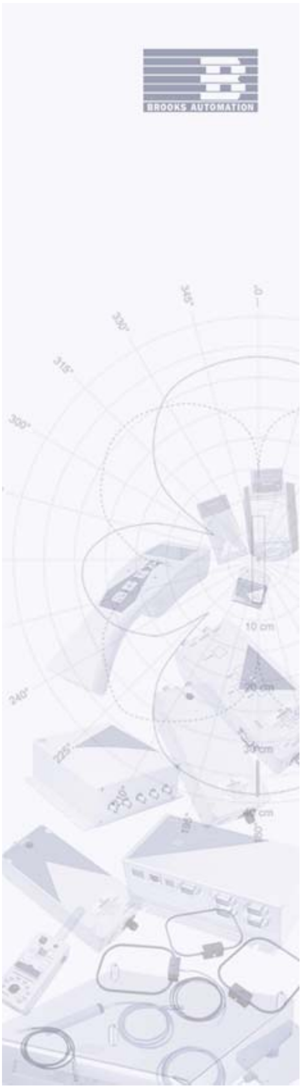


Description

ASCHFProtLib.dll

Version 1.2

ID050054
Rev 11-2005
Printed in Germany
Subject to modifications

TABLE OF CONTENTS

1 METHODS AND FUNCTIONS

1.1 HFProt

Definition: public HFProt

The function returns no value as it is the constructor function.

NOTE: All objects and timers created at construction must be disposed off neatly while destruction using Dispose.

1.2 Dispose

Definitions: public void Dispose()

The function closes all open ports, releases all memory space and stops any running timers. This function MUST be called while disposing the class' object. This could be required, for example, upon termination of the program. Calling this function ensures a neat garbage collection as well as termination of any running communication.

1.3 OpenRSCom

Definition: public bool OpenRSCom(int baud, string port)

Parameter: int baud
Baudrate of the serial interface.

Example: 19200, 38400, 57600 etc.

Parameter: string port
Number of the used COM port.

Example: „COM1“, „COM2“ etc.

The function returns a value of type boolean. A positive response means that the COM port was opened successfully. A negative response means that the COM could not be opened.

1.4 CloseRSCom

Definition: public bool CloseRSCom()

The function closes the open COM port and responses a success value.

1.5 OpenSocket

Definition: public bool OpenSocket(string IPAdd, int port)

Parameter: string IPAdd
Is the IP address of the communication partner (transponder reader).

Example: „127.0.0.1“, „192.168.100.101“ etc.

Parameter: int port
Is the TCP/IP port of the communication partner.

Example: „1001“, „3241“ etc.

The function returns a value of type boolean. A positive response means that the function to connect the device was started. The Event *OnConnect* confirms the successful connection. A negative response means that the port request was not started successfully.

1.6 CloseSocket

Definitions: public bool CloseSocket()

The function starts the closing of the currently opened Ethernet connection. The event *OnConnect* confirms the closing of the connection.

1.7 SendRS232

Definition: public bool SendRS232(string cmd)

Parameter: string cmd

The parameter ‘cmd’ of type string is a arbitrary command string which is implemented within the protocol specification.

Examples:

„N0“	reset command
„V0“	request software version
„F001“	request value of parameter ‚01’

The return value shows the success of the sending of the message. If there is no open interface the function returns ‘false’.
The library calculates the complete protocol string automatically (start sign, length, command, end sign and checksum).

1.8 SendSocket

Definition: public bool SendSocket(string cmd)

Parameter: string cmd

The parameter ‘cmd’ of type string is a arbitrary command string which is implemented within the protocol specification. Before you can use this function a Ethernet connection must be established by using the function OpenSocket(IPAdd,Port).

Examples:

„N0“	reset command
„V0“	request software version
„F001“	request value of parameter ‚01’

The return value shows the success of the sending of the message.
The library calculates the complete protocol string automatically (start sign, length, command, end sign and checksum).

2 EVENTS

2.1 MsgRS232Received

Definition: public delegate void MsgRS232Received(object sender, string AscMsg)

Event: public event MsgRS232Received OnRS232AscMsg

The event OnRS232AscMsg provides a command string of type string. The string contains a message command as defined in the ASC-I1 communication protocol of the device.

Example:

Request:	SendRS232(„H0“)
AscMsg =	„h012340000“
Request:	SendRS232(„V0“)
AscMsg =	„v0524956332E302E30“(„RIV3.0.0“)

2.2 RS232MsgError

Definition: public delegate void RS232MsgError(object sender, int err)

Event: public event RS232MsgError OnRS232MsgErr;

The event OnRS232MsgErr provide an error code.
Err = 1: checksum error in the received message
Err = 2: timer timeout between two received signs

2.3 EthConnect

Definition: public delegate void EthConnect(object sender,
bool connected)

Event: public event EthConnect OnEthConnect;

The event provides the status of the Ethernet connection.
Is **connected = true** the the connection was established successfully.
Is **connected = false** then the connection was closed.

2.4 MsgEthReceived

Definition: public delegate void MsgEthReceived(object sender,
string AscMsg)
Event: public event MsgEthReceived OnEthAscMsg;

The Event OnEthAscMsg provides a message command of type string which is implemented within the ASC-I1 communication protocol.

Example: see OnRS232AscMsg

2.5 EthMsgError

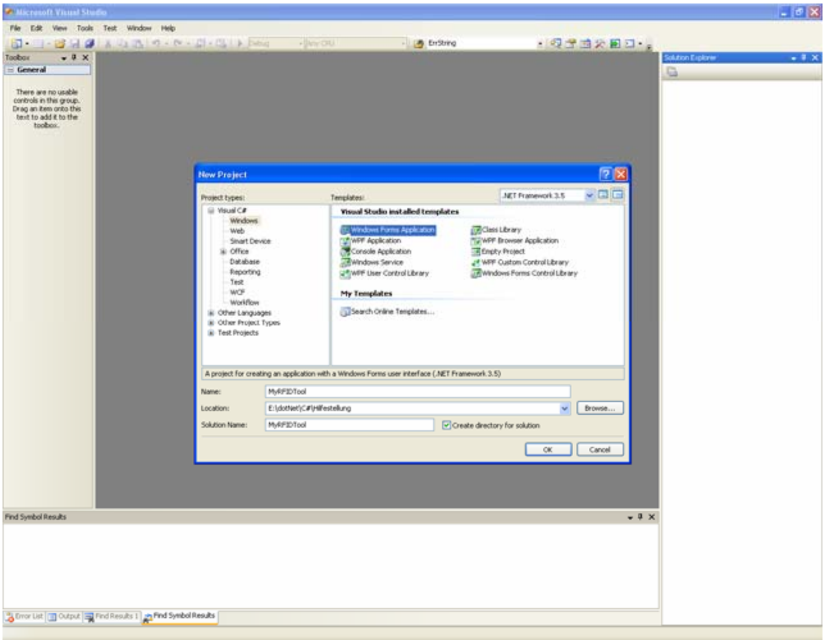
Definition: public delegate void EthMsgError(object sender, int err)

Event: public event EthMsgError OnEthMsgErr;

The event OnEthMsgErr provides an error code.
Err = 2 timer timeout between two received signs

3 SOURCE CODE TEST APPLICATION

1. Open your Visual Studio Environment and create a new Project, *MyRFIDTool* as a Windows Form Application.



A project for creating an application with a Windows Forms user interface (.NET Framework 3.5)

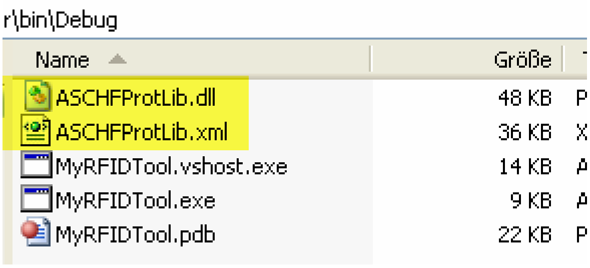
Name: MyRFIDTool

Location: E:\dotNet\C#\Hilfestellung

Solution Name: MyRFIDTool

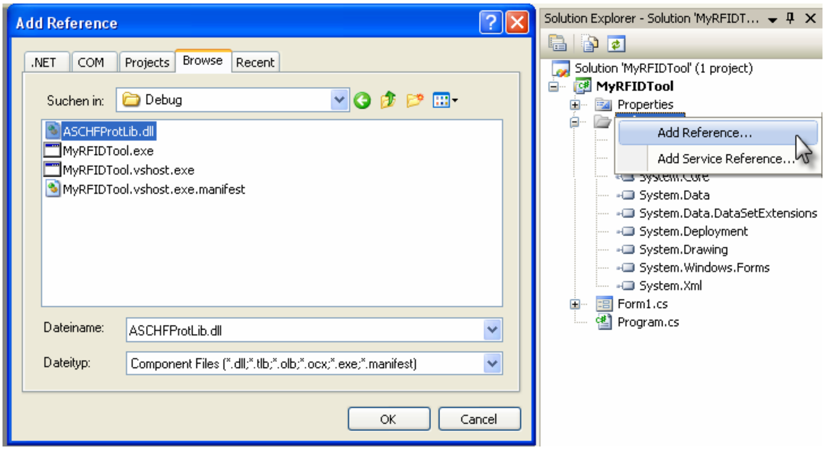
[✓] Create directory for solution

2. To the Project, add the Reference to the ASCHFProtLib.dll. Please make sure that the DLL as well as the documentation XML are stored together with the executable application:



r\bin\Debug

Name	Größe	T
ASCHFProtLib.dll	48 KB	P
ASCHFProtLib.xml	36 KB	X
MyRFIDTool.vshost.exe	14 KB	A
MyRFIDTool.exe	9 KB	A
MyRFIDTool.pdb	22 KB	P



Suchen in: Debug

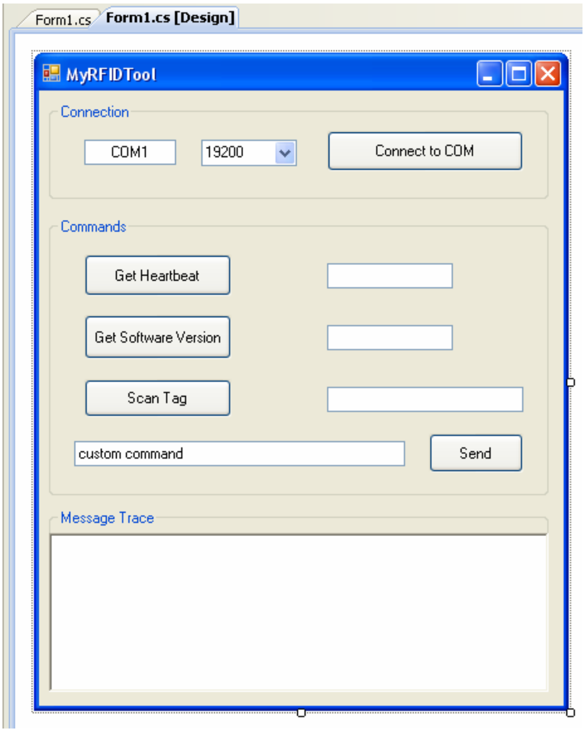
ASCHFProtLib.dll
MyRFIDTool.exe
MyRFIDTool.vshost.exe
MyRFIDTool.vshost.exe.manifest

Dateiname: ASCHFProtLib.dll

Dateityp: Component Files (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)

Solution 'MyRFIDTool' (1 project)
MyRFIDTool
Properties
System.Core
System.Data
System.Data.DataSetExtensions
System.Deployment
System.Drawing
System.Windows.Forms
System.Xml
Form1.cs
Program.cs

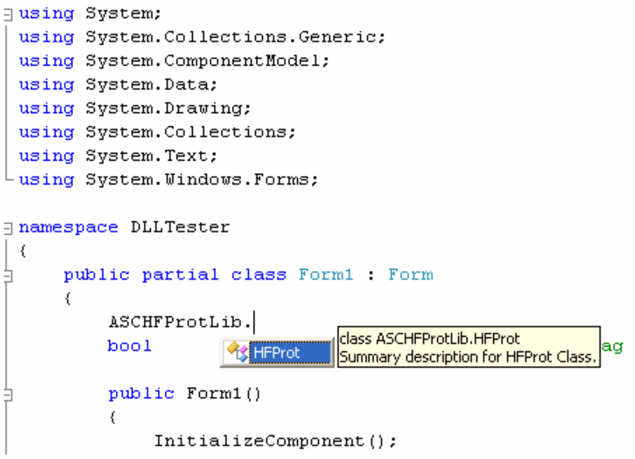
3. Design the Form such that there is a possibility to enter I.P. Address or COM Port to connect to the Brooks RFID Reader, send commands and follow the message traffic.



COM1 19200

custom command

4. In you code, create an instance of the ASCHFProtLib.dll



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Collections;
using System.Text;
using System.Windows.Forms;

namespace DLLTester
{
    public partial class Form1 : Form
    {
        ASCHFProtLib.
        bool          HFProt   class ASCHFProtLib.HFProt          ag
                               Summary description for HFProt Class.

        public Form1()
        {
            InitializeComponent();
```

5. Initialize the object and create a Click event for the Connect Button.

NOTE: It is important to dispose the class upon termination of your program. In this example, we shall call the FormClosing event of the Windows Form.

```
public Form1()
{
    InitializeComponent();

    m_Prot = new ASCHFProtLib.HFProt();
    m_Prot.OnRS232AscMsg += new
        ASCHFProtLib.HFProt.MsgRS232Received(OnRS232AscMsg);
}

private void Form1_FormClosing(object sender,
                                        FormClosingEventArgs e)
{
    if (m_Prot != null)
    {
        if (m_Prot.RS232Connected)
            m_Prot.CloseRSCom();             //Close port

        m_Prot.Dispose ();                  //Free the object!
        m_Prot = null;
    }
}
```

```
private void Btn_Connect_Click(object sender, EventArgs e)
{
    if (!m_Connected)                //If NOT already connected
    {
        if (m_Prot != null)     //safe to check if object exists
        {
            //Open the port now.
            if (m_Prot.OpenRSCom (Convert.ToInt32(CB_Baud.Text),
                                             TB_COMPort.Text)) ;
            {
                m_Connected = m_Prot.RS232Connected;
            }
        }
    }
    else                             //We are already connected!!
    {
        m_Prot.CloseRSCom();
        m_Connected = m_Prot.RS232Connected;
    }
}
```

6. Create events for all the buttons, and fill them as below:

```
private void Btn_Heartbeat_Click(object sender, EventArgs e)
{
    m_Prot.SendRS232("H0");
}

private void Btn_Version_Click(object sender, EventArgs e)
{
    m_Prot.SendRS232("V0");
}

private void Btn_Send_Click(object sender, EventArgs e)
{
    m_Prot.SendRS232(TB_Cmd.Text);
}
```

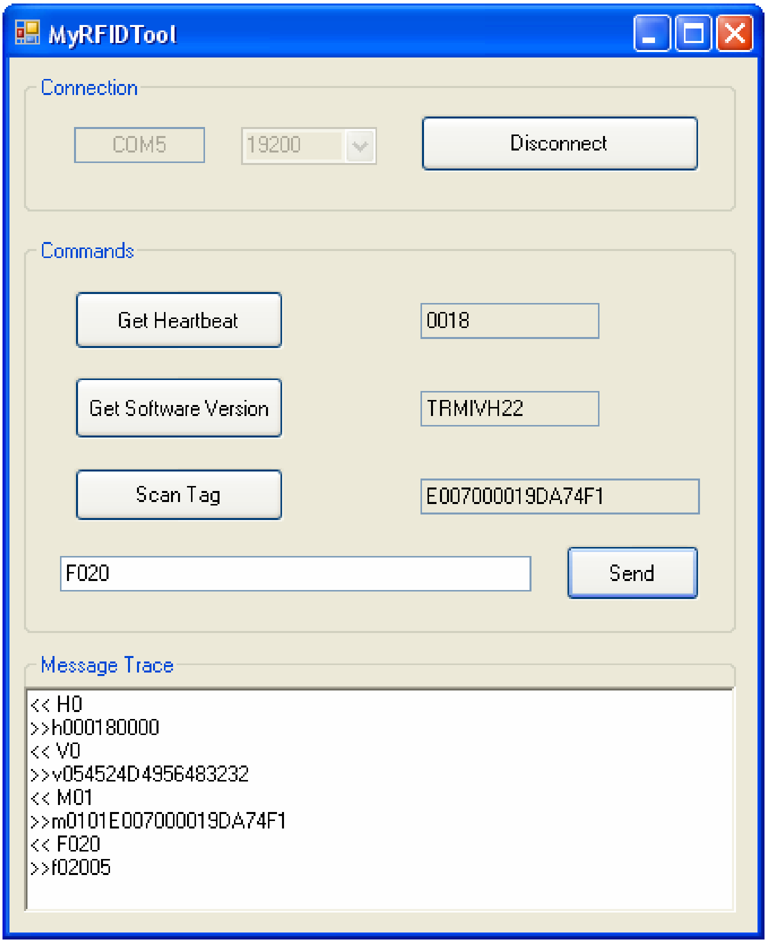
7. Catch the fired events (declared while creating the object) in their respective functions:

```
void OnRS232AscMsg(object sender, string AscMsg)
{
    //process incoming messages in another function
    processASCMsg(AscMsg);
}
```

8. Process incoming messages as desired. Incoming and outgoing messages can also be displayed, if desired. For such purposes, where a Windows Form Control is to be updated, InvokeRequired is recommended.

9. For detailed code and exception catching, please refer to the sample Project accompanying this documentation.

10. Run your application. Connect to a given COM Port, send heartbeat or get software Version!



COM5 | 19200

Get Heartbeat: 0018
Get Software Version: TRMIVH22
Scan Tag: E007000019DA74F1
F020

```
<< H0
>>h000180000
<< V0
>>v054524D4956483232
<< M01
>>m0101E007000019DA74F1
<< F020
>>f02005
```