
Directory Structure	3
Getting Started	3
Distribution of Applications	4
Examples	4
C/C++	4
C#	5
Visual Basic	8
Excel	8
Getting Started	8
Coding Details	9
Adjust Working Directory	9
Set License Key	9
Open Port	9
Connect	9
Read Date	9
Write Date	10
Read Back Date	10
Functional Description	11
Functions	11
Initialization/Termination	11
ValidateLicense	11
Open	11
Close	11
GetConfiguration	12
SetConfiguration	12
Operation	12
Connect	12
Disconnect	12
SendRequest	13
GetConnectionStatus	13
GetComPortStatus	13
Structures	14
T_THB_strConfiguration	14
T_THB_strConnection	14
Constants	15

Overview

The Hart Driver DLL is implementing the Hart communication protocol like the already existing HartDLL 6.0 of Borst Automation. The DLL is not (!) using any framework like MFC. It does not use the Windows Registry and is not depending on any other DLL except the standard Windows system DLLs. The DLL itself is using standard Windows API calls and is therefore compatible to all Versions of Windows with the 32 Bit API.

The implementation of the Hart Protocol does not contain any restriction to frame lengths in Hart 5.x (e.g.). Therefore the communication functions can be used for devices supporting Hart 5.x up to Hart 6 and Hart 7 as it was recently released.

The usage of the driver requires a certain amount of steps. Before using the communication the application has to register for a com port of the PC. This can be any com port from 1 to 255 including virtual com ports as they are used for USB like the hart modems of MACTek[®] or Microflex.

Typical Service Processing Program Flow

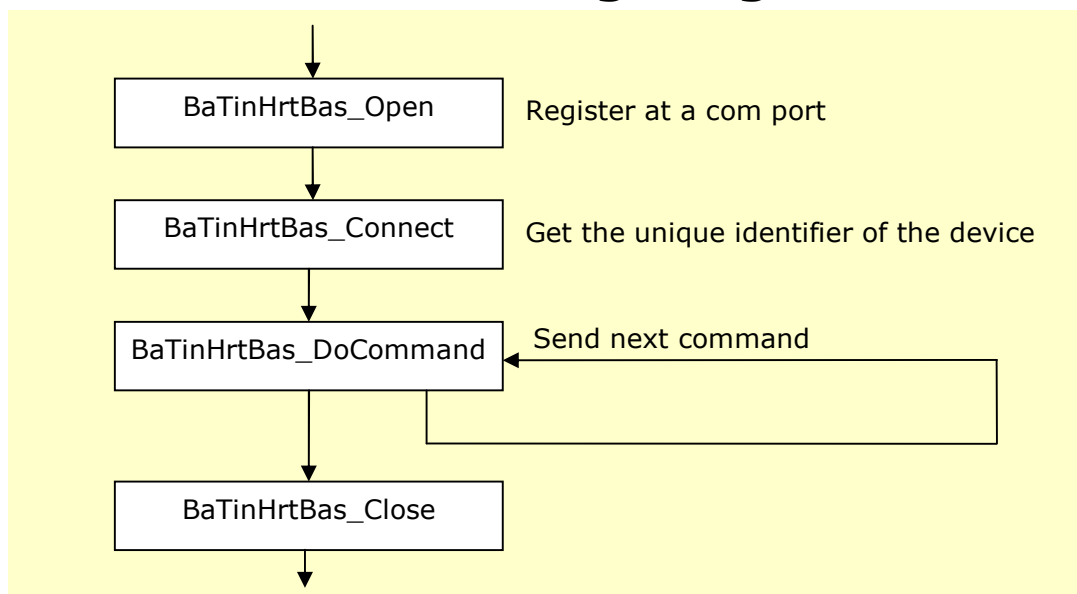


Figure 1: Typical Application Program Flow Structure

When a service is processed using the function BaTinHrtDas_DoCommand the program is returning when the service is totally completed even if there are errors or if the device is not responding.

Note: If a device is not responding, the function delay for a multiple of the number of retries which had been configured by the function BaTinHrtBas_SetConfiguration.

Architecture

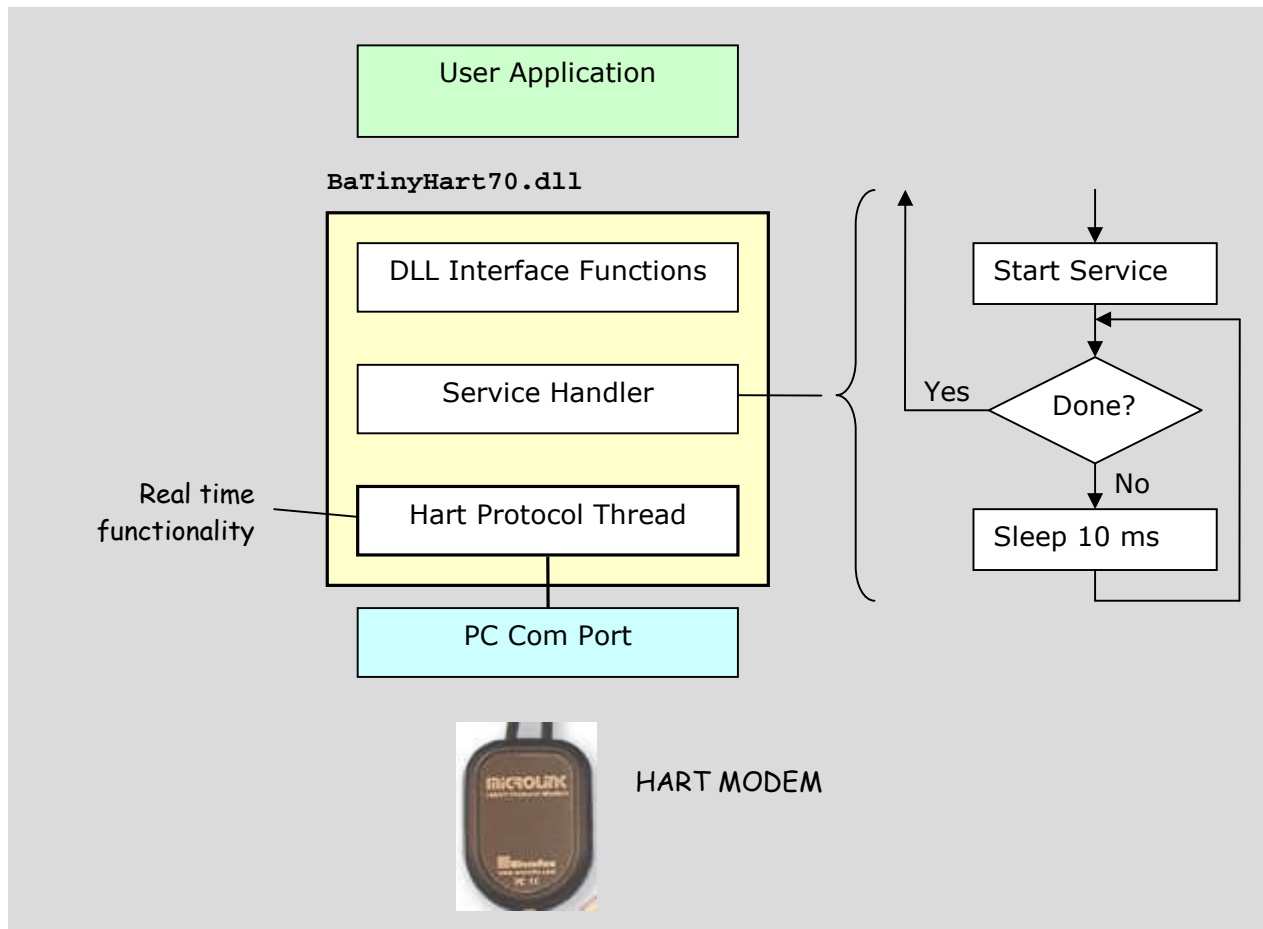


Figure 2: The Internal Structure of the DLL

The figure above shows that the DLL is using its own thread for the real time application. Thus the calling thread may be of any kind. Even if the DLL is waiting for the completion of the service it is taking the calling thread into sleep mode.

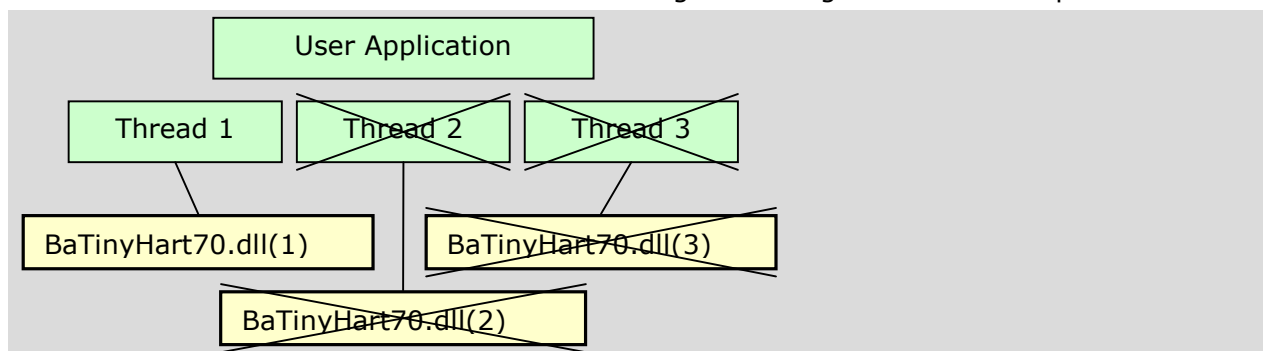


Figure 3: The DLL cannot be used by different Threads

The DLL is only running within the calling thread thus not blocking others. All functions of the DLL are thread safe. However the tiny version does not support more than one instance of the driver DLL.

Directory Structure

After unzipping the Borst Automation Package the following directory structure was generated in the application path.

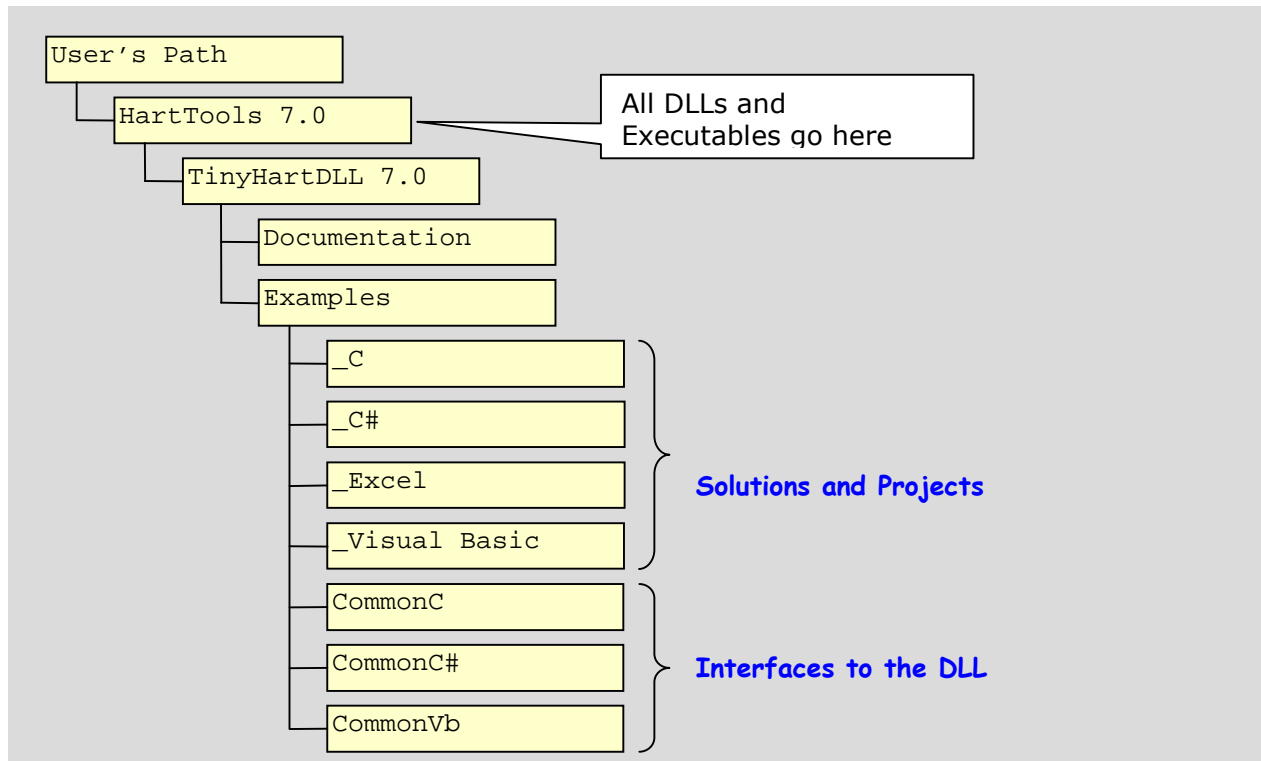


Figure 4: Directory Structure after Unzipping

Getting Started

Unzip the package into a directory of your choice. Open one of the solutions provided in the example pathes.

Note: The projects were generated with Visual Studio 2005. Trying the examples with an earlier Version of Visual Studio will not work.

In the directory `_Excel` you find an xls file containing VBA code for accessing the Tiny Hart DLL.

It is recommended to provide a copy of `BaTinyHart70.dll` on the windows system path to allow the access of the DLL from any directory.

Distribution of Applications

The only thing you have to provide with your application is a copy of the DLL (BaTinyHart70.dll). The DLL has to be stored in the directory the application is starting from or in the Windows system path.

Note: Be sure that the first call of your application is a call of the validation function of the DLL (BaTinHrtBas_ValidateLicense) passing a valid license code to the DLL.

Examples

There are four examples provided to demonstrate the access of the DLL in C/C++, C#, Visual Basic and VBA¹.

All examples have the same functionality. Firstly a com port is opened and a connection is established. Then the hart command 13 is processed and day, month and year are decoded from the frame (response data). In all examples it is provided to write a new date to the device.

C/C++

The solution is placed in the path

```
.\HartTools 7.0\TinyHartDLL 7.0\Examples\_C\BaTinyHartBasExmpl.sln
```

The example is a little console application written in straight C.

```
C:\d:\_HartX.Net.V70beta\02_Test\01_DebugBench\BaTinyHartBas...
Enter Com Port: 2
PrimaryMaster <y/n>? y
Connecting...
Connected!!!
Manufacturer ID: 251
Device ID: 1
Reading New Date..
Response Code 1: 0
Response Code 2: 01000000
Response Data Len: 21
Day: 5
Month: 3
Year: 2010
Write new Date <y/n>?
```

The interface is defined in BaTinyHart70.h which is including another general header file (BaHartDrvGen70.h).

The first call of the DLL is used for setting the license key.

```
/* Set the License in the DLL */
BaTinHrtBas_ValidateLicense("12345678-ABCD-9012-DDDD-1234567TRIAL");
```

The license code here has to be replaced by your license code which is provided to you after purchasing the full version.

¹ VBA = Visual Basic for Applications (used in Excel)

C#

The solution is placed in the path

```
.\HartTools 7.0\TinyHartDLL 7.0\Examples\C#\BaTinyHartBasExmpl_In_Csharp.sln
```

The example is based on a dialog form.

The interface is declared in the module BaTinyHart70_Iface.cs which located in the directory CommonC#.

A few amount of private data is used in the form handling procedures.

```
#region Private Data
byte                byLastError = 255;
byte                byRsp1 = 255;
byte                byRsp2 = 255;
TinyHartBasDLL.T_THB_strConnection strConnection = new TinyHartBasDLL.T_THB_strConnection();
byte                byMyStatus = MYSTAT_IDLE;
byte[]              abyRequestData = new byte[255];
byte[]              abyResponseData = new byte[255];
byte                byRspLen = 0;
#endregion
```

The first call into the DLL is implemented in the load event of the main form (frmMain).

```
TinyHartBasDLL.BaTinHrtBas_ValidateLicense
(new StringBuilder("12345678-ABCD-9012-DDDD-1234567TRIAL"));
```

The StringBuilder class which is used here is referenced in the namespace `System.Text`.

Opening the com port is performed in the SelectedIndexChanged event of then control cmbComPort.

```

if(TinyHartBasDLL.BaTinHrtBas_GetComPortStatus() == TinyHartBasDLL.CPS_OK)
{
    TinyHartBasDLL.BaTinHrtBas_Close();
}
if(cmbComPort.SelectedIndex != 0)
{
    if(TinyHartBasDLL.BaTinHrtBas_Open
        (Convert.ToInt16(cmbComPort.SelectedIndex)) == TinyHartBasDLL.CPS_OK)
    {
        byMyStatus = MYSTAT_HAVING_PORT;
    }
    else
    {
        byMyStatus = MYSTAT_IDLE;
    }
}
}

```

If another com port is already opened it is closed. If the the selected index of cmbComPort is different from 0 (which means no com port) it is opened. If opening the com port was successful the internal state is set to MYSTAT_HAVING_PORT.

In the hart protocol almost all commands are performed by using the so called unique identifier of the device as addressing information. The unique identifier can be read from the HART device by command 0. This is done when a connection is established.

Before the connection is established the driver is configured with some setting in the procedure ConfigureDriver().

```

private void ConfigureDriver()
{
    TinyHartBasDLL.T_THB_strConfiguration strConfiguration =
        new TinyHartBasDLL.T_THB_strConfiguration();

    TinyHartBasDLL.BaTinHrtBas_GetConfiguration(ref strConfiguration);
    strConfiguration.byNumRetries = Convert.ToByte(cmbNumRetries.SelectedIndex);
    strConfiguration.byMasterRole = Convert.ToByte(cmbMasterRole.SelectedIndex);
    strConfiguration.byNumPreambles = Convert.ToByte(cmbNumPreambles.SelectedIndex + 2);
    strConfiguration.byRetryIfBusy = Convert.ToByte(cmbRetryIfBusy.SelectedIndex);
    TinyHartBasDLL.BaTinHrtBas_SetConfiguration(ref strConfiguration);
}

```

There is very few code for establishing the connection in the Click event of the connect button.

```

byLastError = TinyHartBasDLL.BaTinHrtBas_Connect(Convert.ToByte(cmbAddress.SelectedIndex),
                                                ref strConnection);
DisplayCommResults();
if(byLastError == TinyHartBasDLL.ERR_OK)
{
    byMyStatus = MYSTAT_CONNECTED;
}
else
{
    byMyStatus = MYSTAT_HAVING_PORT;
}
}

```

After successfully connecting the structure strConnection contains some information about the connected device.

When reading data (as well as establishing a connection) the service is completed when the call to the DLL is returning. Therefore no polling or any other means are required. The code for reading tag, descriptor and date is integrated in the click event of the read button.

```
byLastError = TinyHartBasDLL.BaTinHrtBas_SendRequest(
    13,
    0,
    ref abyRequestData[0],
    ref byRspLen,
    ref abyResponseData[0],
    ref byRsp1,
    ref byRsp2);

if(byLastError == TinyHartBasDLL.ERR_OK)
{
    txtDay.Text = abyResponseData[18].ToString();
    txtMonth.Text = abyResponseData[19].ToString();
    ushort us = (ushort)(abyResponseData[20] + 1900);
    txtYear.Text = us.ToString();
    byMyStatus = MYSTAT_HAVING_DATA;
}
else
{
    byMyStatus = MYSTAT_CONNECTED;
}
```

After the communication was successful day, month and year are picked from the response.

For writing new values to day, month and year command 18 is used. Before inserting the data into the request frame the response data is copied to the request data byte array.

```
Array.Copy(abyResponseData, abyRequestData, byRspLen);
```

Then the new data is inserted into the request stream.

```
abyRequestData[18] = Convert.ToByte(cmbDay.SelectedIndex + 1);
abyRequestData[19] = Convert.ToByte(cmbMonth.SelectedIndex + 1);
if(cmbYear.SelectedIndex == 0)
{
    abyRequestData[20] = 0;
}
else if(cmbYear.SelectedIndex == 11)
{
    abyRequestData[20] = 255;
}
else
{
    abyRequestData[20] = Convert.ToByte(109 + cmbYear.SelectedIndex);
}
```

Finally command 18 together with the new data is sent to the device.

```
byLastError = TinyHartBasDLL.BaTinHrtBas_SendRequest(
    18,
    21,
    ref abyRequestData[0],
    ref byRspLen,
    ref abyResponseData[0],
    ref byRsp1,
    ref byRsp2);
```

Visual Basic

The example in Visual Basic is exactly the same as for C#. For functional details please refer to the C# example.

The interface is declared in the module BaTinyHart70_Iface.vb.

For using the StringBuilder object in the following statement

```
TinyHartBasDLL.BaTinHrtBas_ValidateLicense _
    (new StringBuilder("12345678-ABCD-9012-DDDD-1234567TRIAL"))
```

it is required to import the namespace System.Text.

```
Imports System.Text
```

Excel

As the codes for C# and Visual Basic.Net are very similar using the same virtual machine VBA² is completely different.

The worksheet which is used is very simple and straight forward.

	A	B	C	D	E
1	Test	Day	Month	Year	ComPort
2	Test				2
3	Old				
4	New	5	3	2010	
5	Stored				
6					

Getting Started

Double click the file TestTinyHartDLL.xls in the example path _Excel. Excel opens and appears with a button in the upper left corner. Enter the com port number of the port you have connected a device to. Press the button and the Visual Basic Editor will appear because the program was stopped at a 'breakpoint'.

```
'Connect to device with address 0
byErr = BaTinHrtBas_Connect(0, strConnection)
If byErr = ERR_OK _
Then
    'Read tag descriptor date
    byTest = BaTinHrtBas_SendRequest(13, 0, byReqData(0), byRspDataLen, byRspData(0), _
        byRspCode1, byRspCode2)
    Range("B3") = Format(byRspData(18), "0")
    Range("C3") = Format(byRspData(19), "0")
    Range("D3") = Format(byRspData(20) + 1900, "0")
    Stop
```

² VBA = Visual Basic for Applications

Coding Details



While the module HartTest is containing the little test program the module HartInterface contains the necessary structures and functions declarations. The

following is an example of the declaration of one of the functions in the DLL.

```

Public Declare Function BaTinHrtBas_SendRequest Lib "BaTinyHart70.dll" _
  (ByVal byCmd As Byte, _
  ByVal byReqDataLen As Byte, _
  ByRef pbyReqData As Byte, _
  ByRef pbyRspDataLen As Byte, _
  ByRef pbyRspData As Byte, _
  ByRef pbyRspCode1 As Byte, _
  ByRef pbyRspCode2 As Byte _
  ) As Byte
  
```

Adjust Working Directory

The first sequence is used to adjust the current directory to the path with the DLL.

```

'Set working directory relative to xls path to
'allocate the directory with the DLL
sDir = ActiveWorkbook.Path
sDrive = Left$(sDir, 2)
ChDrive sDrive
ChDir sDir
ChDir "..\..\..\\"
  
```

Set License Key

Then the license key is registered.

```

'Set the license information
byErr = BaTinHrtBas_ValidateLicense(ByVal "12345678-ABCD-9012-DDDD-1234567TRIAL")
  
```

Open Port

For opening the com port the number of the port is taken from excel sheet.

```

'Open Com from Cell E2
'Configuration will be default
iComPort = Range("E2")
byErr = BaTinHrtBas_Open(iComPort)
  
```

Connect

The connection is established to allow the DLL to store the unique identifier of the device.

```

'Connect to device with address 0
byErr = BaTinHrtBas_Connect(0, strConnection)
  
```

Read Date

If the connectiob was O.K. command 13 is send to get tag, descriptor and date.

```

'Read tag descriptor date
byTest = BaTinHrtBas_SendRequest(13, 0, byReqData(0), byRspDataLen, byRspData(0), _
  byRspCode1, byRspCode2)
Range("B3") = Format(byRspData(18), "0")
Range("C3") = Format(byRspData(19), "0")
Range("D3") = Format(byRspData(20) + 1900, "0")
Stop
  
```

The data from the response frame is stored in cells B3, C3 and D3 of the worksheet.

Write Date

Prepare Request Frame

Firstly the response data is copied to the byte stream for the request.

```
'Copy the response to the request
For e = 0 To 17
  byReqData(e) = byRspData(e)
Next e
```

Then day, month and year are picked from the excel worksheet and are inserted into the request data byte array.

```
'Set day
byReqData(18) = Range("B4")
'Set month
byReqData(19) = Range("C4")
'Set Year
byReqData(20) = Range("D4") - 1900
```

Send Command 18

Finally command 18 is sent.

```
'Send command 18
byTest = BaTinHrtBas_SendRequest(18, 21, byReqData(0), byRspDataLen, byRspData(0), _
  byRspCode1, byRspCode2)
```

Read Back Date

As a last sequence the date is read back and stored in the cells B5, C5 and D5.

```
'Read back tag descriptor date
byTest = BaTinHrtBas_SendRequest(13, 0, byReqData(0), byRspDataLen, byRspData(0), _
  byRspCode1, byRspCode2)
Range("B5") = Format(byRspData(18), "0")
Range("C5") = Format(byRspData(19), "0")
Range("D5") = Format(byRspData(20) + 1900, "0")
```

Functional Description

Functions

All functions of the DLL are thread safe. The interface for the functions calls is the same as the WINAPI functions. Thus the DLL may be used by all applications which support calls to the WINAPI functions.

Initialization/Termination

ValidateLicense

<code>unsigned char BaTinHrtBas_ValidateLicense(const char * pcLicenseCode);</code>		
BaTinyHart70.h, BaTinyHart70.lib		
Parameter	Type	Description
pcLicenseCode	In	Pointer to a text containing a valid license code.
Returns		
0: no error (license was accepted) 1..255: error		

The first call into the DLL should be a call to this function passing the correct license key to the software.

Open

<code>T_UCHR BaTinHrtBas_Open(unsigned char ucComPort);</code>		
BaTinyHart70.h, BaTinyHart70.lib		
Parameter	Type	Description
ucComPort	In	Number of the PC com port (1..254)
Returns		
0: Driver not active, com port not valid 1: Driver active		

Close

<code>void BaTinHrtBas_Close(void);</code>		
BaTinyHart70.h, BaTinyHart70.lib		

Note: Close has to be called when the application is terminating to insure that taken resources are freed.

GetConfiguration

```
void BaTinHrtBas_GetConfiguration
    (T_THB_strConfiguration * pstrConfiguration);
```

BaTinyHart70.h, BaTinyHart70.lib

Parameter	Type	Description
pstrConfiguration	Out	The function assigns the configuration to the structure pointed to.

SetConfiguration

```
void BaTinHrtBas_SetConfiguration
    (T_THB_strConfiguration * pstrConfiguration);
```

BaTinyHart70.h, BaTinyHart70.lib

Parameter	Type	Description
pstrConfiguration	In	The function uses the value of the given structure to configure the driver. In case of a parameter error the function defaults to the minimum respectively maximum value of the parameter.

If the function is not called after loading the DLL the driver is using the default values.

Operation

Connect

```
unsigned char BaTinHrtBas_Connect(unsigned char ucAddr,
    T_THB_strConnection * pstrConnection);
```

BaTinyHart70.h, BaTinyHart70.lib

Parameter	Type	Description
ucAddr	In	Hart device address (0..15)
pstrConnection	Out	The function assigns the connection details to the structure pointed to.

Returns

0: no error
 1: driver not active
 2: resource error (e.g. no free service)
 3: communication error
 4: no device response
 5..255: any other error

Disconnect

```
void BaTinHrtBas_Disconnect(void);
```

BaTinyHart70.h, BaTinyHart70.lib

The function may not be used. If the Connect function is called while a connection is existing the existing connection is overwritten by the new one.

However, if SendRequest is called after a call of Disconnect SendRequest will return an error.

SendRequest

```
unsigned char BaTinHrtBas_SendRequest(unsigned char    ucCommand,
                                       unsigned char    ucReqDataLen,
                                       unsigned char *   pucReqData,
                                       unsigned char *   pucRspDataLen,
                                       unsigned char *   pucRspData,
                                       unsigned char *   pucRspCode1,
                                       unsigned char *   pucRspCode2);
```

BaTinyHart70.h, BaTinyHart70.lib

Parameter	Type	Description
ucCommand	In	The command to be sent with the request
ucReqDataLen	In	The length of the request data
pucReqData	In	The data to be sent with the request
pucRspDataLen	In/Out	Pointer to the length of the data in the response As an in-parameter it specifies the maximum length of the buffer in the application. As out parameter it returns the actual length of the response data.
pucRspData	Out	Pointer to the response data
pucRspCode1	Out	Pointer to the response code 1
pucRspCode2	Out	Pointer to the response code 2

Returns

0: no error
 1: driver not active
 2: resource error (e.g. no free service)
 3: communication error
 4: no device response
 5: not connected to device
 6..254: any other error
 255: Trial version, number of services exceeded

The function is used to send a hart command with or without data. If the length of request data is set to 0 no data will be sent in the request.

If any null-pointer is passed the particular item will not be handled.

If a null-pointer is passed as pucReqData, the request data length is assumed to be zero and no data is sent with the request.

GetConnectionStatus

```
unsigned char BaTinHrtBas_GetConnectionStatus(void);
```

BaTinyHart70.h, BaTinyHart70.lib

Returns

0: connected
 1: not connected

GetComPortStatus

```
unsigned char BaTinHrtBas_GetComPortStatus(void);
```

BaTinyHart70.h, BaTinyHart70.lib

Returns

0: O.K.
 1: not valid

Structures

T_THB_strConfiguration

Type	Name	Description
unsigned char	ucNumPreambles	Number of preambles used for a request (5..20) Default: 5
unsigned char	ucNumRetries	Number of retries if device response is erroneous (0..3) Default: 2
unsigned char	ucRetryIfBusy	0: Do not retry if device is responding with busy code
		1..255: Retry the command if device is responding with busy code. The number of retries is reflected in the confirmation as ucUsedRetries.
		Default: 1
unsigned char	ucMasterRole	0: Primary master 1: Secondary master default = 0
unsigned short	usAddTimeOut	Additional time out to wait for a slave response in ms. Typical 100, 200 etc. Default: 0
unsigned short	usAddGapTime	Additional time for gap between characters in ms. Typical 5, 10 etc. Default: 0
unsigned short	usAddRtsOffDelay	Additional delay before Rts is switched off (carrier off) in ms. Typical 1, 2, 5, 10 etc. Default: 0

T_THB_strConnection

Type	Name	Description
unsigned char	ucManId	Manufacturer id as defined by the Hart Communication Foundation
unsigned char	ucDevId	Vendor's device id
unsigned char	ucNumPreambs	Number of preambles defined by the device
unsigned char	ucCmdRevNum	Command set revision number as defined by Hart
unsigned char	ucSpecRevCode	Device specific revision code
unsigned char	ucSwRev	Software revision code (0..255)
unsigned char	ucHwRev	Hardware revision code
unsigned char	ucHartFlags	The flags as defined by Hart
unsigned char	ucRespCode1	Response code 1 as defined by the Hart specification
unsigned char	ucRespCode2	Response code 2 as defined by the Hart specification

Constants

Name	Value	Description
Error Codes		
ERR_OK	0x00	No Error
ERR_INVALID_HANDLE	0x01	The driver is not active (due to a missing com port e.g.)
ERR_NO_RESOURCE	0x02	It was not possible to launch a service
ERR_COMM_ERROR	0x03	There was a communication error in the Hart protocol
ERR_NO_DEV_RESPONSE	0x04	There was no response from the device received
ERR_NOT_CONNECTED	0x05	The device addressing details are not yet known
ERR_TRIAL_VERSION	0xFF	Trial version, number of allowed services exceeded
Connection States		
CST_CONNECTED	0x00	The driver is ready to communicate any command
CST_NOT_CONNECTED	0x01	The device addressing details are not yet known
Com Port States		
CPS_OK	0x00	Com port was successfully reserved
CPS_NOT_VALID	0x01	Com port could not be opened
License States		
LICENSE_VALID	0x00	License code was accepted
LICENSE_NOT_VALID	0x01	License code was not accepted