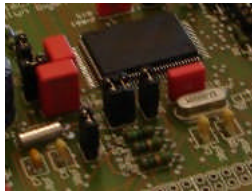


## Hart Tools

# TinyHartCPrDLL 7.0

## Software Documentation

Revision: 1  
Date: 18.5.2010



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](mailto: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

<b>Overview</b>	<b>1</b>
<b>Typical Service Processing Program Flow</b> .....	<b>1</b>
<b>Architecture</b> .....	<b>2</b>
<b>Directory Structure</b> .....	<b>3</b>
<b>Getting Started</b> .....	<b>3</b>
<b>Distribution of Applications</b> .....	<b>4</b>
<b>Examples</b>	<b>4</b>
<b>C/C++</b> .....	<b>4</b>
<b>C#</b> .....	<b>5</b>
<b>Visual Basic</b> .....	<b>8</b>
<b>Excel</b> .....	<b>8</b>
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
Reading the Range .....	10
<b>Functional Description</b>	<b>11</b>
<b>Functions</b> .....	<b>11</b>
Initialization/Termination .....	11
ValidateLicense .....	11
Open.....	11
Close .....	11
Configure .....	12
Operation.....	13
Connect.....	13
Disconnect.....	13
SendRequest.....	14
GetConnectionStatus .....	14
GetComPortStatus .....	14
Reading Parameters .....	15
ReadPrimVar .....	15
ReadCurrPerc .....	15
ReadDynVars .....	16
ReadMessage .....	16
ReadDescription .....	17
ReadSensInfo.....	17
ReadOutputInfo.....	18
ReadFinAssNum .....	18
Writing Parameters .....	19
WriteMessage .....	19
WriteDescription.....	19
WriteFinAssNum .....	20
WriteDamping .....	20
WriteRange.....	21

Device Management .....	22
GetDevInfo .....	22
SetAddress .....	22
GetBurstMode .....	23
SetBurstMode .....	24
<b>Constants.....</b>	<b>24</b>
<b>General Information</b>	<b>25</b>
<b>Device Status .....</b>	<b>25</b>
<b>Unit Codes .....</b>	<b>26</b>
<b>Hart MODEMs .....</b>	<b>27</b>

# Overview

The Hart Driver DLL is implementing the Hart communication protocol like the already existing HartDLL 7.0 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.

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<sup>®</sup> or Microflex.

## Typical Service Processing Program Flow

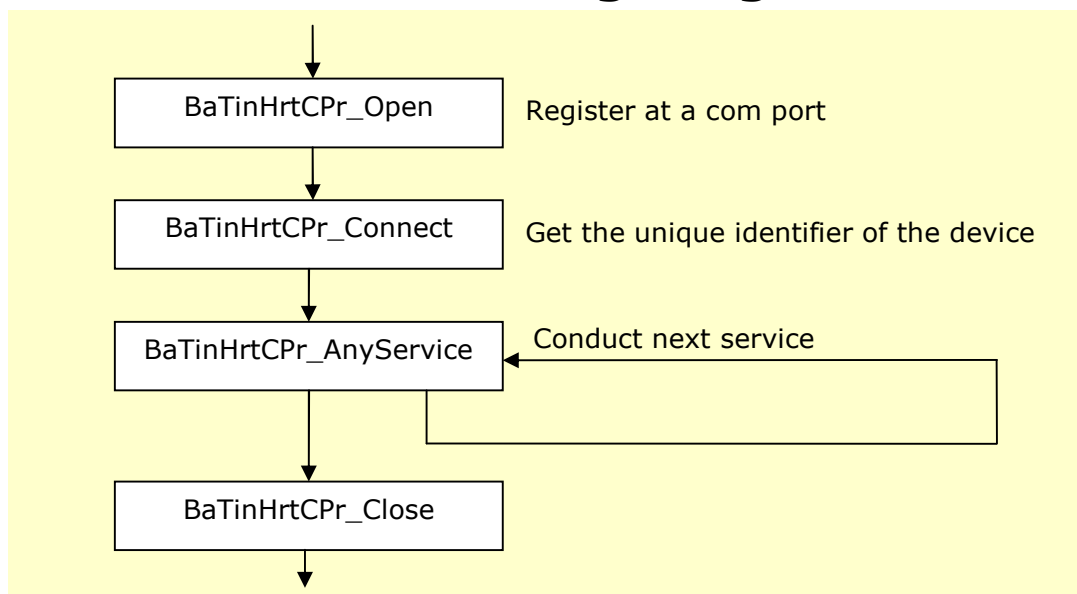


Figure 1: Typical Application Program Flow Structure

When a service is processed 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 delays for a multiple of the number of retries which had been configured by the function BaTinHrtCPr\_Configure.

# Architecture

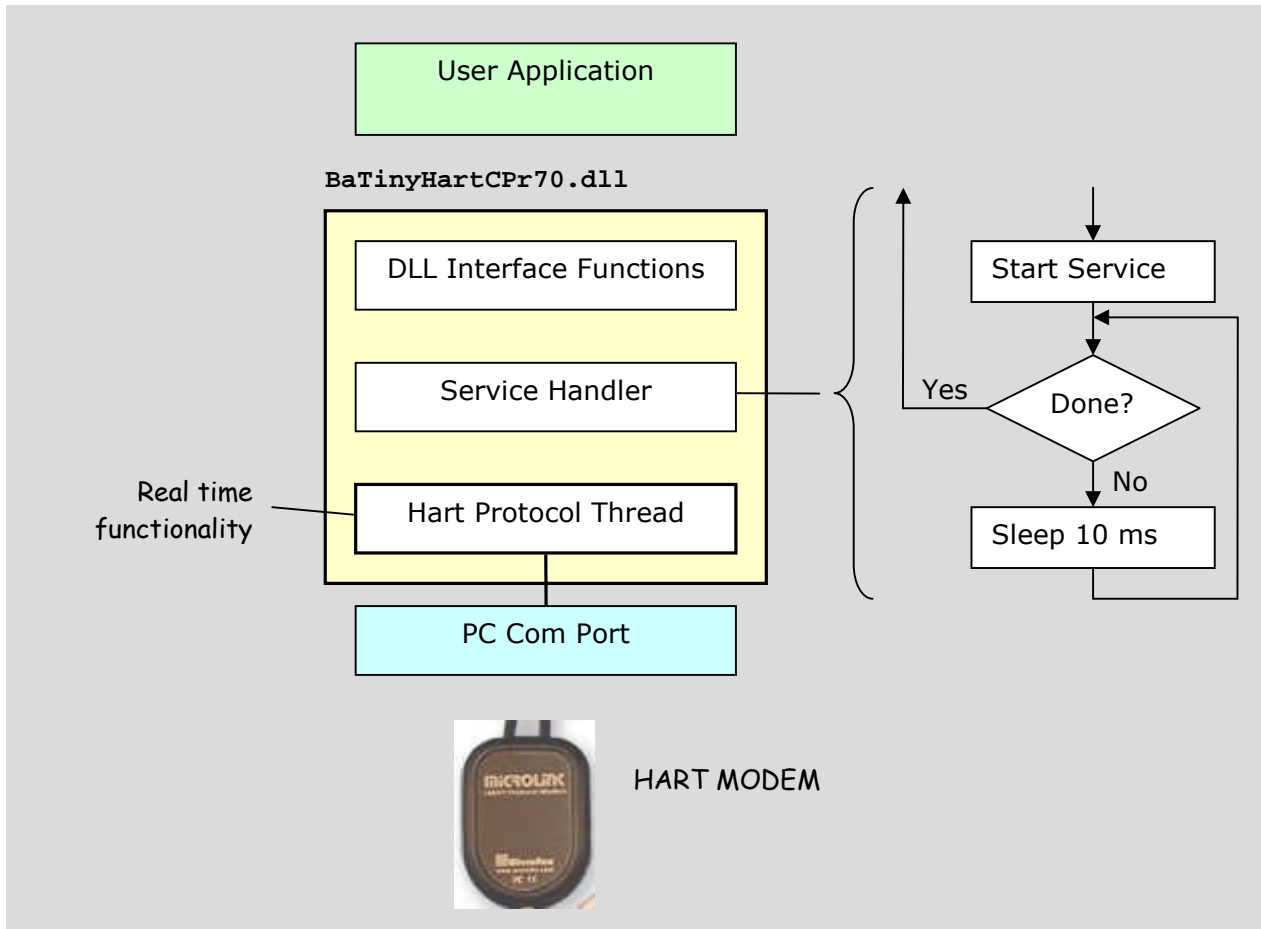


Figure 2: The Internal Structure of the DLL

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.

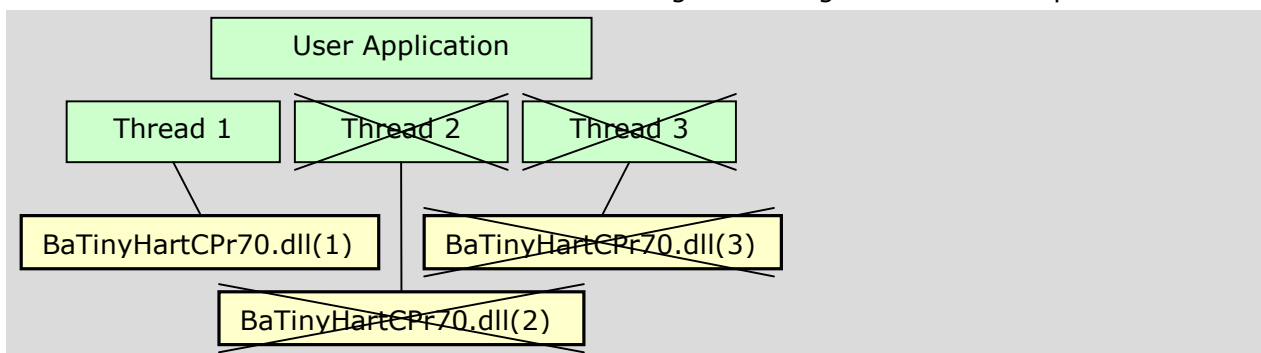


Figure 3: The DLL cannot be used by different Threads

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.

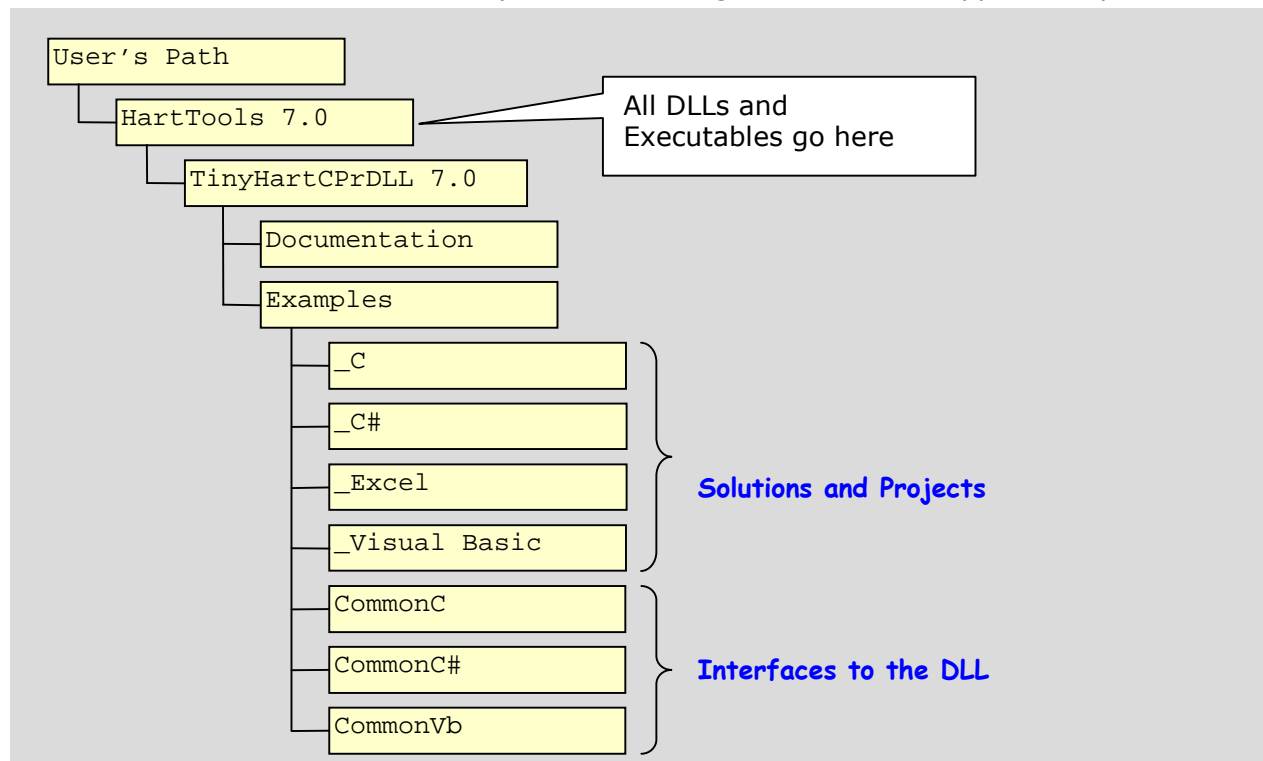


Figure 4: Directory Structure after Unzipping

## 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 `BaTinyHartCPr70.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 (BaTinyHrtCPr70.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 (BaTinHrtCPr\_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 VBA<sup>1</sup>.

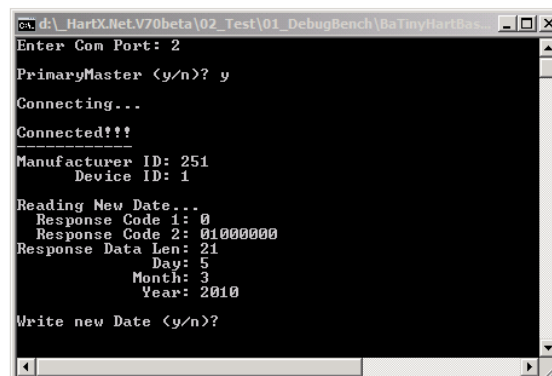
All examples have the same functionality. Firstly a com port is opened and a connection is established. Then the hart command 13 is processed and day, month and year are decoded from the frame (response data). In all examples it is provided to write a new date to the device.

### C/C++

The solution is placed in the path

```
.\HartTools 7.0\TinyHrtCPrDLL 7.0\Examples\_C\BaTinyHrtCPrExmpl.sln
```

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



```
C:\d:\_HartX.Net.V70beta\02_Test\01_DebugBench\BaTinyHrtCPr70.dll
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 BaTinyHrtCPr70.h which is including another general header file (BaHrtDrvGen70.h).

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

```
/* Set the License in the DLL */
BaTinHrtCPr_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.

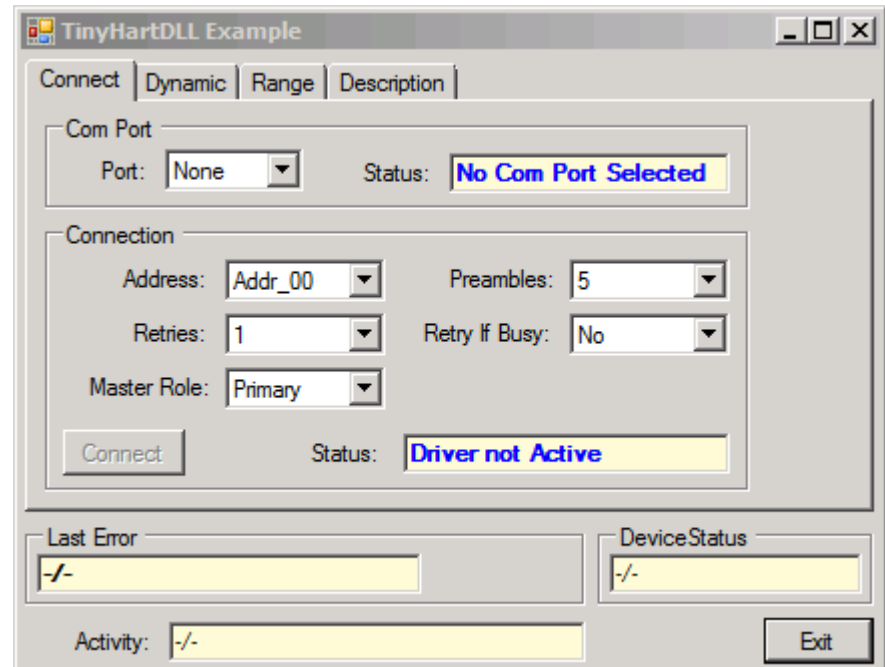
<sup>1</sup> VBA = Visual Basic for Applications (used in Excel)

## C#

The solution is placed in the path

```
.\HartTools 7.0\TinyHartCPrDLL 7.0\Examples\_C#\BaTinyHartCPrExmpl_In_Csharp.sln
```

The example is based on a dialog form.



The interface is declared in the module BaTinyHartCPr70\_Iface.cs which is located in the directory CommonC#.

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

```
#region Private Data
  UInt32          m_uiLastError = 0;
  byte            m_byMyStatus = MYSTAT_IDLE;
  byte[]          m_abyRequestData = new byte[255];
  byte[]          m_abyResponseData = new byte[255];
  //Description
  StringBuilder   m_sbTagName = null;
  StringBuilder   m_sbDescriptor = null;
  byte            m_byDay = 0;
  byte            m_byMonth = 0;
  ushort         m_usYear = 0;
  //etc...
#endregion
```

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

```
TinyHartCPrDLL.BaTinHrtCPr_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(TinyHartCPrDLL.BaTinHrtCPr_GetComPortStatus() == TinyHartCPrDLL.CPS_OK)
{
    TinyHartCPrDLL.BaTinHrtCPr_Close();
}
if(cmbComPort.SelectedIndex != 0)
{
    if(TinyHartCPrDLL.BaTinHrtCPr_Open
        (Convert.ToInt16(cmbComPort.SelectedIndex)) == TinyHartCPrDLL.CPS_OK)
    {
        m_byMyStatus = MYSTAT_HAVING_PORT;
    }
    else
    {
        m_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()
{
    TinyHartCPrDLL.BaTinHrtCPr_Configure(Convert.ToByte(cmbAddress.SelectedIndex),
                                        1,
                                        Convert.ToByte(cmbNumPreambles.SelectedIndex + 2),
                                        Convert.ToByte(cmbNumRetries.SelectedIndex),
                                        Convert.ToByte(cmbRetryIfBusy.SelectedIndex),
                                        Convert.ToByte(cmbMasterRole.SelectedIndex),
                                        0,
                                        10, //Precaution for USB modems
                                        0
                                        );
}

```

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

```

if(TinyHartCPrDLL.BaTinHrtCPr_GetComPortStatus() == TinyHartCPrDLL.CPS_OK)
{
    BeginService();
    SetActivity("Connecting...");
    m_uiLastError = TinyHartCPrDLL.BaTinHrtCPr_Connect(ref m_byLastDeviceStatus);
    DisplayLastError();
    if(m_uiLastError == TinyHartCPrDLL.ERR_OK)
    {
        m_byMyStatus = MYSTAT_CONNECTED;
    }
    else
    {
        m_byMyStatus = MYSTAT_HAVING_PORT;
    }
    CompleteService();
}
}

```

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 ReadDescription button.

```
private void butReadDescription_Click(object sender, EventArgs e)
{
    m_sbTagName = new StringBuilder(8);
    m_sbDescriptor = new StringBuilder(16);
    m_sbTagName.Length = 8;
    m_sbDescriptor.Length = 16;

    BeginService();
    SetDescriptionUpdating();
    SetActivity("Reading Description...");
    m_uiLastError = TinyHartCPrDLL.BaTinHrtCPr_ReadDescription
        ( m_sbTagName, m_sbDescriptor, ref m_byDay, ref m_byMonth,
          ref m_usYear, ref m_byLastDeviceStatus);
    DisplayLastError();
    DisplayConnectionStatus();
    DisplayComPortStatus();
    if(m_uiLastError == TinyHartCPrDLL.ERR_OK)
    {
        DisplayDescription();
    }
    else
    {
        SetDescriptionEmpty();
    }
    CompleteService();
}
```

For writing new values to the description Hart command 18 is used. However, the user does not have to cope with this. He calls a function which is doing all the work.

```
private void butWriteDescription_Click(object sender, EventArgs e)
{
    BeginService();
    SetActivity("Writing Description...");

    m_sbTagName = PreparePackedASCII(txtNewTagName.Text, 8);
    m_sbDescriptor = PreparePackedASCII(txtNewDescriptor.Text, 16);

    m_byDay = Convert.ToByte(cmbNewDay.SelectedIndex + 1);
    m_byMonth = Convert.ToByte(cmbNewMonth.SelectedIndex + 1);
    m_usYear = Convert.ToInt16(cmbNewYear.SelectedIndex + 1900);
    m_uiLastError = TinyHartCPrDLL.BaTinHrtCPr_WriteDescription
        (m_sbTagName, m_sbDescriptor, m_byDay, m_byMonth, m_usYear,
         ref m_byLastDeviceStatus);
    SetDescriptionEmpty();
    CompleteService();
}
```

## 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 BaTinyHartCPr70\_Iface.vb.

For using the StringBuilder object in the following statement

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

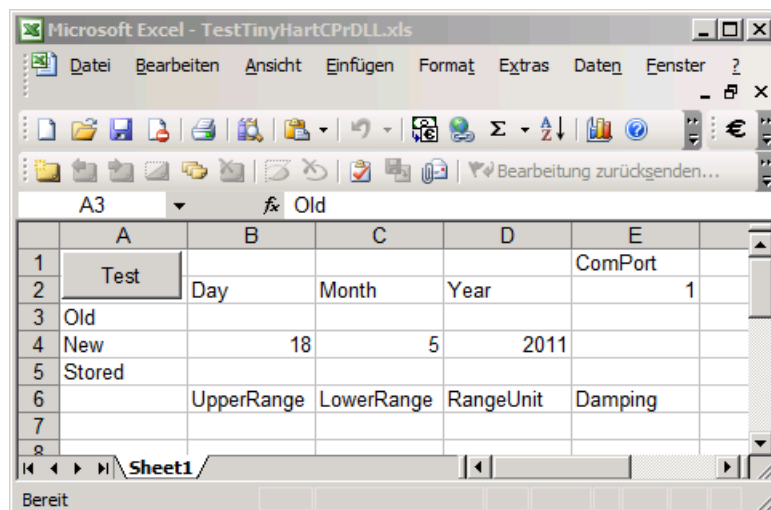
it is required to import the namespace System.Text.

```
Imports System.Text
```

## Excel

As the codes for C# and Visual Basic.Net are very similar using the same virtual machine VBA<sup>2</sup> is completely different.

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



## Getting Started

Double click the file TestTinyHartCPrDLL.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'.

```
lErr = BaTinHrtCPr_ReadOutputInfo(byRangeUnit, fUpperRange, fLowerRange, fDamping,
Range("B7") = fUpperRange
Range("C7") = fLowerRange
Range("D7") = byRangeUnit
Range("E7") = fDamping
Stop
```

<sup>2</sup> 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 BaTinHrtCPr_ReadDescription Lib "BaTinyHartCPr70.dll" _
  (ByVal sTagName As String, _
   ByVal sDescriptor As String, _
   ByRef rbyDay As Byte, _
   ByRef rbyMonth As Byte, _
   ByRef riYear As Integer, _
   ByRef rbyDeviceStatus As Byte _
  ) As Long
  
```

## 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
lErr = BaTinHrtCPr_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")
lErr = BaTinHrtCPr_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
BaTinHrtCPr_Configure 0, 0, 5, 1, 0, 0, 10, 0
lErr = BaTinHrtCPr_Connect(byDeviceStatus)
  
```

## Read Date

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

```

'Read tag descriptor date
lErr = BaTinHrtCPr_ReadDescription(sTagName, sDescriptor, byDay, byMonth, iYear, _
                                   byDeviceStatus)

Range("B3") = byDay
Range("C3") = byMonth
Range("D3") = iYear
Stop
  
```

Day, Month and Year are stored in cells B3, C3 and D3 of the worksheet.

## Write Date

Writing the Date is as easy as reading it.

```
byDay = Range("B4")
byMonth = Range("C4")
iYear = Range("D4")
lErr = BaTinHrtCPr_WriteDescription(sTagName, sDescriptor, byDay, byMonth, iYear, _
                                   byDeviceStatus)
```

## Reading the Range

Reading the range of the output is also very easy because decoding the float data from the Hart communication is done by the DLL.

```
lErr = BaTinHrtCPr_ReadOutputInfo(byRangeUnit, fUpperRange, fLowerRange, fDamping, _
                                   byWrProtCode, byDeviceStatus)

Range("B7") = fUpperRange
Range("C7") = fLowerRange
Range("D7") = byRangeUnit
Range("E7") = fDamping
```

# Functional Description

## Functions

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 int BaTinHrtCPr_ValidateLicense(const char * pcLicenseCode);</code>		
BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70_Iface.cs/vb		
Parameter	Type	Description
pcLicenseCode	In	Pointer to a text containing a valid license code.
<b>Returns</b>		
LICENSE_VALID(0): no error (license was accepted) LICENSE_NOT_VALID(1): error		
<b>Usage</b>		
<pre>TinyHartCPrDLL.BaTinHrtCPr_ValidateLicense     (new StringBuilder("12345678-ABCD-9012-DDDD-1234567TRIAL"));</pre>		

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

#### Open

<code>unsigned int BaTinHrtCPr_Open(unsigned char ucComPort);</code>		
BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70_Iface.cs/vb		
Parameter	Type	Description
ucComPort	In	Number of the PC com port (1..254)
<b>Returns</b>		
CPS_OK(0): driver not active, com port not valid CPS_NOT_VALID(1): driver active		
<b>Usage</b>		
<pre>if(TinyHartCPrDLL.BaTinHrtCPr_Open     (Convert.ToInt16(cmbComPort.SelectedIndex)) == TinyHartCPrDLL.CPS_OK) {</pre>		

#### Close

<code>void BaTinHrtCPr_Close(void);</code>		
BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70_Iface.cs/vb		

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

## Configure

```

unsigned int BaTinHrtCPr_Configure(unsigned char          ucAddr,
                                   unsigned char          ucNumConnRetries,
                                   unsigned char          ucNumPreambles,
                                   unsigned char          ucNumServRetries,
                                   unsigned char          ucRetryIfBusy,
                                   unsigned char          ucMasterRole,
                                   unsigned short         usAddTimeOut,
                                   unsigned short         usAddGapTime,
                                   unsigned short         usAddRtsOffDelay);

```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
ucAddr	In	Hart device address (0..15) Default: 0
ucNumConnRetries	In	Defines how many re-trials the driver is using for establishing a connection Default: 0
ucNumPreambles	In	Number of preambles used for a request (5..20) Default: 5
ucNumServRetries	In	Number of retries if device response is erroneous (0..3) Default: 2
ucRetryIfBusy	In	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
ucMasterRole	In	0: Primary master, 1: Secondary master Default: 0
usAddTimeOut	In	Additional time out to wait for a slave response [ms] Typical 100, 200 etc. Default: 0
usAddGapTime	In	Additional time for gap between characters [ms] Typical 5, 10 etc. Default: 0
usAddRtsOffDelay	In	Additional delay to is switched off Rts(carrier off) [ms] Typical 1, 2, 5, 10 etc. Default: 0

**Returns**

ERR\_OK(0): no error  
ERR\_PAR\_LOC\_RANGE(8): one of the parameters out of limits

**Usage**

```

TinyHartCPrDLL.BaTinHrtCPr_Configure(Convert.ToByte(cmbAddress.SelectedIndex),
                                     1,
                                     Convert.ToByte(cmbNumPreambles.SelectedIndex + 2),
                                     Convert.ToByte(cmbNumRetries.SelectedIndex),
                                     Convert.ToByte(cmbRetryIfBusy.SelectedIndex),
                                     Convert.ToByte(cmbMasterRole.SelectedIndex),
                                     0,
                                     10, //Precaution for USB modems
                                     0
                                     );

```

Usually a call of this function is not necessary. Only if a special setting is desired or required, the default can be overwritten by a call of this function.

**Note:** Configure() has to be called before Connect() is called.

## Operation

### Connect

```
unsigned int BaTinHrtCPr_Connect(unsigned char * pucDeviceStatus);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
pucDeviceStatus	Out	Reference to the device status

#### Returns

ERR\_OK(0): no error  
 ERR\_INVALID\_HANDLE(1): driver not active  
 ERR\_NO\_RESOURCE(2): resource error (e.g. no free service)  
 ERR\_COMM\_ERROR(3): communication error  
 ERR\_NO\_DEV\_RESPONSE(4): no device response

#### Usage

```
m_uiLastError = TinyHartCPrDLL.BaTinHrtCPr_Connect(ref m_byLastDeviceStatus);
```

Note: The device status can only be returned if the connection establishment was successful.

### Disconnect

```
void BaTinHrtCPr_Disconnect(void);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

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 any communication function is called after a call of Disconnect the communication function will return an error.

## SendRequest

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

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

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

ERR\_OK(0): no error  
 ERR\_INVALID\_HANDLE(1): driver not active  
 ERR\_NO\_RESOURCE(2): resource error (e.g. no free service)  
 ERR\_COMM\_ERROR(3): communication error  
 ERR\_NO\_DEV\_RESPONSE(4): no device response  
 ERR\_NOT\_CONNECTED(5): not connected to device  
 ERR\_TRIAL\_VERSION(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 BaTinyHrtCPr_GetConnectionStatus(void);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

### Returns

CST\_CONNECTED(0): connected  
 CST\_NOT\_CONNECTED(1): not connected

## GetComPortStatus

```
unsigned char BaTinyHrtCPr_GetComPortStatus(void);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

### Returns

CPS\_OK(0): O.K.  
 CPS\_NOT\_VALID(1): not valid

## Reading Parameters

### ReadPrimVar

```
unsigned int BaTinHrtCPr_ReadPrimVar(unsigned char * pucPrimVarUnit,
                                     float * pfPrimVarValue,
                                     unsigned char * pucDeviceStatus);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
pucPrimVarUnit	Out	Reference to the unit code of the primary variable
pucDeviceStatus	Out	Reference to the value of the primary variable
pucDeviceStatus	Out	Reference to the device status

#### Returns

```
ERR_OK(0): no error
ERR_INVALID_HANDLE(1): driver not active
ERR_NO_RESOURCE(2): resource error (e.g. no free service)
ERR_COMM_ERROR(3): communication error
ERR_NO_DEV_RESPONSE(4): no device response
ERR_NOT_CONNECTED(5): not connected to device
ERR_TRIAL_VERSION(255): Trial version, number of services exceeded
```

### ReadCurrPerc

```
unsigned int BaTinHrtCPr_ReadCurrPerc(float * pfCurrent,
                                       float * pfPercent,
                                       unsigned char * pucDeviceStatus);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
pfCurrent	Out	Reference to the current value (units of mA)
pfPercent	Out	Reference to the percentage value (units of percent)
pucDeviceStatus	Out	Reference to the device status

#### Returns

See ReadPrimVar()

#### Usage

```
m_uiLastError = TinyHartCPrDLL.BaTinHrtCPr_ReadCurrPerc(ref m_fCurrent, ref m_fPercent,
                                                       ref m_byLastDeviceStatus);
if (m_uiLastError == TinyHartCPrDLL.ERR_OK)
{
    DisplayCurrPerc();
}
else
{
    SetCurrPercEmpty();
}
```

## ReadDynVars

```

unsigned int BaTinHrtCPr_ReadDynVars(float *          pfCurrent,
                                     unsigned char * pucNumValidVars,
                                     unsigned char *  pucPV1_unit,
                                     float *         pfPV1_value,
                                     unsigned char *  pucPV2_unit,
                                     float *         pfPV2_value,
                                     unsigned char *  pucPV3_unit,
                                     float *         pfPV3_value,
                                     unsigned char *  pucPV4_unit,
                                     float *         pfPV4_value,
                                     unsigned char *  pucDeviceStatus);

```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
pfCurrent	Out	Reference to the current value (units of mA)
pucNumValidVars	Out	Reference to the number of valid variables (1..4)
pucPV1_unit	Out	Reference to the unit code of process variable 1
pfPV1_value	Out	Reference to the value of process variable 1
pucPV2_unit	Out	Reference to the unit code of process variable 2
pfPV2_value	Out	Reference to the value of process variable 2
pucPV3_unit	Out	Reference to the unit code of process variable 3
pfPV3_value	Out	Reference to the value of process variable 3
pucPV4_unit	Out	Reference to the unit code of process variable 4
pfPV4_value	Out	Reference to the value of process variable 4
pucDeviceStatus	Out	Reference to the device status

### Returns

See ReadPrimVar()

### Usage

```

m_uiLastError = TinyHartCPrDLL.BaTinHrtCPr_ReadDynVars(ref m_fCurrent, ref m_byValidVars,
                                                       ref m_byPV1_unit, ref m_fPV1_value,
                                                       ref m_byPV2_unit, ref m_fPV2_value,
                                                       ref m_byPV3_unit, ref m_fPV3_value,
                                                       ref m_byPV4_unit, ref m_fPV4_value,
                                                       ref m_byLastDeviceStatus);

```

## ReadMessage

```

unsigned int BaTinHrtCPr_ReadMessage(unsigned char * paucMessage,
                                     unsigned char * pucDeviceStatus);

```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
paucMessage	Out	Reference to an array of 32 bytes to store the message
pucDeviceStatus	Out	Reference to the device status

### Returns

See ReadPrimVar()

## ReadDescription

```
unsigned int BaTinHrtCPr_ReadDescription(unsigned char *   paucTagName,
                                         unsigned char *   paucDescriptor,
                                         unsigned char *   pucDay,
                                         unsigned char *   pucMonth,
                                         unsigned short *  pusYear,
                                         unsigned char *   pucDeviceStatus);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
paucTagName	Out	Reference to an array of 8 bytes to store the tag name
paucDescriptor	Out	Reference to an array of 16 bytes to store the descriptor
pucDay	Out	Reference to the day (1..31)
pucMonth	Out	Reference to the month (1..12)
pusYear	Out	Reference to the year (1900..2155)
pucDeviceStatus	Out	Reference to the device status

### Returns

See ReadPrimVar()

### Usage

```
m_sbTagName = new StringBuilder(8);
m_sbDescriptor = new StringBuilder(16);
m_sbTagName.Length = 8;
m_sbDescriptor.Length = 16;

m_uiLastError = TinyHartCPrDLL.BaTinHrtCPr_ReadDescription
    (m_sbTagName, m_sbDescriptor, ref m_byDay, ref m_byMonth, ref m_usYear,
    ref m_byLastDeviceStatus);
```

## ReadSensInfo

```
unsigned int BaTinHrtCPr_ReadSensInfo(unsigned long * pulSensSerialNum,
                                       unsigned char *   pucLimitsUnit,
                                       float *           pfUpperLimit,
                                       float *           pfLowerLimit,
                                       float *           pfMinSpan,
                                       unsigned char *   pucDeviceStatus);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
pulSensSerialNum	Out	Reference to the sensor serial number (24 bit)
pucLimitsUnit	Out	Reference to unit code for the limits
pfUpperLimit	Out	Reference to the upper sensor limit
pfLowerLimit	Out	Reference to the lower sensor limit
pfMinSpan	Out	Reference to the minimum sensor span
pucDeviceStatus	Out	Reference to the device status

### Returns

```
ERR_OK(0): no error
ERR_INVALID_HANDLE(1): driver not active
ERR_NO_RESOURCE(2): resource error (e.g. no free service)
ERR_COMM_ERROR(3): communication error
ERR_NO_DEV_RESPONSE(4): no device response
ERR_NOT_CONNECTED(5): not connected to device
ERR_PAR_INVALID(7): one or more of the float values are not valid
ERR_TRIAL_VERSION(255): trial version, number of services exceeded
```

## ReadOutputInfo

```
unsigned int BaTinHrtCPr_ReadOutputInfo(unsigned char *   pucRangeUnit,
                                       float *          pfUpperRange,
                                       float *          pfLowerRange,
                                       float *          pfDamping,
                                       unsigned char *   pucDeviceStatus);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
pucRangeUnit	Out	Reference to unit code for the range values
pfUpperRange	Out	Reference to the upper range value
pfLowerRange	Out	Reference to the lower range value
pfDamping	Out	Reference to the damping value (units of seconds)
pucDeviceStatus	Out	Reference to the device status

### Returns

See ReadSensInfo()

### Usage

```
//Read Output Info
m_uiLastError = TinyHartCPrDLL.BaTinHrtCPr_ReadOutputInfo(ref   m_byRangeUnit,
                                                         ref   m_fUpperRange,
                                                         ref   m_fLowerRange,
                                                         ref   m_fDamping,
                                                         ref   m_byWrProtCode,
                                                         ref m_byLastDeviceStatus);

if (m_uiLastError == TinyHartCPrDLL.ERR_OK)
{
    DisplayRange();
}
else
{
    SetRangeEmpty();
    SetDampingEmpty();
}
```

## ReadFinAssNum

```
unsigned int BaTinHrtCPr_ReadFinAssNum(unsigned long *   pulFinAssNum,
                                       unsigned char *   pucDeviceStatus);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
pulFinAssNum	Out	Reference to the final assembly number (24 bit)
pucDeviceStatus	Out	Reference to the device status

### Returns

See ReadPrimVar()

## Writing Parameters

### WriteMessage

```
unsigned int BaTinHrtCPr_WriteMessage(unsigned char * paucMessage,
                                       unsigned char * pucDeviceStatus);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
paucMessage	In	Reference to an array of 32 bytes to get the message from
pucDeviceStatus	Out	Reference to the device status

#### Returns

ERR\_OK(0): no error  
 ERR\_INVALID\_HANDLE(1): driver not active  
 ERR\_NO\_RESOURCE(2): resource error (e.g. no free service)  
 ERR\_COMM\_ERROR(3): communication error  
 ERR\_NO\_DEV\_RESPONSE(4): no device response  
 ERR\_NOT\_CONNECTED(5): not connected to device  
 ERR\_ACC\_DENIED(10): access denied, e.g. device write protected  
 ERR\_TRIAL\_VERSION(255): trial version, number of services exceeded

### WriteDescription

```
unsigned int BaTinHrtCPr_WriteDescription(unsigned char * paucTagName,
                                          unsigned char * paucDescriptor,
                                          unsigned char ucDay,
                                          unsigned char ucMonth,
                                          unsigned short usYear,
                                          unsigned char * pucDeviceStatus);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
paucTagName	In	Reference to an array of 8 bytes to get the tag name from
paucDescriptor	In	Reference to an array of 16 bytes to get the descriptor from
pucDay	In	Day of month (1..31)
pucMonth	In	Month of year (1..12)
pusYear	In	Year (1900..2155)
pucDeviceStatus	Out	Reference to the device status

#### Returns

ERR\_OK(0): no error  
 ERR\_INVALID\_HANDLE(1): driver not active  
 ERR\_NO\_RESOURCE(2): resource error (e.g. no free service)  
 ERR\_COMM\_ERROR(3): communication error  
 ERR\_NO\_DEV\_RESPONSE(4): no device response  
 ERR\_NOT\_CONNECTED(5): not connected to device  
 ERR\_PAR\_LOC\_RANGE(8): at least one parameter out of range (e.g. day > 31)  
 ERR\_ACC\_DENIED(10): Access denied, e.g. device write protected  
 ERR\_TRIAL\_VERSION(255): trial version, number of services exceeded

#### Usage

```
m_sbTagName = PreparePackedASCII(txtNewTagName.Text, 8);
m_sbDescriptor = PreparePackedASCII(txtNewDescriptor.Text, 16);

m_byDay = Convert.ToByte(cmbNewDay.SelectedIndex + 1);
m_byMonth = Convert.ToByte(cmbNewMonth.SelectedIndex + 1);
m_usYear = Convert.ToInt16(cmbNewYear.SelectedIndex + 1900);
m_uiLastError = TinyHartCPrDLL.BaTinHrtCPr_WriteDescription
    (m_sbTagName, m_sbDescriptor, m_byDay, m_byMonth, m_usYear,
     ref m_byLastDeviceStatus);
```

The routine PreparePackedASCII contains the following code.

```
private StringBuilder PreparePackedASCII(string s, byte byTargetStringLength)
{
    StringBuilder sb = new StringBuilder(byTargetStringLength);

    if (s.Length > byTargetStringLength)
    {
        s = s.Substring(0, byTargetStringLength);
    }
    if (s.Length < byTargetStringLength)
    {
        s = s.PadRight(byTargetStringLength, '_');
    }
    s = s.ToUpper();
    sb.Insert(0, s);
    return sb;
}
```

### WriteFinAssNum

```
unsigned int BaTinyHrtCPr_WriteFinAssNum(unsigned long          ulFinAssNum,
                                         unsigned char * pucDeviceStatus);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
pulFinAssNum	In	Final assembly number (24 bit)
pucDeviceStatus	Out	Reference to the device status
<b>Returns</b>		
See WriteMessage()		

### WriteDamping

```
unsigned int BaTinyHrtCPr_WriteDamping(float          fDamping,
                                         unsigned char * pucDeviceStatus);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
fDamping	In	New damping value
pucDeviceStatus	Out	Reference to the device status
<b>Returns</b>		
ERR_OK(0): no error ERR_INVALID_HANDLE(1): driver not active ERR_NO_RESOURCE(2): resource error (e.g. no free service) ERR_COMM_ERROR(3): communication error ERR_NO_DEV_RESPONSE(4): no device response ERR_NOT_CONNECTED(5): not connected to device ERR_PAR_DEV_RANGE(8): device reported range error ERR_ACC_DENIED(10): access denied, e.g. device write protected ERR_TRIAL_VERSION(255): trial version, number of services exceeded		

## WriteRange

```
unsigned int BaTinHrtCPr_WriteRange(unsigned char    ucRangeUnit,
                                   float           fUpperRange,
                                   float           fLowerRange,
                                   unsigned char * pucDeviceStatus);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
pucRangeUnit	In	Unit code for the range values
fUpperRange	In	Upper range value
fLowerRange	In	Lower range value
pucDeviceStatus	Out	Reference to the device status

### Returns

See WriteDamping()

### Usage

```
private void butWriteRange_Click(object sender, EventArgs e)
{
    float fUpperRange = GetFloatFromTextBox(txtNewUpperRange);
    float fLowerRange = GetFloatFromTextBox(txtNewLowerRange);
    byte byRangeUnit = GetByteFromTextBox(txtNewRangeUnit);
    if(fUpperRange != fLowerRange)
    {
        BeginService();
        SetActivity("Writing Range...");
        m_uiLastError = TinyHartCPrDLL.BaTinHrtCPr_WriteRange
            (byRangeUnit, fUpperRange, fLowerRange, ref m_byLastDeviceStatus);
        SetRangeEmpty();
        CompleteService();
    }
    else
    {
        MessageBox.Show("Invalid Range Values!");
    }
}
```

## Device Management

### GetDevInfo

```
unsigned int BaTinHrtCPr_GetDevInfo(unsigned char * pucManufacturerID,
                                     unsigned char * pucDeviceID,
                                     unsigned char * pucSwRev,
                                     unsigned char * pucHwRev,
                                     unsigned char * pucDeviceStatus);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
pucManufacturerID	Out	Reference to the manufacturer ID
pucDeviceID	Out	Reference to the device ID
pucSwRev	Out	Reference to the software revision number (device specific)
pucHwRev	Out	Reference to the hardware revision code (device specific)
pucDeviceStatus	Out	Reference to the device status

#### Returns

ERR\_OK(0): no error  
 ERR\_INVALID\_HANDLE(1): driver not active  
 ERR\_NOT\_CONNECTED(5): not connected to device

Rather than getting the information directly from the device (by command 0) this function is only retrieving the local information which was stored when the connection with the device was established (which requires command 0).

### SetAddress

```
unsigned int BaTinHrtCPr_SetAddress(unsigned char ucNewAddress,
                                     unsigned char * pucDeviceStatus);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
ucNewAddress	In	New device address (0..15)
pucDeviceStatus	Out	Reference to the device status

#### Returns

ERR\_OK(0): no error  
 ERR\_INVALID\_HANDLE(1): driver not active  
 ERR\_NO\_RESOURCE(2): resource error (e.g. no free service)  
 ERR\_COMM\_ERROR(3): communication error  
 ERR\_NO\_DEV\_RESPONSE(4): no device response  
 ERR\_NOT\_CONNECTED(5): not connected to device  
 ERR\_PAR\_LOC\_RANGE(8): address out of range, attempt rejected  
 ERR\_ACC\_DENIED(10): access denied, e.g. device write protected  
 ERR\_TRIAL\_VERSION(255): trial version, number of services exceeded

The function is not only setting the new address in the device but is also immediately re-establishing the connection by using the new address. Thus the execution of this function takes about 1200 ms if it is executed successfully.

### GetBurstMode

```
unsigned int BaTinHrtCPr_GetBurstMode(unsigned char * pucBurstMode,
                                     unsigned char * pucDeviceStatus);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
pucBurstMode	Out	Reference to the burst mode (BURST_ON, BURST_OFF)
pucDeviceStatus	Out	Reference to the device status
<b>Returns</b>		
See GetDevInfo()		

The function is retrieving the information if the device is bursting from the locally stored data which is associated with the actual connection.

#### Why is Burst Mode Important?

If a device (Hart slave) is in burst mode it is drastically reducing the speed for acyclic Hart services because the device is continuously sending cyclic data to both masters even if only one of the masters is connected to the device.

A typical burst message takes about 220 ms to be transmitted. Burst messages are sent in a distance of 100 ms. A master is listening on the communication. If a master is detecting burst frames it enters the synchronous protocol mode. In this mode a master is only allowed to send a request after a burst message or a response to the other master.

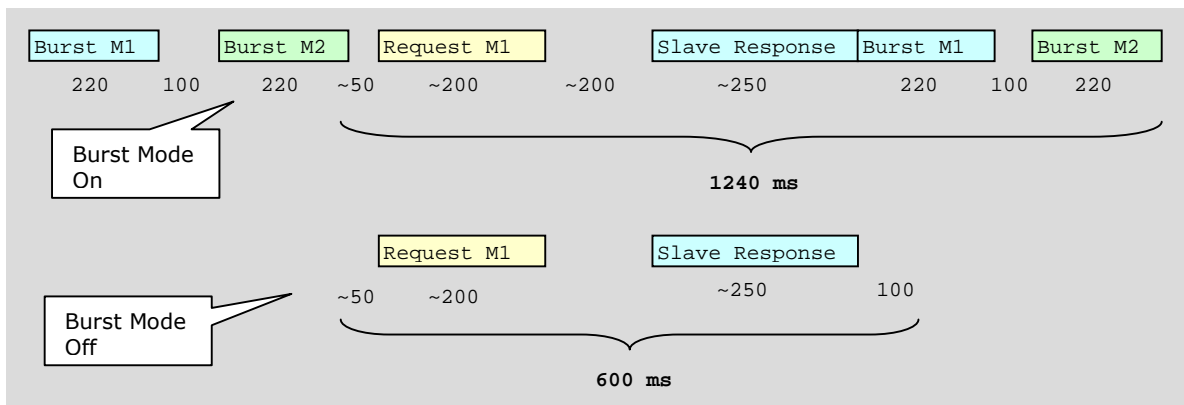


Figure 5: Typical Duration of Hart Services

The figure above shows the timing of Hart services with and without burst messages frames. Therefore it could be of significant advantage if the host can switch of the device's burst mode.

The figure shows also that sending burst messages implies the dual master synchronous mode of the Hart protocol.

## SetBurstMode

```
unsigned int BaTinyHrtCPr_SetBurstMode(unsigned char ucNewBurstMode,
                                       unsigned char * pucDeviceStatus);
```

BaTinyHartCPr70.h, BaTinyHartCPr70.lib, BaTinyHartCPr70\_Iface.cs/vb

Parameter	Type	Description
ucNewBurstMode	In	New burst mode (BURST_OFF, BURST_ON)
pucDeviceStatus	Out	Reference to the device status

**Returns**

ERR\_OK(0): no error  
 ERR\_INVALID\_HANDLE(1): driver not active  
 ERR\_NO\_RESOURCE(2): resource error (e.g. no free service)  
 ERR\_COMM\_ERROR(3): communication error  
 ERR\_NO\_DEV\_RESPONSE(4): no device response  
 ERR\_NOT\_CONNECTED(5): not connected to device  
 ERR\_ACC\_DENIED(10): access denied, e.g. device write protected  
 ERR\_TRIAL\_VERSION(255): trial version, number of services exceeded

## Constants

Name	Value	Description
<b>Error Codes</b>		
ERR_OK	0x0000	No Error
ERR_INVALID_HANDLE	0x0001	The driver is not active (due to a missing com port e.g.)
ERR_NO_RESOURCE	0x0002	It was not possible to launch a service
ERR_COMM_ERROR	0x0003	There was a communication error in the Hart protocol
ERR_NO_DEV_RESPONSE	0x0004	There was no response from the device received
ERR_NOT_CONNECTED	0x0005	The device addressing details are not yet known
ERR_CMD_INVALID	0x0006	The command is not supported by the device
ERR_PAR_INVALID	0x0007	One or more of the requested parameters is not valid
ERR_PAR_LOC_RANGE	0x0008	One or more parameter were out of range when calling the function
ERR_ACC_DEV_RANGE	0x0009	The device detected that one or more parameter were out of range when trying to write new values
ERR_ACC_DENIED	0x000A	The access of a service was denied by the device
ERR_TRIAL_VERSION	0x00FF	Trial version, number of allowed services exceeded
<b>Connection States</b>		
CST_CONNECTED	0x00	The driver is ready to communicate any command
CST_NOT_CONNECTED	0x01	The device addressing details are not yet known
<b>Com Port States</b>		
CPS_OK	0x00	Com port was successfully reserved
CPS_NOT_VALID	0x01	Com port could not be opened
<b>License States</b>		
LICENSE_VALID	0x00	License code was accepted
LICENSE_NOT_VALID	0x01	License code was not accepted
<b>Burst Modes</b>		
BURST_OFF	0x00	Device not bursting / stop burst mode
BURST_ON	0x01	Device is bursting / start burst mode

# General Information

## Device Status

The device status is an 8 bit unsigned integer value with a separate meaning for each bit.

Flag Number / Meaning	Description
Bit #7 Field Device Malfunction	The device has detected a hardware error or failure. Further information may be available through the Read Additional Transmitter Status Command, #48.
Bit #6 Configuration Changed	A write or set command has been executed.
Bit #5 Cold Start	Power has been removed and reapplied resulting in the reinstallation of the setup information. The first command to recognize this condition will automatically reset this flag. This flag may also be set following a Master Reset or a Self Test.
Bit #4 More Status Available	More status information is available than can be returned in the Field Device Status. Command #48, Read Additional Status Information, will provide this additional status information.
Bit #3 Primary Variable Analog Output Fixed	The analog and digital analog outputs for the Primary Variable are held at the requested value. They will not respond to the applied process.
Bit #2 Primary Variable Analog Output Saturated	The analog and digital analog outputs for the Primary Variable are beyond their limits and no longer represent the true applied process.
Bit #1 Non Primary Variable Out of Limits	The process applied to a sensor, other than that of the Primary Variable, is beyond the operating limits of the device. The Read Additional Transmitter Status Command, #48, may be required to identify the variable.
Bit #0 Primary Variable Out of Limits	The process applied to the sensor for the Primary Variable is beyond the operating limits of the device.

## Unit Codes

The following table is only informational; it is not part of a specification.

Unit	Code	Unit	Code	Unit	Code
<b>Temperature</b>		<b>Volume</b>		<b>Mass</b>	
C°	32	gal	40	g	60
F°	33	l	41	kg	61
R°	34	ImpGal	42	ton	62
K	35	m <sup>3</sup>	43	lb	63
<b>Pressure</b>		bbl	46	shton	64
inH <sub>2</sub> O	1	bush	110	lton	65
inHg	2	yd <sup>3</sup>	111	ounce	125
ftH <sub>2</sub> O	3	ft <sup>3</sup>	112	<b>Viscosity</b>	
mmH <sub>2</sub> O	4	in <sup>3</sup>	113	cs	54
mmHg	5	liq bbl	124	cP	55
psi	6	Norm m <sup>3</sup>	166	<b>Voltage</b>	
Bar	7	Norm l	167	mV	36
mBar	8	Std ft <sup>3</sup>	168	V	58
g/cm <sup>2</sup>	9	hl	236	<b>Resistance</b>	
kg/cm <sup>2</sup>	10	<b>Length</b>		Ohm	37
Pa	11	ft	44	kOhm	136
kPa	12	m	45	<b>Energy</b>	
torr	13	in	47	Nm	69
ATM	14	cm	48	dekatherm	89
MPa	237	mm	49	ftlbf	126
inH <sub>2</sub> O(4 C°)	238	<b>Time</b>		kWh	128
mmH <sub>2</sub> O(4 C°)	239	min	50	MJ	164
<b>Volumetric Flow</b>		s	51	Btu	165
ft <sup>3</sup> /min	15	h	52	Mcal	162
gal/min	16	day	53	<b>Power</b>	
l/min	17	<b>Mass Flow</b>		kW	127
ImpGal/min	18	g/s	70	HP	129
m <sup>3</sup> /h	19	g/min	71	Mcal/h	140
gal/s	22	g/h	72	MJ/h	141
MGal/day	23	kg/s	73	Btu/h	142
l/s	24	kg/min	74	<b>Radial Velocity</b>	
l/day	25	kg/h	75	deg/s	117
ft <sup>3</sup> /s	26	kg/day	76	rev/s	118
ft <sup>3</sup> /day	27	ton/min	77	rev/min	119
m <sup>3</sup> /s	28	ton/h	78	<b>Miscellaneous</b>	
m <sup>3</sup> /day	29	ton/day	79	Hz	38
ImpGal/h	30	lb/s	80	µS	56
ImpGal/day	31	lb/min	81	%	57
Norm m <sup>3</sup> /h	121	lb/h	82	pH	59
Norm l/h	122	lb/day	83	mS/cm	66
Std Ft <sup>3</sup> /min	123	shton/min	84	µS/cm	67
ft <sup>3</sup> /h	130	shtom/h	85	N	68
m <sup>3</sup> /min	131	shton/day	86	degBrix	101
bbl/s	132	lton/h	87	ppm	139
bbl/min	133	lton/day	88	°	143
bbl/h	134	<b>Mass per Volume</b>		pF	153
bbl/day	135	SGU	90	not used	250
gal/h	136	g/cm <sup>3</sup>	91	none	251
ImpGal/s	137	kg/m <sup>3</sup>	92	unknown	252
l/h	138	lb/gal	93	special	253
gal/day	235	lb/ft <sup>3</sup>	94	Please refer to the Hart Communication Foundation (HCF) specifications for codes not listed in this table.	
<b>Velocity</b>		g/ml	95		
ft/s	20	kg/l	96		
m/s	21	g/l	97		
in/s	114	lb/in <sup>3</sup>	98		
in/min	115	shton/yd <sup>3</sup>	99		
ft/min	116	degTwad	100		
m/h	120	degBaum hv	102		
<b>Current</b>		degBaum lt	103		
mA	39	degAPI	104		

---

## Hart MODEMs

Borst Automation recommends MicroLink MODEMs by the Microflex Company. For detailed information see:  
[http://www.microflx.com/Microlink\\_USB.htm](http://www.microflx.com/Microlink_USB.htm).