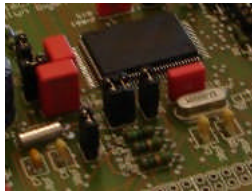


Hart Tools

HartDLL 7.0

Software Documentation

Revision: 1
Date: 28.4.2010



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	3
Directory Structure	4
Getting Started	4
Distribution of Applications	5
Examples	6
HartDLL	6
C/C++	6
C#	7
RdWrRangeAndTag	7
ConnectAndRead	8
GetCyclicData	9
GetUnIDbyTag	11
RunAsSlave	12
Visual Basic.....	12
Excel	12
Functional Description	13
Functions	13
Initialization/Termination	13
ValidateLicense	13
OpenChannel	13
CloseChannel	14
GetConfiguration	14
SetConfiguration	14
Master Services	15
Connect by Address	15
Connect by Unique ID	15
Connect by Tag Name	16
FetchConnection	16
Do Command	17
Is Service Completed	17
Fetch Confirmation	18
Encoding	18
Int8	18
Int16	18
Int24	19
Int32	19
Float	19
Packed ASCII	20
Octet(Bytes)	21
String	21
Decoding	22
Int8	22
Int16	22
Int24	22
Int32	23
Float	23
Packed ASCII	24
Octet(Bytes)	24
String	24

Slave Emulation	25
Slave Mode	25
Slave Configuration.....	26
Dynamic Variables	27
Polling for Incoming Requests	27
Delivering a Response	28
Receiving Cyclic Data	28
Burst Control	28
Asynchronous Reading	29
Using A Callback Function	29
Structures	31
Master Services	31
T_strConfiguration	31
T_strConnection	32
T_strConfirmation.....	33
Slave Emulation	33
T_strSlaveRequest.....	33
T_strSlaveResponse.....	33
T_strSlaveDynamicValues.....	34
T_strSlaveConfiguration	35
Cyclic Data.....	35
T_strCyclicData	35
Constants.....	36

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 like 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

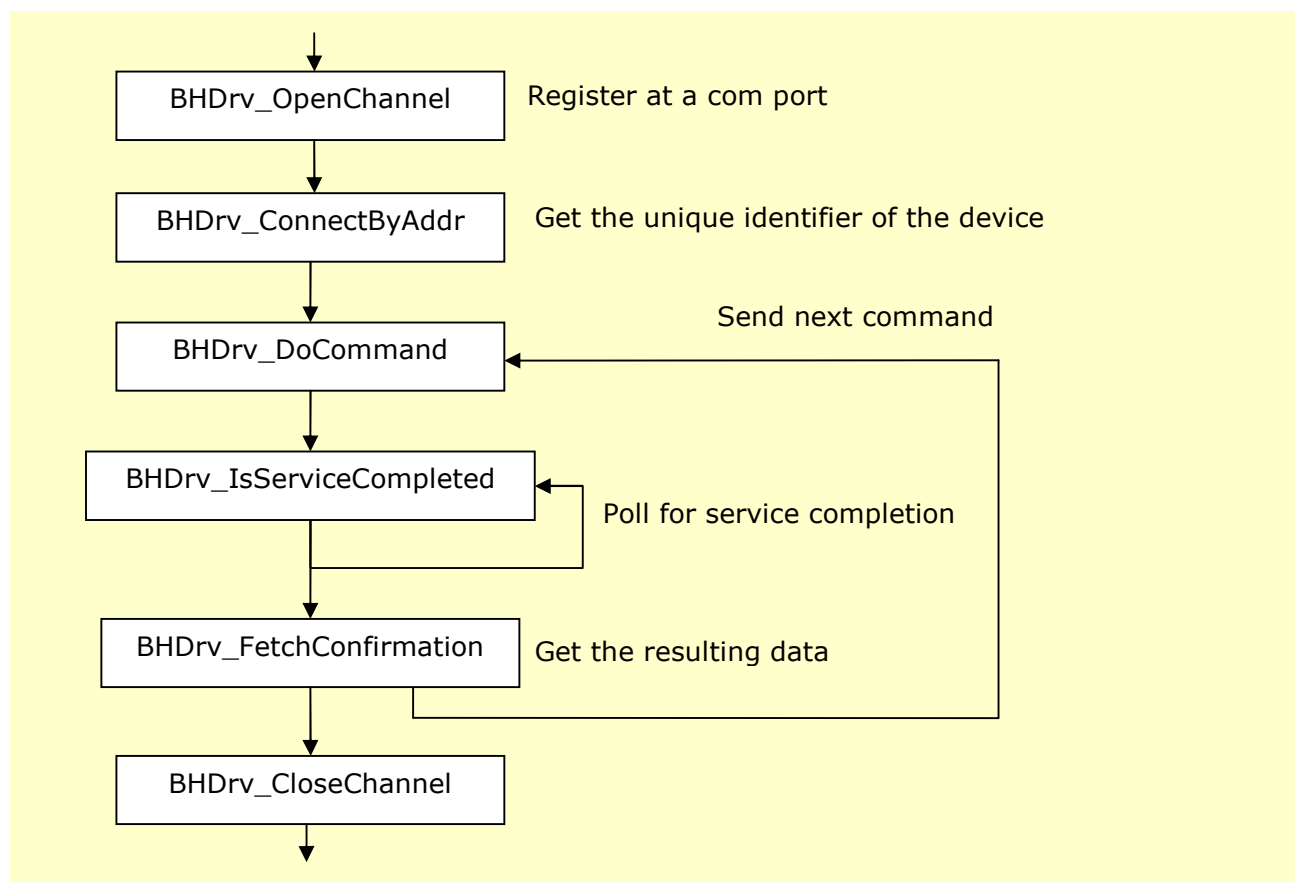


Figure 1: Polling for Service Completion

Because command 0 is the only command in Hart which is working with the short address (0..15) the unique identifier has to be fetched from the device to use it for the other commands.

There are two ways to wait for the completion of a service. Picture 1 is showing the no wait mode. In the no wait mode the client program has to poll the DLL by calling `BHDrv_IsServiceCompleted`.

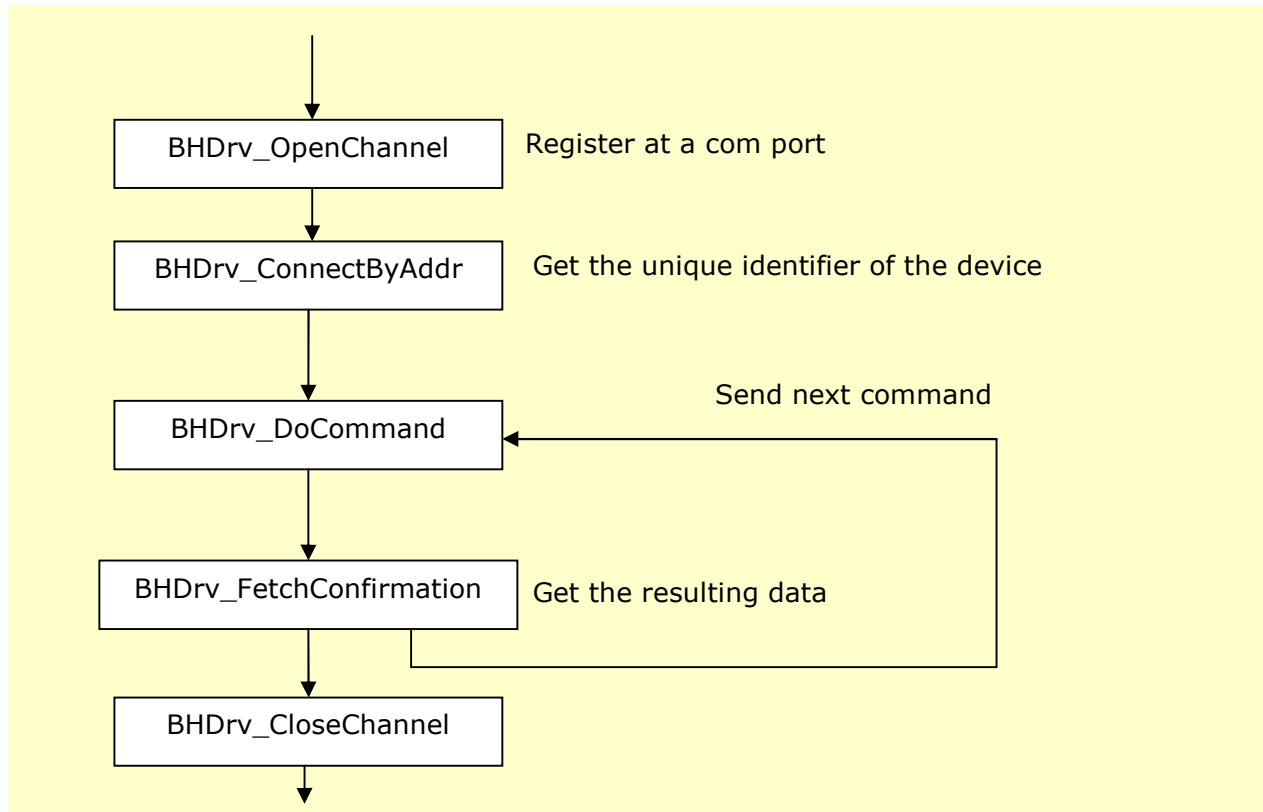


Figure 2: Using the Wait Mode of the DLL

When a service is processed using the function `BHDrv_DoCommand` with the option flag `DRV_WAIT` the program is returning when the service is totally completed even if there are errors or if the device is not responding. Waiting for a service results in a small delay of approximately 250 ms.

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 `BHDrv_SetConfiguration`.

Architecture

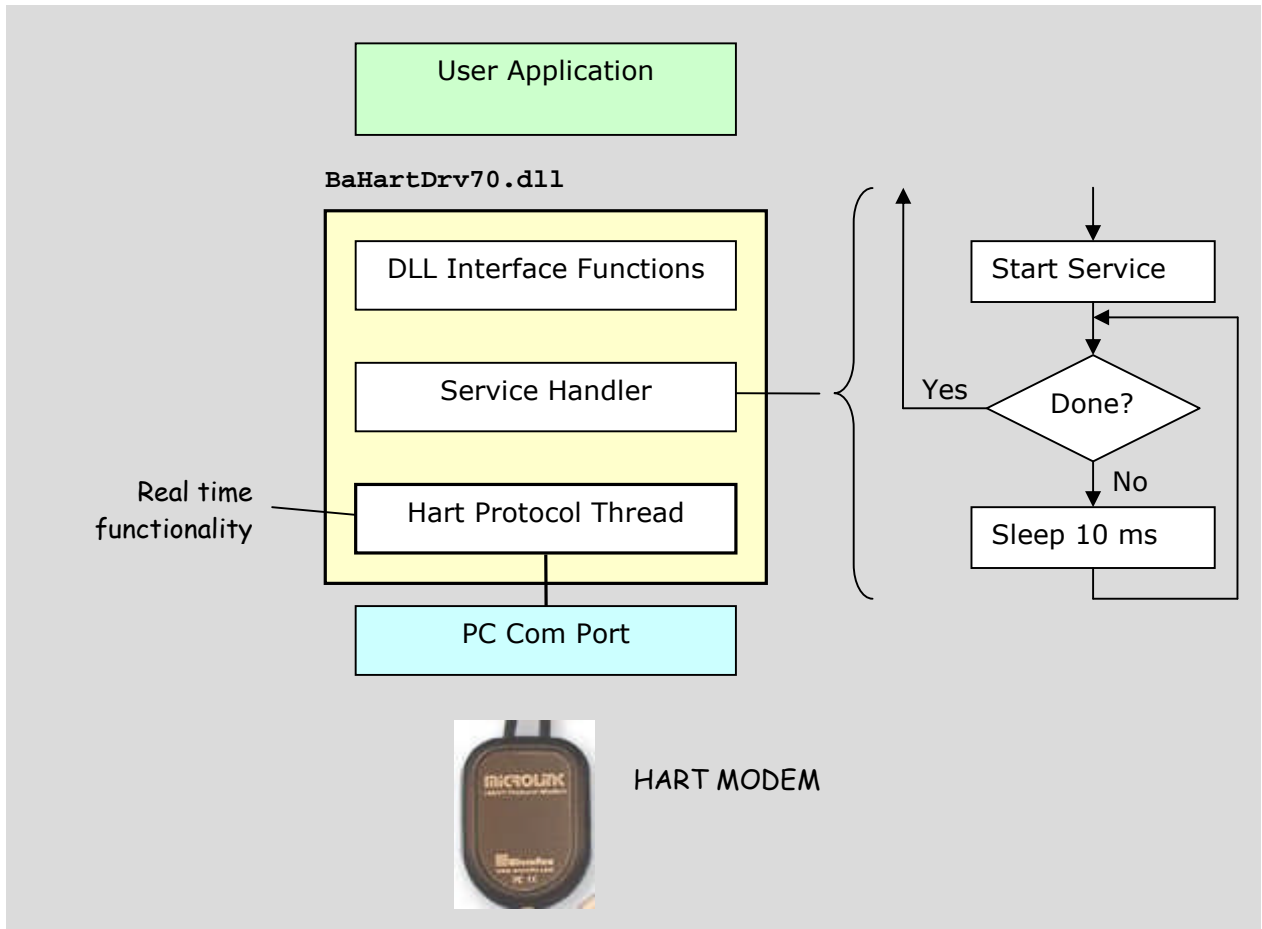


Figure 3: The Internal Structure of the DLL

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.

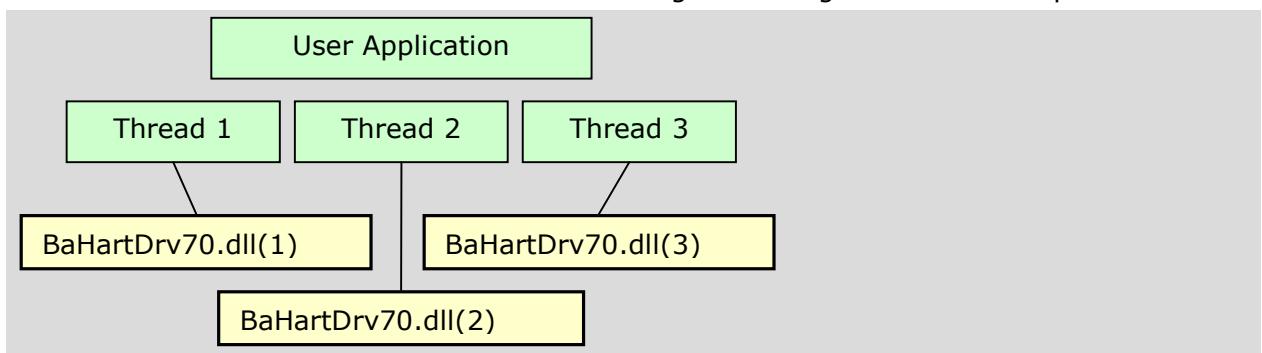


Figure 4: The DLL can be used by different Threads

The DLL may be called from several threads. The functions and communication services are thread safe. Each thread should register explicitly to get its own handle.

Directory Structure

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

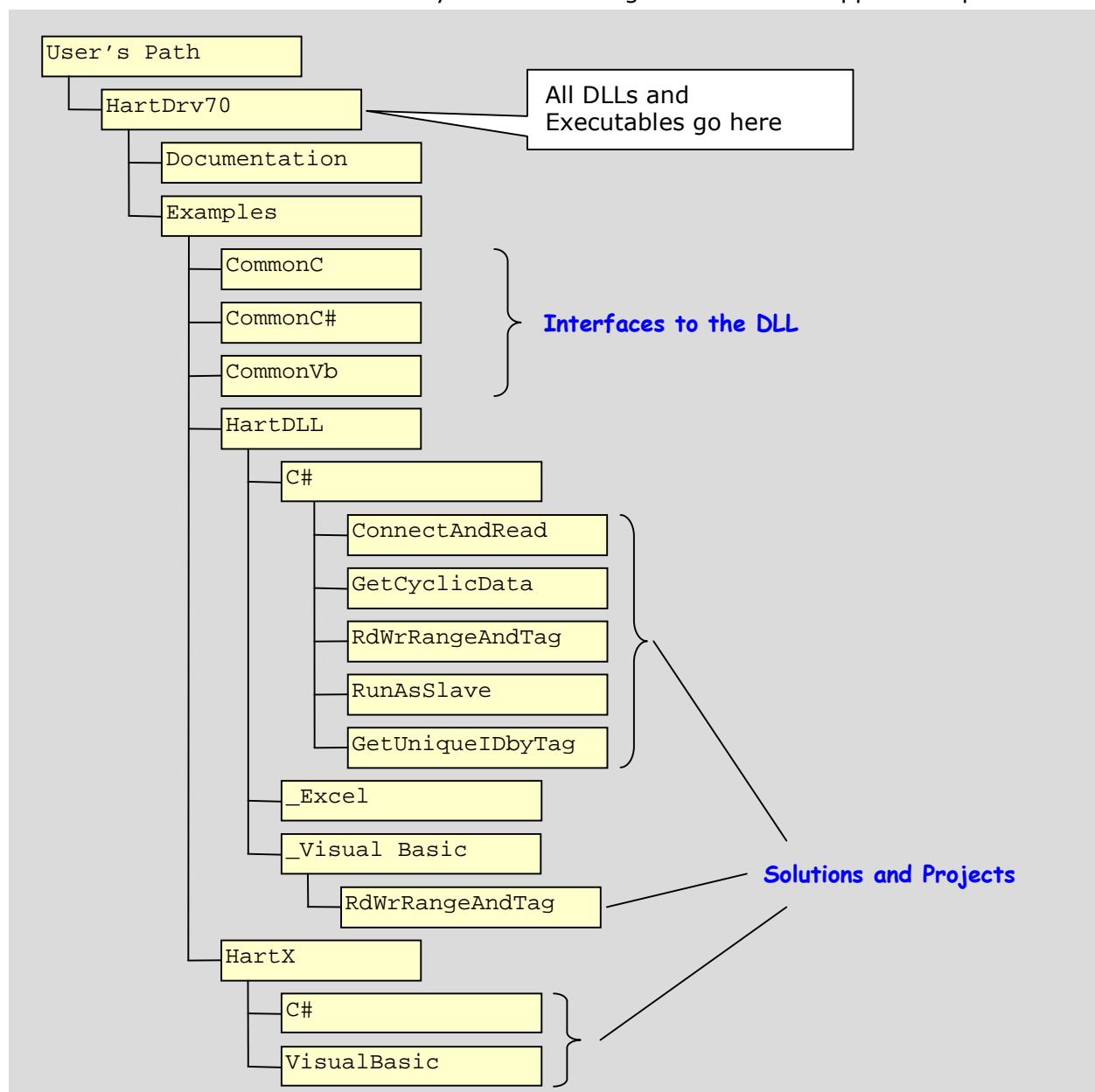


Figure 5: Directory Structure after Setup

Getting Started

Install 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 (BaHartDrv70.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 (BHDrv_ValidateLicense) passing a valid license code to the DLL.

Examples

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

All examples have a similar functionality. Firstly a com port is opened and a connection is established. Then some Hart data is accessed.

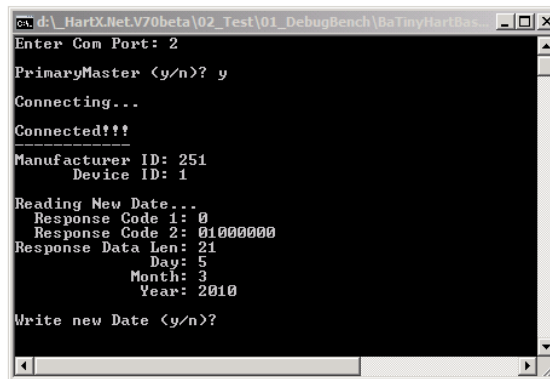
HartDLL

C/C++

The solution is placed in the path

```
.\Examples\HartDLL\Cpp\UsingBaHartDrv70.sln
```

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



```
cmd: 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 BaHartDrv70.h.

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

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

Code Snippet 1: Calling ValidateLicense

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)

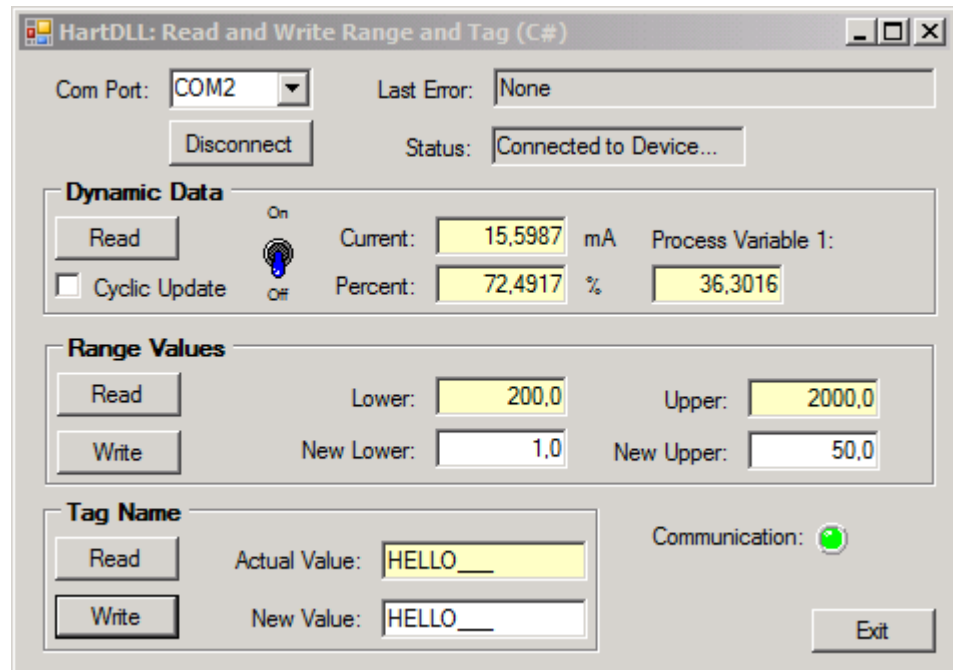
C#

RdWrRangeAndTag

The solution is placed in the path

```
.\Examples\HartDLL\C#\RdWrRangeAndTag\CsRdWrRangeAndTag.sln
```

The example is based on a dialog form.



The interface to HartDrv70 is declared in the module BaHartDrv70_Iface.cs which located in the directory CommonC#.

The example is demonstrating two features of the driver DLL. The first is thread safety the second is the way how to encode and decode data.

The first call into the DLL is implemented in the constructor of the class CCom, which is implemented in the module Communication.cs.

```
HartDLL.BHDrv_ValidateLicense(new StringBuilder("12345678-ABCD-9012-DDDD-1234567TRIAL"));
```

Code Snippet 2: Calling ValidateLicense

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

The thread safety of the DLL is demonstrated by providing a separate thread for each service thus running more than one thread at a time requesting Hart services.

It has to taken into account that for the connection of the worker threads and the application, which is a Windows form, the method BeginInvoke() is used.

Let's have a look to the code for reading cyclically refreshed variables (cyclic data). This is started either in the Click event of the button butReadPVs or the RunWorkerCompleted event of the worker thread wrkReadPVs. Both events are calling the method RunWorkerAsync of this worker thread. The event DoWork of the said thread is calling the method ReadPVsInWrkThread of the CCom class.

The thread is performing two Hart services for reading the current the percentage value and the primary variable.

```
public void ReadPVsInWrkThread(frmMain frm)
{
    float fCurrent = 0.0f;
    float fPercent = 0.0f;
    float fPv1 = 0.0f;

    if(HandleCommand(2, m_abyData, 0, 8))
    {
        fCurrent = HartDLL.BHDrv_PickFloat(0, ref m_strConfirmation.aby_Data[0], HartDLL.MSB_FIRST);
        fPercent = HartDLL.BHDrv_PickFloat(4, ref m_strConfirmation.aby_Data[0], HartDLL.MSB_FIRST);
    }
    if(HandleCommand(1, m_abyData, 0, 5))
    {
        fPv1 = HartDLL.BHDrv_PickFloat(1, ref m_strConfirmation.aby_Data[0], HartDLL.MSB_FIRST);
    }
    delPVsResult dPVsResult = new delPVsResult(frm.GetPVsResult);
    frm.BeginInvoke(dPVsResult, new Object[] {fCurrent, fPercent, fPv1});
}
```

Code Snippet 3: Reading the Dynamic Values

Because a thread must not access the controls of the main form a function provided by the parent (main form) is invoked. It is important to recognize that BeginInvoke is used. The reason is, that the BeginInvoke method is using the PostMessage function of Windows, thus not blocking the calling thread.

```
public void GetPVsResult(float fCurrent, float fPercent, float fPv1)
{
    txtCurrent.Text = fCurrent.ToString("0.0###");
    txtPercent.Text = fPercent.ToString("0.0###");
    txtPV1.Text = fPv1.ToString("0.0###");
    HandleLastError();
    if(!chkCyclic.Checked)
    {
        butReadPVs.Enabled = true;
    }
}
```

Code Snippet 4: Displaying the Dynamic Values

ConnectAndRead

The Example demonstrates the usage of the connection information and the BHDrv_IsServiceCompleted method.

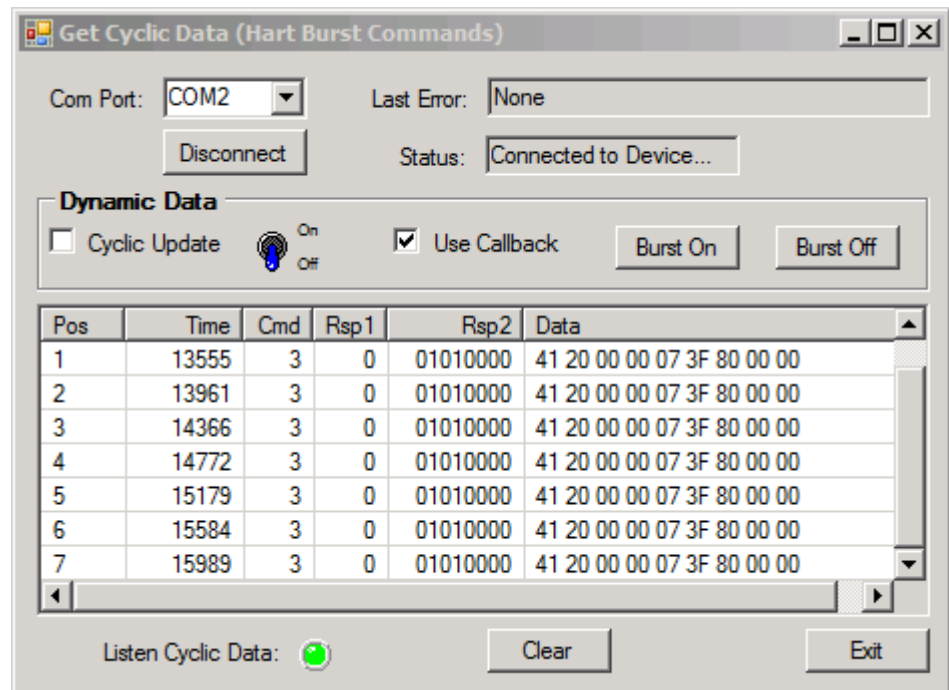
The solution is located in the following path.

```
.\Examples\HartDLL\C#\ConnectAndRead\ConnectAndRead.sln
```

GetCyclicData

The solution is stored in the subdirectory:

```
.\Examples\HartDLL\C#\GetCyclicData\CsGetCyclicData.sln
```



It is a new feature of HartDLL 7.0 to allow receiving burst messages from the Hart protocol. There are two ways to get the messages. One is to call the function `BHDrv_CycSrvGetData` to read eventual received burst frames. The other way is to register a call back which is called when a burst message arrived. As most programmers should be pretty much familiar with the polling method, the callback is not very common at least by C# users. Therefore a more detailed description is given in this section.

The callback function for fetching burst data of a device is implemented in the class `CCom` in the module `Communication.cs`.

```
private delegate void delDisplayCyclicData(HartDLL.T_strCyclicData strCyclicData);
private void SubscribeCyclicData(ref HartDLL.T_strCyclicData rstrCyclicData)
{
    m_strCyclicData = rstrCyclicData;
    try
    {
        m_frmParent.BeginInvoke(new delDisplayCyclicData(DisplayCyclicData),
                                new Object[] {m_strCyclicData});
    }
    catch
    {
    }
}
```

Code Snippet 5: The Callback Function and the Invokation Delegate

The called function takes a local copy of the data and invokes a function which is accessing the controls of the client form. The function which is invoked is `DisplayCyclicData`.

This function displays the data carried within the structure `strCyclicData`.

```
private void DisplayCyclicData(HartDLL.T_strCyclicData strCyclicData)
{
    string[] s;
    ListViewItem Item = m_lstCyclicData.Items.Add(m_lstCyclicData.Items.Count.ToString("0"));
    Item.SubItems.Add(strCyclicData.ulTimeStamp.ToString("0"));
    Item.SubItems.Add(strCyclicData.byCmd.ToString("0"));
    Item.SubItems.Add(strCyclicData.byRsp1.ToString("0"));
    Item.SubItems.Add(CHelpers.GetBin(strCyclicData.byRsp2));
    if(strCyclicData.byDataLen > 0)
    {
        s = CHelpers.simpleHexDump(strCyclicData.abby_Data, m_strCyclicData.byDataLen,200);
    }
    else
    {
        s = new string[1] {"No Data"};
    }
    Item.SubItems.Add(s[0]);
    Item.EnsureVisible();
}
```

Code Snippet 6: Displaying the Burst Data in a ListView Control

The delegate for passing the callback to the HartDLL is declared in the private data region of CCom (Communication.cs).

```
HartDLL.delSubscribeCyclicData m_dlCallback;
```

Code Snippet 7: Declaration of the Delegate for the Callback

In the constructor of CCom the function is inserted into the delegate object.

```
public CCom(frmMain frm, ListView lst)
{
    //Insert arrays into structures
    m_strConnection.abbyUniqueID = new byte[5];
    m_strConfirmation.abby_Data = new byte[64];
    m_strCyclicData.abby_Data = new byte[64];
    //Insert callback into delegate
    m_dlCallback = new HartDLL.delSubscribeCyclicData(SubscribeCyclicData);
    //Take copies of client objects
    m_lstCyclicData = lst;
    m_frmParent = frm;
}
```

Code Snippet 8: Constructor of the CCom Class

Finally the callback is registered at the HartDLL in the method `RegisterCallback` of the CCom class.

```
public void RegisterCallback()
{
    if(m_hDrv != HartDLL.INVALID_DRV_HANDLE)
    {
        HartDLL.BHDrv_CycSrvRegisterCB(m_hDrv, m_dlCallback);
    }
}
```

Code Snippet 9: Registering the Callback at the HartDLL

Precautions

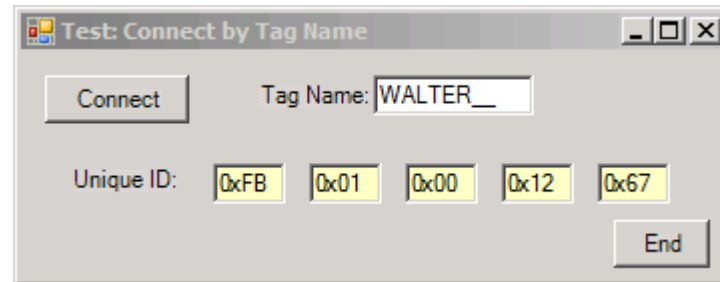
If structures are used which are containing one or more arrays these arrays must be instantiated before accessing them (see Code Snippet 8).

For not blocking the thread which is performing the callback use `BeginInvoke` for the call of the function in the parent environment (see Code Snippet 5).

GetUnIDbyTag

The solution of this example is stored at the following place.

```
.\Examples\HartDLL\C#\GetUnIDbyTag\GetUnIDbyTag.sln
```



The example demonstrates the usage of the function BHDrv_ConnectByTagName of the HartDLL.

```
private void butConnect_Click(object sender, System.EventArgs e)
{
    StringBuilder sb = new StringBuilder(" ");

    if(m_hChannel != BaHartDrv70.HartDLL.INVALID_DRV_HANDLE)
    {
        sb.Insert(0, txtTagName.Text);
        BaHartDrv70.HartDLL.BHDrv_PutPackedASCII(sb, 8, 0, ref this.abyData[0]);
        m_hService = BaHartDrv70.HartDLL.BHDrv_ConnectByTagName
            (
                m_hChannel,
                ref this.abyData[0],
                BaHartDrv70.HartDLL.DRV_WAIT,
                2
            );

        BaHartDrv70.HartDLL.BHDrv_FetchConnection(m_hService, ref m_strConnection);
        txtUnId0.Text = "0x" + m_strConnection.abyUniqueID[0].ToString("X02");
        txtUnId1.Text = "0x" + m_strConnection.abyUniqueID[1].ToString("X02");
        txtUnId2.Text = "0x" + m_strConnection.abyUniqueID[2].ToString("X02");
        txtUnId3.Text = "0x" + m_strConnection.abyUniqueID[3].ToString("X02");
        txtUnId4.Text = "0x" + m_strConnection.abyUniqueID[4].ToString("X02");
    }
}
```

Code Snippet 10: Connecting by Tag Name

Precaution

The number of the com port is hard coded in the form Load event.

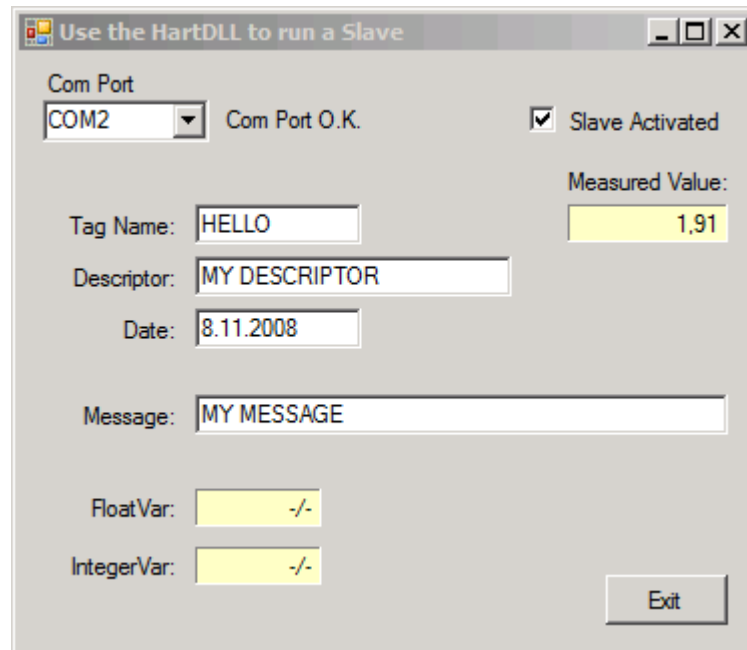
```
private void frmMain_Load(object sender, EventArgs e)
{
    m_hChannel = BaHartDrv70.HartDLL.BHDrv_OpenChannel(2);
}
```

Code Snippet 11: Setting the Com Port

RunAsSlave

The example can be found in the path:

```
.\Examples\HartDLL\C#\RunAsSlave\RunHartSlave.sln
```



The behavior of the example is explained in a html document which can be opened under the following path:

<http://borst-automation.com/manuals/RunAsSlave/CsExampleMasterSlave.htm>.

Visual Basic

There is one Visual Basic example available in the following path.

```
.\Examples\HartDLL\Visual Basic\RdWrRangeAndTag\VbRdWrRangeAndTag.sln
```

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

The interface is declared in the module BaTinyHart70_Iface.vb.

For using the StringBuilder object in the following statement

```
HartDLL.BHDrv_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. However, VBA² is completely different.

The documentation of the example is available in a separate document:

<http://borst-automation.com/manuals/UsingHartInExcel/ExcelUsingHart.pdf>.

² VBA = Visual Basic for Applications

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>void BHDrv_ValidateLicense(const char * pcLicenseCode);</code>		
BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb		
Parameter	Type	Description
pcLicenseCode	In	Pointer to a text containing a valid license code.
Usage		
<code>BHDrv_ValidateLicense("12345678-ABCD-9012-DDDD-1234567TRIAL");</code>		

The first call into the DLL should be a call to this function passing the correct license key to the software. On computers, where the Borst Automation software was installed, this is not necessary because the license was copied to the registry. However, if you plan to distribute your application it has to call this function.

OpenChannel

<code>unsigned int BHDrv_OpenChannel(unsigned short usComPort);</code>		
BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb		
Parameter	Type	Description
usComPort	In	Number of the PC com port (1..255)
Returns		
INVALID_DRV_HANDLE: Com port could not be registered Any other value: Registration successful		
Usage		
<code>m_hDrv = HartDLL.BHDrv_OpenChannel(Convert.ToInt16(m_iComPort));</code>		

The function allocates the selected com port if possible and starts its own working thread for accessing Hart services. The value which is returned is a handle which has to be passed to all functions which are requesting a service.

CloseChannel

```
void BHDrv_CloseChannel(unsigned int hDrv);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description
hDrv	In	The handle which was returned by OpenChannel

Usage

```
HartDLL.BHDrv_CloseChannel(m_hDrv);
```

It is required to call this function at least when the application is terminating.

GetConfiguration

```
void BHDrv_GetConfiguration(unsigned int hDrv,
                             T_strConfiguration * pstrConfiguration);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description
hDrv	In	The handle which was returned by OpenChannel
pstrConfiguration	Out	The function copies the configuration to the structure pointed to.

Usage

```
BHDrv_GetConfiguration(hDrv, &strConfiguration);
```

SetConfiguration

```
void BHDrv_SetConfiguration(unsigned int hDrv,
                             T_strConfiguration * pstrConfiguration);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description
hDrv	In	The handle which was returned by OpenChannel
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.

Usage

```
strConfiguration.ucInitialMasterRole = 1;
BHDrv_SetConfiguration(hDrv, &strConfiguration);
```

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

Master Services

Connect by Address

<pre>unsigned int BHDrv_ConnectByAddr(unsigned int hDrv, unsigned char ucAddr, unsigned char ucQOS, unsigned char ucNumRetries);</pre>			
BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb			
Parameter	Type	Description	
hDrv	In	The handle which was returned by the OpenChannel function	
ucAddr	In	Hart device address (0..15)	
ucQOS	In	Quality of service	
		DRV_WAIT	Return if the service is completed
		DRV_NO_WAIT	Return immediately, user will poll for completion
ucNumRetries	In	Number of retries to repeat the service in case of a communication error respectively missing response	
Returns			
INVALID_SRV_HANDLE: No resource available, service could not be started Any other value: Service handle which is used to access the service results			
Usage			
<pre>m_hService = HartDLL.BHDrv_ConnectByAddr(m_hDrv, m_byAddress, HartDLL.DRV_WAIT, 2);</pre>			

The service is using the short address to perform Hart command 0 to get the unique identifier.

Connect by Unique ID

<pre>unsigned int BHDrv_ConnectByUniqueID(unsigned int hDrv, unsigned char * pucUniqueID, unsigned char ucQOS, unsigned char ucNumRetries);</pre>			
BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb			
Parameter	Type	Description	
hDrv	In	The handle which was returned by the OpenChannel function	
pucUniqueID	In	Pointer to a five octet array with the unique identifier	
ucQOS	In	Quality of service	
		DRV_WAIT	Return if the service is completed
		DRV_NO_WAIT	Return immediately, user will poll for completion
ucNumRetries	In	Number of retries to repeat the service in case of a communication error respectively missing response	
Returns			
INVALID_SRV_HANDLE: No resource available, service could not be started Any other value: Service handle which is used to access the service results			

The function is using the unique identifier. It can be used on a multidrop network even if all devices have the same short address. However using this function makes only sense if the unique identifier of each device is known.

Connect by Tag Name

<pre>unsigned int BHDrv_ConnectByUniqueID(unsigned int hDrv, unsigned char * pucTagName, unsigned char ucQOS, unsigned char ucNumRetries);</pre>			
BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb			
Parameter	Type	Description	
hDrv	In	The handle which was returned by the OpenChannel function	
pucTagName	In	Pointer to the octet array of a length of 6 or 24 depending wether the short or long tag name is used	
ucQOS	In	Quality of service	
		DRV_WAIT	Return if the service is completed
		DRV_NO_WAIT	Return immediately, user will poll for completion
ucNumRetries	In	Number of retries to repeat the service in case of a communication error respectively missing response	
Returns			
INVALID_SRV_HANDLE: No resource available, service could not be started Any other value: Service handle which is used to access the service results			
Usage			
<pre>StringBuilder sb = new StringBuilder(" "); sb.Insert(0, txtTagName.Text); BaHartDrv70.HartDLL.BHDrv_PutPackedASCII(sb, 8, 0, ref this.abData[0]); m_hService = BaHartDrv70.HartDLL.BHDrv_ConnectByTagName(m_hChannel, ref this.abData[0], BaHartDrv70.HartDLL.DRV_WAIT, 2);</pre>			

The function uses the tag name of the device to retrieve the unique identifier. Typically this function is used in multidrop networks.

FetchConnection

<pre>void BHDrv_FetchConnection(unsigned int hDrv, T_strConnection * pstrConnection);</pre>		
BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb		
Parameter	Type	Description
hSrv	In	The service handle which was returned by the service request for connecting to a device
pstrConnection	Out	The function assigns the connection details (device information) to the structure pointed to. In case of a communication error only the error variable is set (see service completion codes).
Usage		
<pre>HartDLL.BHDrv_FetchConnection(m_hService, ref m_strConnection); if(m_strConnection.byError != HartDLL.SRV_SUCCESSFUL) { m_iLastError = ERR_CONNECTION_FAILED; } else { m_iComPortStatus = COM_CONNECTED; }</pre>		

Note: After a call of this function the driver is deleting the service. hService is no longer valid after calling FetchConnection once.

Do Command

```

unsigned int BHDrv_DoCommand(unsigned int          hDrv,
                             unsigned char        ucCommand,
                             unsigned char        ucQOS,
                             unsigned char *      pucReqData,
                             unsigned char        ucReqDataLen,
                             unsigned long        dwAppKey,
                             unsigned char *      pucUniqueID
                             );

```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description	
hDrv	In	The handle which was returned by the OpenChannel function	
ucCommand	In	Hart command to be sent with the request	
ucQOS	In	Quality of service	
		DRV_WAIT	Return if the service is completed
		DRV_NO_WAIT	Return immediately, user will poll for completion
pucReqData	In	Request data: octet stream to be sent with the command	
ucReqDataLen	In	Length of the request data stream (number of octets/bytes)	
dwAppKey	In	Any value. The value which the user is setting here is returned by the confirmation as is. This could be something like a pointer to a callback function.	
pucUniqueID	In	Unique identifier of the addressed device	

Returns

INVALID_SRV_HANDLE	No resource available, service could not be started
Any other value	Service handle which is used to access the service results

Usage

```

int hService = HartDLL.BHDrv_DoCommand(          m_hDrv,          byCmd, HartDLL.DRV_WAIT,
                                                ref m_abyData[0], byReqLen,          0,
                                                ref m_strConnection.abyUniqueID[0]);

if(hService != HartDLL.INVALID_SRV_HANDLE)
{
    HartDLL.BHDrv_FetchConfirmation(hService, ref m_strConfirmation);
    if(m_strConfirmation.byError == HartDLL.SRV_SUCCESSFUL)
    {
        etc...
    }
}

```

Is Service Completed

```

unsigned char BHDrv_IsServiceCompleted(unsigned int hSrv);

```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description
hSrv	In	The service handle which was returned by the connect request or do command function

Returns

T_TRUE(1)	Service is completed
T_FALSE(0)	Service is still processing

Usage

```

if(pCSampler.hService != HartDLL.INVALID_SRV_HANDLE)
{
    if(HartDLL.BHDrv_IsServiceCompleted(pCSampler.hService) == HartDLL.T_TRUE)
    {
        etc...
    }
}

```

The function may be used in implementations which use to initiate several services in parallel.

Fetch Confirmation

```
void BHDrv_FetchConfirmation(unsigned int          hSrv,
                             T_strConfirmation * pstrConfirmation);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description
hSrv	In	The service handle which was returned by the do command function
pstrConfirmation	Out	Points to a structure to assign the result of a service to

Usage

```
HartDLL.BHDrv_FetchConfirmation(hService, ref m_strConfirmation);
if(m_strConfirmation.byError == HartDLL.SRV_SUCCESSFUL)
{
    etc...
```

Encoding

Int8

```
void BHDrv_PutInt8(unsigned char    ucData,
                   unsigned char    ucOffset,
                   unsigned char * pucStream);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description
ucData	In	8 bit value to be inserted into the stream
ucOffset	In	Offset to the first octet (byte) of the stream
pucStream	In	Pointer to the byte stream to insert the encoded value into

Int16

```
void BHDrv_PutInt16(unsigned short   usData,
                    unsigned char    ucOffset,
                    unsigned char * pucStream,
                    unsigned char    ucEndian);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description	
usData	In	16 bit value to be inserted into the stream	
ucOffset	In	Offset to the first octet (byte) of the stream	
pucStream	In	Pointer to the byte stream to insert the encoded value into	
ucEndian	In	Byte order	
		MSB_FIRST(0)	Big Endian (Hart standard)
		LSB_FIRST(1)	Little Endian

To allow a flexible byte order is interesting for vendor specific commands, which are sometimes not following the Hart standard. MSB stand for Most Significant Byte and LSB for Least Significant Byte.

Int24

```
void BHDrv_PutInt24(unsigned long    ulData,
                   unsigned char    ucOffset,
                   unsigned char * pucStream,
                   unsigned char    ucEndian);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description	
ulData	In	24 bit value to be inserted into the stream	
ucOffset	In	Offset to the first octet (byte) of the stream	
pucStream	In	Pointer to the byte stream to insert the encoded value into	
ucEndian	In	Byte order	
		MSB_FIRST(0)	Big Endian (Hart standard)
		LSB_FIRST(1)	Little Endian

Int32

```
void BHDrv_PutInt32(unsigned long    ulData,
                   unsigned char    ucOffset,
                   unsigned char * pucStream,
                   unsigned char    ucEndian);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description	
ulData	In	32 bit value to be inserted into the stream	
ucOffset	In	Offset to the first octet (byte) of the stream	
pucStream	In	Pointer to the byte stream to insert the encoded value into	
ucEndian	In	Byte order	
		MSB_FIRST(0)	Big Endian (Hart standard)
		LSB_FIRST(1)	Little Endian

Float

```
void BHDrv_PutFloat(float    fData,
                   unsigned char    ucOffset,
                   unsigned char * pucStream,
                   unsigned char    ucEndian);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description	
fData	In	Float value (single precision) to be inserted into the stream	
ucOffset	In	Offset to the first octet (byte) of the stream	
pucStream	In	Pointer to the byte stream to insert the encoded value into	
ucEndian	In	Byte order	
		MSB_FIRST(0)	Big Endian (Hart standard)
		LSB_FIRST(1)	Little Endian

Usage

```
HartDLL.BHDrv_PutFloat(m_fNewUpperRange, 1, ref m_abyData[0], HartDLL.MSB_FIRST);
```

The following summarizes the IEEE 754 and recommends that standards are referred to for implementation.

The floating point values passed by the protocol are based on the IEEE 754 single precision floating point standard.



- S - Sign of the mantissa; 1 = negative
- E - Exponent; Biased by 127 decimal in two's complement format
- M - Mantissa; 23 least significant bits, fractional portion

The value of the floating point number described above is obtained by multiplying 2, raised to the power of the unbiased exponent, by the 24-bit mantissa. The 24-bit mantissa is composed of an assumed most significant bit of 1, a decimal point following the 1, and the 23 bits of the mantissa.

$$S1.M \cdot 2^{(E-127)}$$

The floating point parameters not used by a device will be filled with 7F A0 00 00: Not-a-Number.

Packed ASCII

```
void BHDrv_PutPackedASCII(unsigned char * pucString,
                          unsigned char ucStringLen,
                          unsigned char ucOffset,
                          unsigned char * pucStream);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description
pucString	In	Pointer to an octet stream of 8-bit visible characters
ucStringLen	In	Length of the visible string (number of characters)
ucOffset	In	Offset to the first octet (byte) of the stream
pucStream	In	Pointer to the byte stream to insert the encoded value into

Usage

```
StringBuilder sb = new StringBuilder(m_sNewTagName);
//Put the new tag name into the frame
HartDLL.BHDrv_PutPackedASCII(sb, 8, 0, ref m_abyData[0]);
```

Packed ASCII Coding

The packed ASCII Format uses 6 Bit to encode a character. Therefore 4 characters in the original string require 3 octets in the resulting data. It is recommended to provide strings always as a multiple ordinal of 4 characters

Construction of Packed-ASCII characters:

- a) Truncate Bit #6 and #7 of each ASCII character.
- b) Pack four, 6 bit-ASCII characters into three bytes.

Reconstruction of ASCII characters:

- a) Unpack the four, 6-bit ASCII characters.
- b) Place the complement of Bit #5 of each unpacked, 6-bit ASCII character into Bit #6.
- c) Set Bit #7 of each of the unpacked ASCII characters to zero.

The Packed ASCII code (hexadecimal) allows the representation of the following characters.

CHAR	CODE	CHAR	CODE	CHAR	CODE	CHAR	CODE
@	00	P	10	Space	20	0	30
A	01	Q	11	!	21	1	31
B	02	R	12	"	22	2	32
C	03	S	13	#	23	3	33
D	04	T	14	\$	24	4	34
E	05	U	15	%	25	5	35
F	06	V	16	&	26	6	36
G	07	W	17	'	27	7	37
H	08	X	18	(28	8	38
I	09	Y	19)	29	9	39
J	0A	Z	1A	*	2A	:	3A
K	0B	[1B	+	2B	;	3B
L	0C	\	1C	,	2C	<	3C
M	0D]	1D	-	2D	=	3D
N	0E	^	1E	.	2E	>	3E
O	0F	_	1F	/	2F	?	3F

Note: The implementation of the function is assuming that the packed ascii string should be an ordinal multiple of 3. If the length of the passed string is not an ordinal multiple of 4 the missing packed ascii characters are replaced by spaces.

Octet(Bytes)

```
void BHDrv_PutOctets(unsigned char * pucBytes,
                    unsigned char ucNumberOfBytes,
                    unsigned char ucOffset,
                    unsigned char * pucStream);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description
pucBytes	In	Pointer to a stream of bytes
ucNumberOfBytes	In	Length of the stream (number of bytes)
ucOffset	In	Offset to the first octet (byte) of the target stream
pucStream	In	Pointer to the byte stream to insert the encoded value into

String

```
void BHDrv_PutString(unsigned char * pucString,
                    unsigned char ucStringLength,
                    unsigned char ucOffset,
                    unsigned char * pucStream);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description
pucString	In	Pointer to an octet stream of 8-bit visible characters
ucStringLength	In	Length of the visible string (number of characters)
ucOffset	In	Offset to the first octet (byte) of the stream
pucStream	In	Pointer to the byte stream to insert the encoded value into

The function inserts the visible string as it is.

Decoding

Int8

<pre>unsigned char BHDrv_PickInt8(unsigned char ucOffset, unsigned char * pucStream);</pre>		
BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb		
Parameter	Type	Description
ucOffset	In	The handle which was returned by the OpenChannel function
pucStream	In	Offset to the first octet (byte) of the stream
Returns		
8 bit unsigned integer value		
Usage		
<pre>'Copy the response to the request For e = 0 To 17 byReqData(e) = BHDrv_PickInt8(e, ByVal strConfirmation.sData) Next e</pre>		

The example was taken from a VBA program.

Int16

<pre>unsigned int BHDrv_PickInt16(unsigned char ucOffset, unsigned char * pucStream, unsigned char ucEndian);</pre>			
BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb			
Parameter	Type	Description	
ucOffset	In	The handle which was returned by the OpenChannel function	
pucStream	In	Offset to the first octet (byte) of the stream	
ucEndian	In	Byte order	
		MSB_FIRST(0)	Big Endian (Hart standard)
		LSB_FIRST(1)	Little Endian
Returns			
16 bit unsigned integer value			

Note: In VBA the return value can only be interpreted as signed integer.

Int24

<pre>unsigned long BHDrv_PickInt24(unsigned char ucOffset, unsigned char * pucStream, unsigned char ucEndian);</pre>			
BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb			
Parameter	Type	Description	
ucOffset	In	The handle which was returned by the OpenChannel function	
pucStream	In	Offset to the first octet (byte) of the stream	
ucEndian	In	Byte order	
		MSB_FIRST(0)	Big Endian (Hart standard)
		LSB_FIRST(1)	Little Endian
Returns			
Unsigned long representing a 24 bit integer value.			

Int32

```
unsigned long BHDrv_PickInt32(unsigned char    ucOffset,
                             unsigned char * pucStream,
                             unsigned char    ucEndian);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description	
ucOffset	In	The handle which was returned by the OpenChannel function	
pucStream	In	Offset to the first octet (byte) of the stream	
ucEndian	In	Byte order	
		MSB_FIRST(0)	Big Endian (Hart standard)
		LSB_FIRST(1)	Little Endian
Returns			
32 bit unsigned integer value.			

Float

```
unsigned char BHDrv_PickFloat(unsigned char    ucOffset,
                              unsigned char * pucStream,
                              unsigned char    ucEndian);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description	
ucOffset	In	The handle which was returned by the OpenChannel function	
pucStream	In	Offset to the first octet (byte) of the stream	
ucEndian	In	Byte order	
		MSB_FIRST(0)	Big Endian (Hart standard)
		LSB_FIRST(1)	Little Endian
Returns			
Single precision floating point value.			
Usage			
fLower = HartDLL.BHDrv_PickFloat(5, ref m_strConfirmation.aby_Data[0], HartDLL.MSB_FIRST);			

Packed ASCII

```
unsigned char BHDrv_PickPackedASCII(unsigned char * pucString
                                   unsigned char ucStringLength
                                   unsigned char ucOffset,
                                   unsigned char * pucStream);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description	
pucString	Out	Pointer to an octet stream for the 8-bit visible characters	
ucStringLength	In	Length of the resulting string. It should be an ordinal multiple of 4.	
		String length	Packed ASCII length
		4	3
		8	6
		12	12
		and so on...	
ucOffset	In	Offset to the first octet (byte) of the stream	
pucStream	In	Pointer to the byte stream to get the data from	
Returns			
Single precision floating point value.			
Usage			
<pre>StringBuilder sb = new StringBuilder(8); HartDLL.BHDrv_PickPackedASCII(sb, 8, 0, ref m_strConfirmation.aby_Data[0]); sTag = sb.ToString();</pre>			

Octet(Bytes)

```
void BHDrv_PickOctets(unsigned char * pucBytes,
                      unsigned char ucNumberOfBytes,
                      unsigned char ucOffset,
                      unsigned char * pucStream);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description
pucBytes	Out	Pointer to copy the bytes to
ucNumberOfBytes	In	Length of the stream (number of bytes)
ucOffset	In	Offset to the first octet (byte) of the target stream
pucStream	In	Pointer to the byte stream to insert the encoded value into

String

```
void BHDrv_PickString(unsigned char * pucString,
                      unsigned char ucStringLength,
                      unsigned char ucOffset,
                      unsigned char * pucStream);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description
pucString	Out	Pointer to an octet stream of 8-bit visible characters to copy the string to
ucStringLength	In	Length of the visible string (number of characters)
ucOffset	In	Offset to the first octet (byte) of the stream
pucStream	In	Pointer to the byte stream to insert the encoded value into

Slave Emulation

The driver can be used to emulate a HART slave. If the slave mode is enabled the driver takes active control of its role.

In slave mode the driver is automatically responding to connection establishment commands of Hart such as command 0 and command 11. If the driver is in slave mode it passes eventual received requests to the slave control unit if an answer of the application is needed.

The control unit is waiting for an answer for about 200 ms. If no answer is given by the application the slave control unit automatically sends a busy frame to inform the master about the delay.

To be continued...

Slave Mode

If the DLL should emulate a slave the slave control unit must be activated by setting the slave mode to SLAVE_ENABLED.

Get

```
unsigned char BHDrv_Slave_GetMode(unsigned int hDrv);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description
hDrv	In	The handle which was returned by the OpenChannel function
Returns		
SLAVE_ENABLED(1)		Slave active
SLAVE_DISABLED(0)		Slave not active
Usage		
<pre>if(HartDLL.BHDrv_Slave_GetMode(hChannel) != 0) { chkSlaveActive.Checked = true; } else { chkSlaveActive.Checked = false; }</pre>		

Set

```
unsigned char BHDrv_Slave_SetMode(unsigned int hDrv,
    unsigned char ucMode);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description	
hDrv	In	The handle which was returned by the OpenChannel function	
ucMode	In	Slave mode to be set	
		SLAVE_ENABLED(1)	Activate slave emulation
		SLAVE_DISABLED(0)	Deactivate slave
Usage			
<pre>HartDLL.BHDrv_Slave_SetMode(hChannel, 1);</pre>			

Slave Configuration

In order to simulate vendor specific behavior in the protocol part the configuration can be set by the application programmer.

Note: Requests with Command 0 and command 11 are not passed to the application. They are directly responded to by the protocol layer.

Get

```
void BHDrv_Slave_GetConfiguration(unsigned int hDrv,
                                T_strSlaveConfiguration * pstrSlaveConfig);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description
hDrv	In	The handle which was returned by OpenChannel
pstrSlaveConfig	Out	The function copies the slave configuration to the structure pointed to.

Usage

```
HartDLL.BHDrv_Slave_GetConfiguration(hChannel, ref strSlaveConfiguration);
```

Set

```
void BHDrv_Slave_SetConfiguration(unsigned int hDrv,
                                T_strSlaveConfiguration * pstrSlaveConfig);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description
hDrv	In	The handle which was returned by OpenChannel
pstrSlaveConfig	In	The function uses the value of the given structure to configure the slave emulation. In case of a parameter error the function defaults to the minimum respectively maximum value of the parameter.

Usage

```
sb.Insert(0, "HELLO ");
HartDLL.BHDrv_PutString(sb, 8, 0, ref strSlaveConfiguration.aucTagName[0]);
HartDLL.BHDrv_Slave_SetConfiguration(hChannel, ref strSlaveConfiguration);
```

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

Dynamic Variables

To realize the implementation of commands 1 to 3 in a most efficient way also the requests on commands 1, 2 and 3 are not passed to the application. Therefore the values for these commands have to be provided to the slave control unit.

Get

<pre>void BHDrv_Slave_GetDynValues(unsigned int hDrv, T_strSlaveDynamicValues * pstrDynValues);</pre>		
BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb		
Parameter	Type	Description
hDrv	In	The handle which was returned by OpenChannel
pstrDynValues	Out	Pointer to a structure to copy the values to.
Usage		
HartDLL.BHDrv_Slave_GetDynValues(hChannel, ref strSlaveDynValues);		

Set

<pre>void BHDrv_Slave_SetDynValues(unsigned int hDrv, T_strSlaveDynamicValues * pstrDynValues);</pre>		
BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb		
Parameter	Type	Description
hDrv	In	The handle which was returned by OpenChannel
pstrDynValues	In	Pointer to a structure to get the values from.

Polling for Incoming Requests

<pre>unsigned char BHDrv_Slave_PollForRequests(unsigned int hDrv T_strSlaveRequest * pstrRequest);</pre>		
BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb		
Parameter	Type	Description
hDrv	In	The handle which was returned by the OpenChannel function
pstrRequest	Out	Pointer to a structure to copy the request to.
Returns		
T_TRUE(1)	Request received	
T_FALSE(0)	No request pending	
Usage		
<pre>if(HartDLL.BHDrv_Slave_GetMode(hChannel) != 0) { chkSlaveActive.Checked = true; } else { chkSlaveActive.Checked = false; }</pre>		

Delivering a Response

After a request is received a response can be delivered through the PutResponse function. There is no need to deliver the response within a certain time limit because the slave control unit automatically sends a busy signal to the requesting master after 250 ms to indicate that the request is still in progress. A master receiving a busy frame will continue to send the same command until a response other than busy is received.

```
void BHDrv_Slave_PutResponse(unsigned int hDrv,
                             T_strSlaveResponse * pstrResponse);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs, BaHartDrv70_Iface.vb

Parameter	Type	Description
hDrv	In	The handle which was returned by OpenChannel
pstrResponse	In	Pointer to the buffer to get the information which should be delivered with the response.

Receiving Cyclic Data

In Hart devices cyclic data is sometimes propagated by burst messages. These are special Hart protocol primitives which are sent without any request from a master.

There are two ways to implement the access of burst messages. One is to poll the driver for incoming data, the other way is to register a callback function.

Burst Control

Start

```
void BHDrv_CycSrvStart(unsigned int hDrv);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs

Parameter	Type	Description
hDrv	In	The handle which was returned by OpenChannel

Usage

```
if(m_hDrv != HartDLL.INVALID_DRV_HANDLE)
{
    HartDLL.BHDrv_CycSrvStart(m_hDrv);
}
```

Note: If this function is called eventual existing messages in the drivers queue are deleted, thus the reception of Hart burst messages start with an empty queue.

However, before BHDrv_CycSrcStart is called incoming burst messages are discarded.

Stop

```
void BHDrv_CycSrvStop(unsigned int hDrv);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs

Parameter	Type	Description
hDrv	In	The handle which was returned by OpenChannel

Usage

```
if(m_hDrv != HartDLL.INVALID_DRV_HANDLE)
{
    HartDLL.BHDrv_CycSrvStop(m_hDrv);
}
```

After the call of this function the reception of burst messages is halted. Messages already in the queue may be read by BHDrv_CycSrvGetData.

Asynchronous Reading

```
unsigned char BHDrv_Slave_GetMode(unsigned int hDrv);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs

Parameter	Type	Description
hDrv	In	The handle which was returned by the OpenChannel function

Returns

CYCDAT_NO_DATA(1)	No cyclic data available
CYCDAT_OK(0)	Cyclic data was passed into the structure pointed too

Usage

```
ucResult = HartDLL.BHDrv_CycSrvGetData(m_hDrv, ref m_strCyclicData);
if(ucResult == HartDLL.CYCDAT_OK)
{
    DisplayCyclicData(m_strCyclicData);
}
```

Using A Callback Function

It is also possible to use a callback function. For a detailed description of this option please refer to chapter 'GetCyclicData' in the examples section.

Register

```
void BHDrv_CycSrvRegisterCB(unsigned int hDrv,
                           void (__stdcall * pfSubscribeCyclicData)
                              (T_strCyclicData * pstrCyclicData));
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs

Parameter	Type	Description
hDrv	In	The handle which was returned by the OpenChannel function
pfSubscribeCyclicData	In	Pointer to a function to be called when a burst message was received

Usage

```
if(m_hDrv != HartDLL.INVALID_DRV_HANDLE)
{
    HartDLL.BHDrv_CycSrvRegisterCB(m_hDrv, m_d1Callback);
}
```

Unregister

```
void BHDrv_CycSrvUnregister(unsigned int hDrv);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs

Parameter	Type	Description
hDrv	In	The handle which was returned by the OpenChannel function

Usage

```
if(m_hDrv != HartDLL.INVALID_DRV_HANDLE)
{
    HartDLL.BHDrv_CycSrvUnregister(m_hDrv);
}
```

The function deletes the pointer to the callback function.

Callback Prototype

```
void __stdcall BHDrv_CycSrvRegisterCB(T_strCyclicData * pstrCyclicData);
```

BaHartDrv70.h, BaHartDrv70.lib, BaHartDrv70_Iface.cs

Parameter	Type	Description
pstrCyclicData	In	Pointer to a structure to copy the cyclic data to.

Structures

Master Services T_strConfiguration

Type	Name	Description
unsigned int	uiBaudRate	Baudrate as defined in winbase.h CBR_1200 CBR_9600 CBR_19200 CBR_38400 Default: CBR_1200
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	ucInitialMasterRole	0: Primary master 1: Secondary master Default: 0
unsigned char	ucAddressingMode	0: Long address, use short address only to get unique id 1: Tag name use tag name to get unique id 2: Short address always use short address default = 0
unsigned char	ucDoNotUseRtsDtr	0: Use handshake signals 1: Do not use handshake signals 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
unsigned char	bSendJabberOctet	0: Normal sending 1: Append 0xFF to each frame Default: 0

T_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	ucError	Service completion code	
		SRV_EMPTY(0)	Not active
		SRV_NO_DEV_RESP(1)	No device response
		SRV_COMM_ERR(2)	There was some error (too few data e.g.)
		SRV_INVALID_HANDLE(3)	Service handle is invalid
		SRV_IN_PROGRESS(4)	Service working
	SRV_SUCCESSFUL(5)	Service successfully completed	
unsigned char	ucRespCode1	Response code 1 as defined by the Hart specification	
unsigned char	ucRespCode2	Response code 2 as defined by the Hart specification	
unsigned char	ucUsedRetries	Number of retries which were used for completion	
unsigned char	bDeviceInBurstMode	0: Normal mode 1: Device is in burst mode	

T_strConfirmation

Type	Name	Description
unsigned char	ucCmd	Command which was executed
unsigned char	ucRespCode1	Response code 1 as defined by the Hart specification
unsigned char	ucRespCode2	Response code 2 as defined by the Hart specification
unsigned char	ucError	Service completion code
		SRV_EMPTY(0) Not active
		SRV_NO_DEV_RESP(1) No device response
		SRV_COMM_ERR(2) There was some error (too few data e.g.)
		SRV_INVALID_HANDLE(3) Service handle is invalid
		SRV_IN_PROGRESS(4) Service working
	SRV_SUCCESSFUL(5) Service successfully completed	
unsigned char	ucUsedRetries	Number of retries which were used for completion
unsigned char	bDeviceInBurstMode	0: Normal mode 1: Device is in burst mode
unsigned short	usDuration	Time for service execution in ms
unsigned long	dwAppKey	Is returned by the FetchConfirmation function as it was passed to the DoCommand function. This can be used by the programmer for any purpose (e.g. a pointer).
unsigned char	ucReserved1	Reserved for future use
unsigned char	ucReserved2	Reserved for future use
unsigned char	ucReserved3	Reserved for future use
unsigned char	ucLen	Number of response data bytes (octets)
unsigned char	aucData [DATA_BUF_LEN]	Response data bytes (DATA_BUF_LEN = 64)

Slave Emulation

T_strSlaveRequest

Type	Name	Description
unsigned char	ucCommand	The command which was passed with the request
unsigned char	ucMasterType	Master type
		SLV_PRIM(0) Primary master
		SLV_SCND(1) Secondary master
unsigned char	ucDataLen	Number of request data bytes
unsigned char	ucReserved	Reserved for future use
unsigned char	aucData[64]	Request data bytes

T_strSlaveResponse

Type	Name	Description
unsigned char	ucCommand	The command which was passed with the request
unsigned char	ucMasterType	Master type
		SLV_PRIM(0) Primary master
		SLV_SCND(1) Secondary master
unsigned char	ucCmdResponse	Command specific response code
unsigned char	ucDataLen	Number of request data bytes
unsigned char	aucData[64]	Response data bytes

T_strSlaveDynamicValues

Type	Name	Description	
float	fPercent	Actual percent of range	
float	fCurrent	Actual current value as ma	
unsigned char	ucUnitCodePV1	Hart unit code for PV1	
unsigned char	ucUnitCodePV2	Hart unit code for PV2	
unsigned char	ucUnitCodePV3	Hart unit code for PV3	
unsigned char	ucUnitCodePV4	Hart unit code for PV4	
float	fPV1	Value of PV1	
float	fPV2	Value of PV2	
float	fPV3	Value of PV3	
float	fPV4	Value of PV4	
unsigned char	bDeviceMalfunction	Signals device mal function	T_CLEAR(0) T_SET(1)
unsigned char	bCfgChangedPrimMaster	Configuration change flag for primary master	T_CLEAR(0) T_SET(1)
unsigned char	bCfgChangedScndMaster	Configuration change flag for primary master	T_CLEAR(0) T_SET(1)
unsigned char	bColdStartPrimMaster	Cold start flag for primary master	T_CLEAR(0) T_SET(1)
unsigned char	bColdStartScndMaster	Cold start flag for secondary master	T_CLEAR(0) T_SET(1)
unsigned char	bMoreStatusAvail	Flags more status available (see command 48)	T_CLEAR(0) T_SET(1)
unsigned char	bLoopCurrentFixed	Signals fixed current mode active	T_CLEAR(0) T_SET(1)
unsigned char	bLoopCurrentSaturated	Signals current output saturated	T_CLEAR(0) T_SET(1)
unsigned char	bNonPrimVarOutLimits	Signals none primary variable out of limits	T_CLEAR(0) T_SET(1)
unsigned char	bPrimVarOutLimits	Signals primary variable out of limits	T_CLEAR(0) T_SET(1)
unsigned char	ucReserved1	Reserved for future use	
unsigned char	ucReserved1	Reserved for future use	

T_strSlaveConfiguration

Type	Name	Description
unsigned char	ucManufacturerID	Manufacturer's identifier
unsigned char	ucDeviceID	Device identifier
unsigned char	ucNumPreambles	Number of preambles needed in a request (2..20, recommended: 2)
unsigned char	ucCmdSetRevision	Hart compatibility version (5..7, recommended: 5)
unsigned char	ucTransmSpecRev	Transmitter specific revision
unsigned char	ucSoftwareRevision	Software revision number
unsigned char	ucHardwareRevision	Hardware revision number
unsigned char	ucReserved1	Reserved for future use
unsigned char	ucDevNum1	Device number [LSB]
unsigned char	ucDevNum2	Device number [LSB+1]
unsigned char	ucDevNum3	Device number [LSB+2]
unsigned char	ucReserved2	Reserved for future use
unsigned char	aucTagName[8]	Tag name (see 3.3.2.1 Packed ASCII Coding for possible characters)
unsigned char	ucAddress	Slave polling address
unsigned char	ucNumberOfPVs	Defines the number of variables to be sent with command 3
unsigned char	ucReserved3	Reserved for future use
unsigned char	ucReserved4	Reserved for future use

Cyclic Data

T_strCyclicData

Type	Name	Description
unsigned long	ulTimeStamp	Time in ms since recording of burst messages was started
unsigned char	ucCmd	Command of the received frame
unsigned char	ucRsp1	Device response code 1
unsigned char	ucRsp2	Device response code 2
unsigned char	ucDataLen	Number of bytes in productive data
unsigned char	aucData[64]	Productive data of the burst message

Constants

Name	Value	Description
Service Completion Codes		
SRV_EMPTY	0x00	Service not active
SRV_NO_DEV_RESP	0x01	Device did not respond
SRV_COMM_ERR	0x02	There was a communication error (too few data e.g.)
SRV_INVALID_HANDLE	0x03	Service handle not valid
SRV_IN_PROGRESS	0x04	Service not yet completed
SRV_SUCCESSFUL	0x05	Service successfully completed
Handle Values		
INVALID_DRV_HANDLE	-1	Driver handle not valid
INVALID_SRV_HANDLE	-1	Service handle not valid
Endian		
MSB_FIRST	0x00	Big Endian (Hart standard): <u>M</u> ost <u>S</u> ignificant <u>B</u> yte first
LSB_FIRST	0x01	Little Endian: <u>L</u> east <u>S</u> ignificant <u>B</u> yte first
Wait Options		
DRV_NO_WAIT	0x00	User will poll for the completion of service
DRV_WAIT	0x01	The function returns if service is completed
Slave Modes		
SLAVE_DISABLED	0x00	Slave emulation is not active
SLAVE_ENABLED	0x01	Slave emulation is active
Cyclic Data Handling		
CYCDAT_OK	0x00	Cyclic data available
CYCDAT_NO_DATA	0x01	Cyclic data not (yet) available