Tutorial on the MomanLib

Summary

This AppNote presents a brief tutorial about how to set up your environment (e.g. Visual Studio). It continues with replicating the C# - examples delivered in the Library. The advanced part with design techniques is found in AppNote 179.

Applies To

All Motion Control applications, in combination with a PC environment
MC V2.5 and MC V3.0

Table of Contents

Licensing

Visual C#, *Visual Studio* and *MSDN* are Trademarks registered by Microsoft. There may be Additional Terms and / or Conditions for licensed third party software components. For a full list see https://www.microsoft.com/en-us/legal/intellectualproperty/trademarks/en-us.aspx
Log4Net is software provided by the Apache Foundation. There may be Additional Terms and / or Conditions for licensed third party software components.

Getting the Software

Visual Studio

1. Go on https://visualstudio.microsoft.com/vs/older-downloads/ or search for „Visual Studio 2013 Express“. If you feel comfortable with using a later version, that is okay too.
2. Sign In with your credentials or create a new one.
    a. If your account is created recently and you are not seeing any downloads, you may want to join the “Visual Studio Dev Essentials”-Program, which is free (as of Sept 2018, should work later too) to see more downloads.
    Joining the Program is possible under the tab “Subscriptions” [Figure 1: Downloads for Visual Studio green box]
3. Select the downloads for Visual Studio 2013

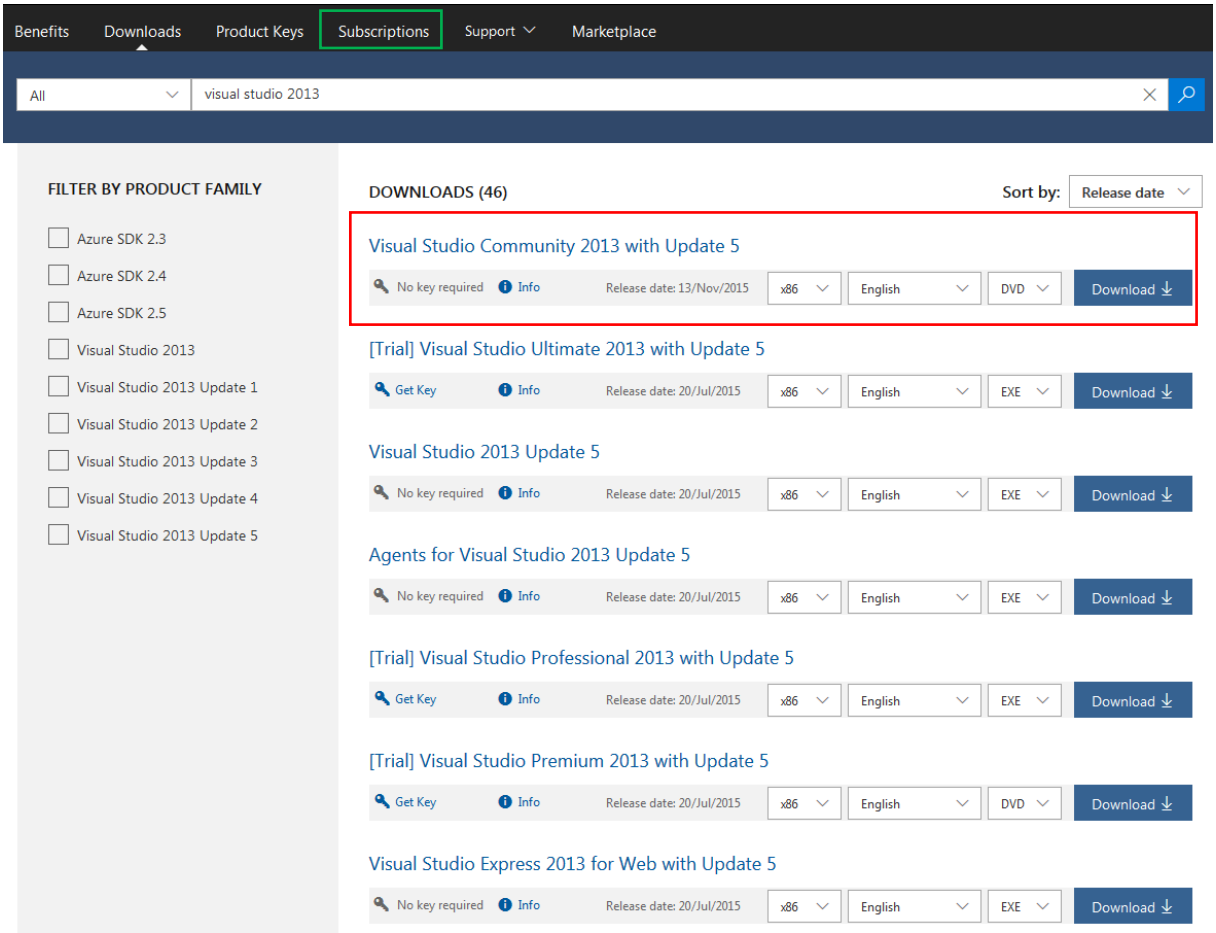


Figure 1: Downloads for Visual Studio

4. Install Visual Studio (typically “Visual Studio Community 2013 with Update 5” would be a good choice, [see Figure 1: Downloads for Visual Studio red box])

5. TIP: If you want to follow the Tutorial with the Visual Studio Language „English“ you can use the following Language Pack (or search for „Visual Studio 2013 Language Pack“ and select English)
   https://my.visualstudio.com/Downloads?q=Visual%20Studio%202013%20Language%20Pack
   Or here https://www.microsoft.com/en-us/download/details.aspx?id=40766

   Alternatively you can select „Tools“, then „Options“.
   Select the List-Item „Environment“, then „International Settings“

Faulhaber MomanLib

1. Download the Library
   a. Go on https://www.faulhaber.com/en/support/drive-electronics/#c65284
   b. Navigate to “Libraries”
2. Select the MoManLib (Programming Library for Windows 32- and 64-bit)
3. Download it to your local Project Folder
4. **If you just want to try the Library**, feel free to use the Examples (C++, C#, Delphi and LabView) provided under “./Examples/Source/”, for C# open “DemoCSharp.csproj” after Visual Studio is installed.

Example Program – Step by Step

If you only want to use the source code, there is a complete version at the end of this chapter:

- Codelisting 11: Final source code of the FormMain.cs
- Codelisting 12: Final source code of the MomanLibSample.cs

Creating the Project

1. Select „New Project“

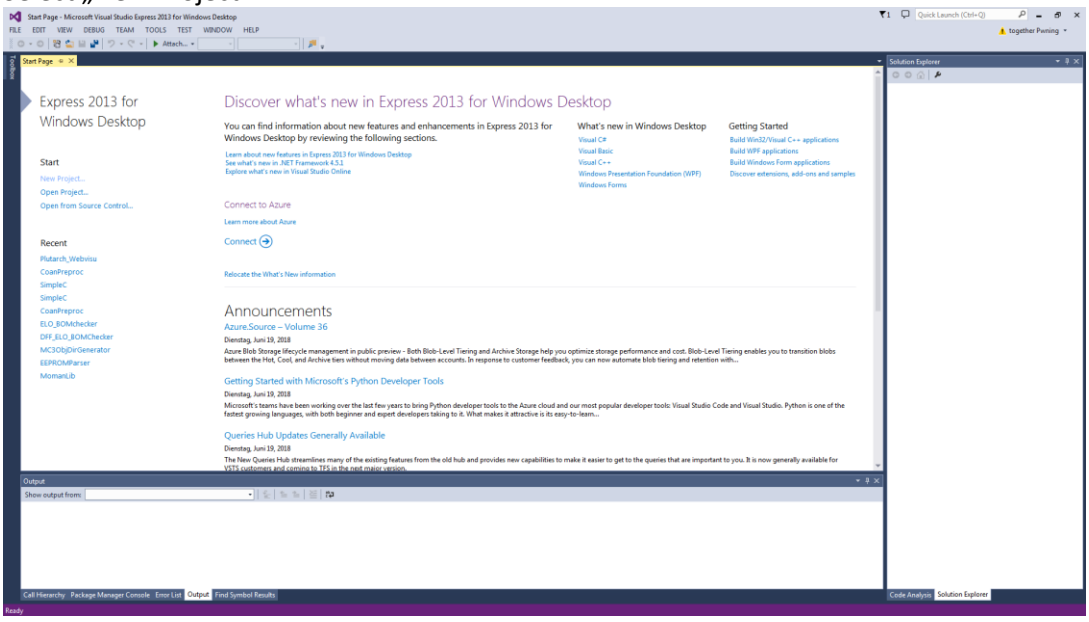


Figure 2: Visual Studio 2013 Project Overview

2. Then Create the Project

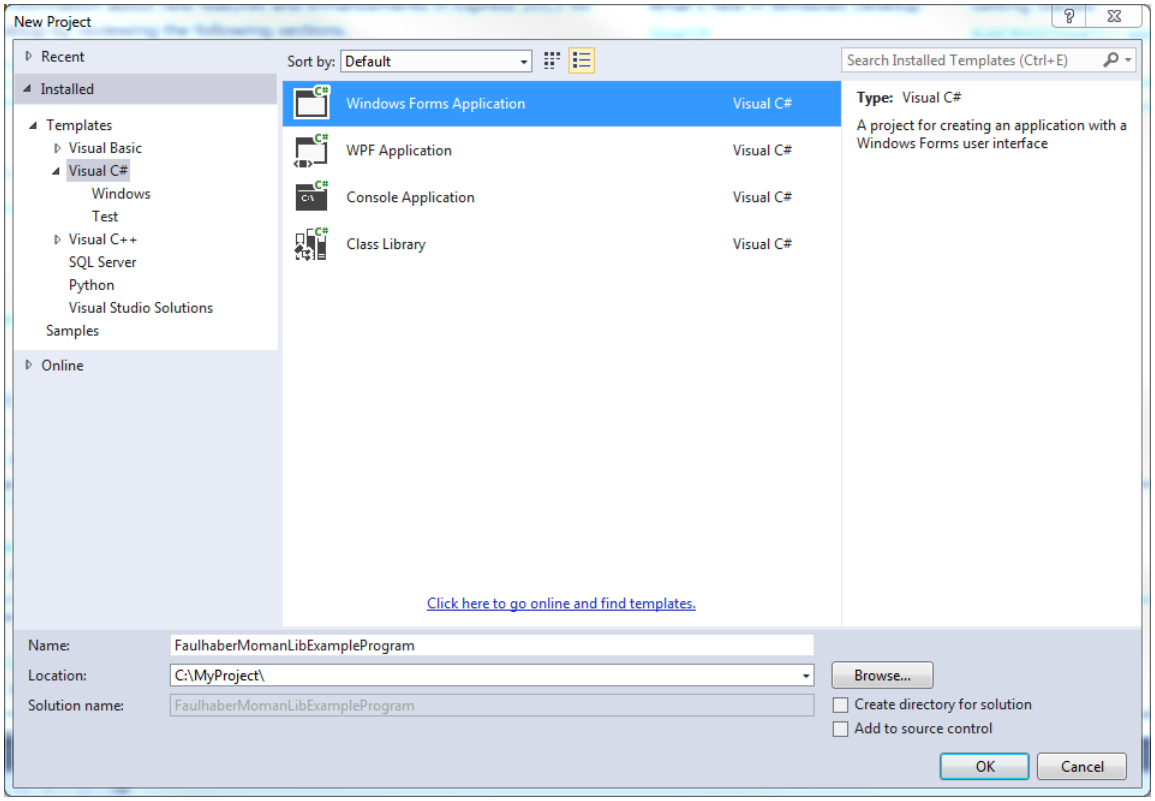


Figure 3: Create a new Visual C#-Project

3. This may be your standard view

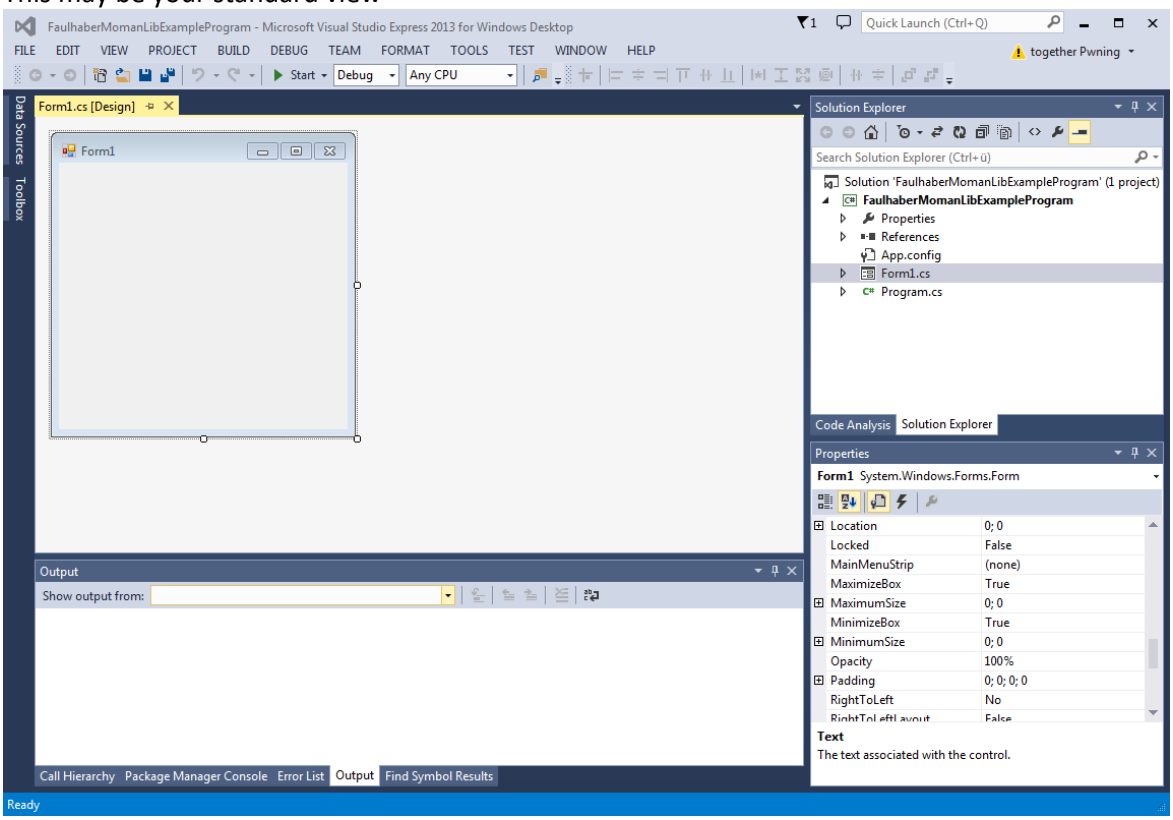


Figure 4: The starting view of a default project

4. Create the Folder „lib“ in the C# Project folder and extract the MomanLib into the Folder „MomanLib“ here. Take notice of the chosen path here.

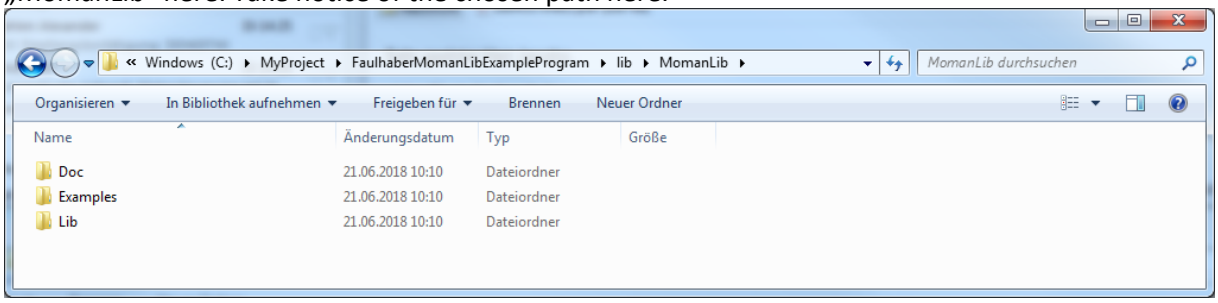


Figure 5: Folder overview

5. You are good to go!

Creating the User Interface

1. Renaming the Main Form From „Form1.cs“ to „FormMain.cs“

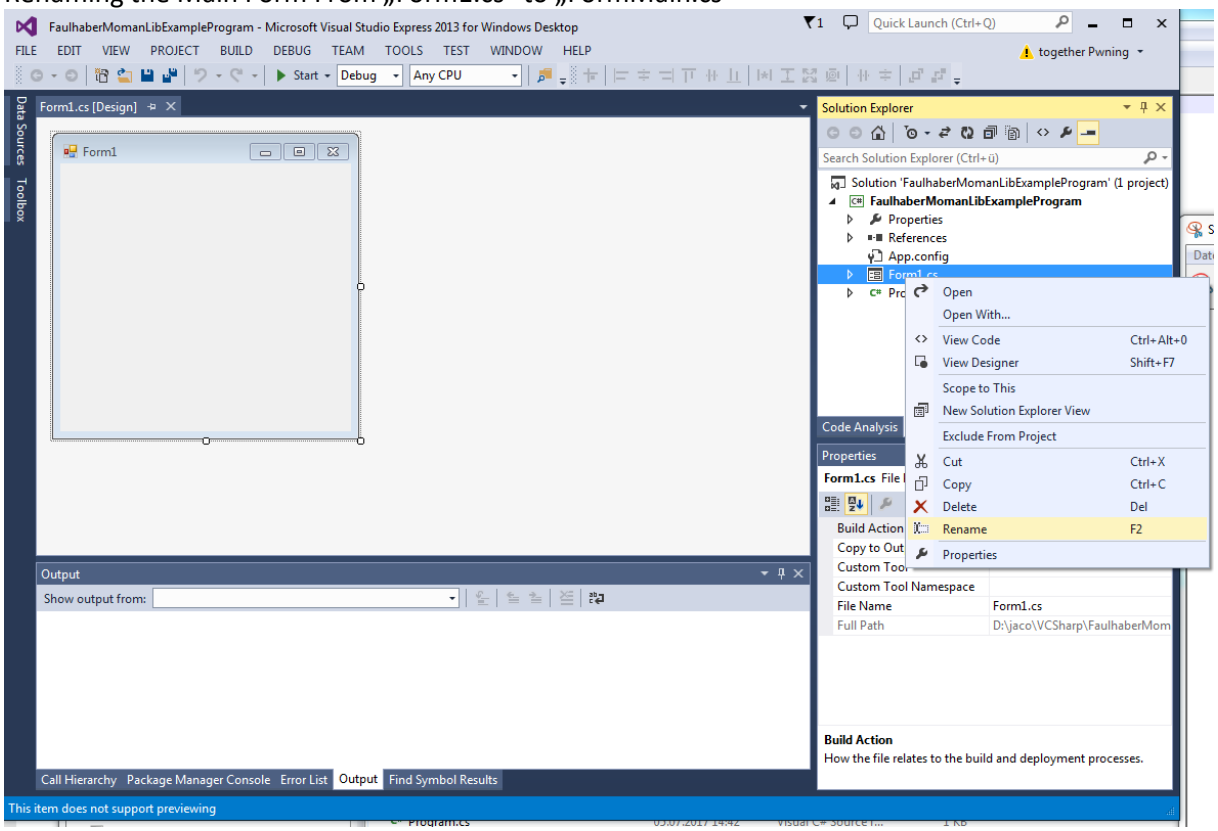


Figure 6: Renaming the default Form File

When asked if you want to rename all references press „yes“

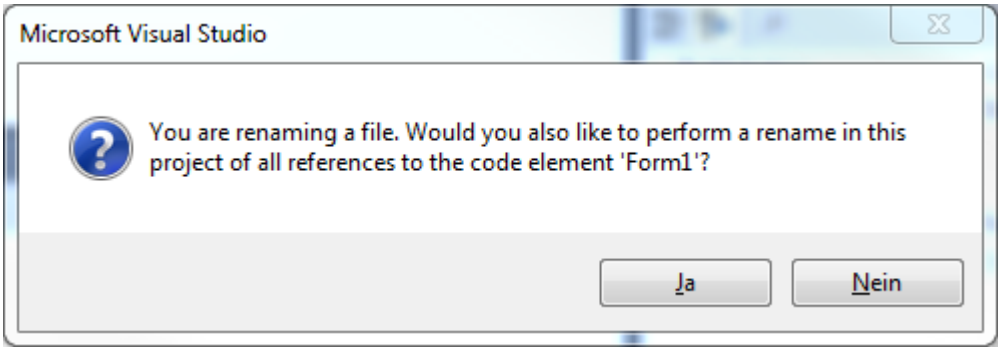


Figure 7: Renaming dialog

2. Resizing the Form and Renaming the Form Title

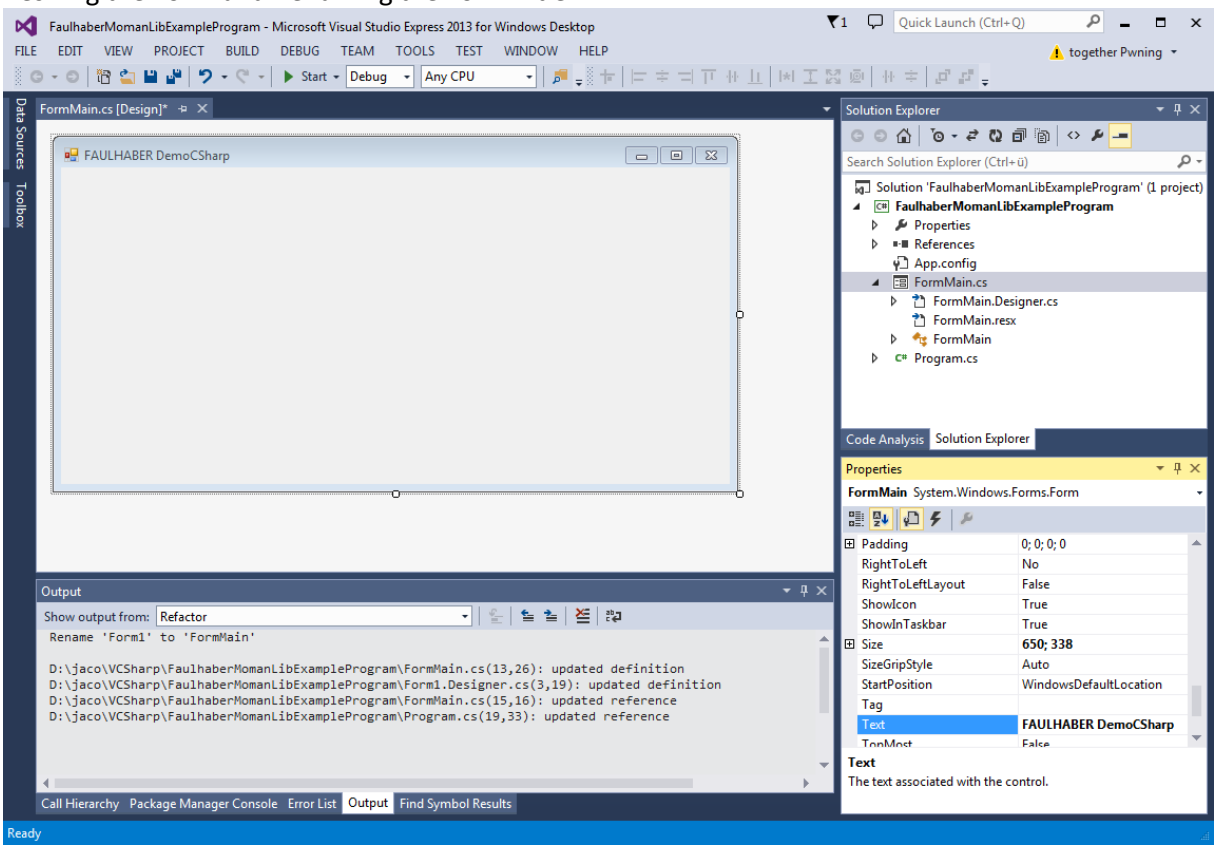


Figure 8: Resizing the Form and renaming it

3. Add the Buttons via „Toolbox“ → „Button“ and Drag them into the Form

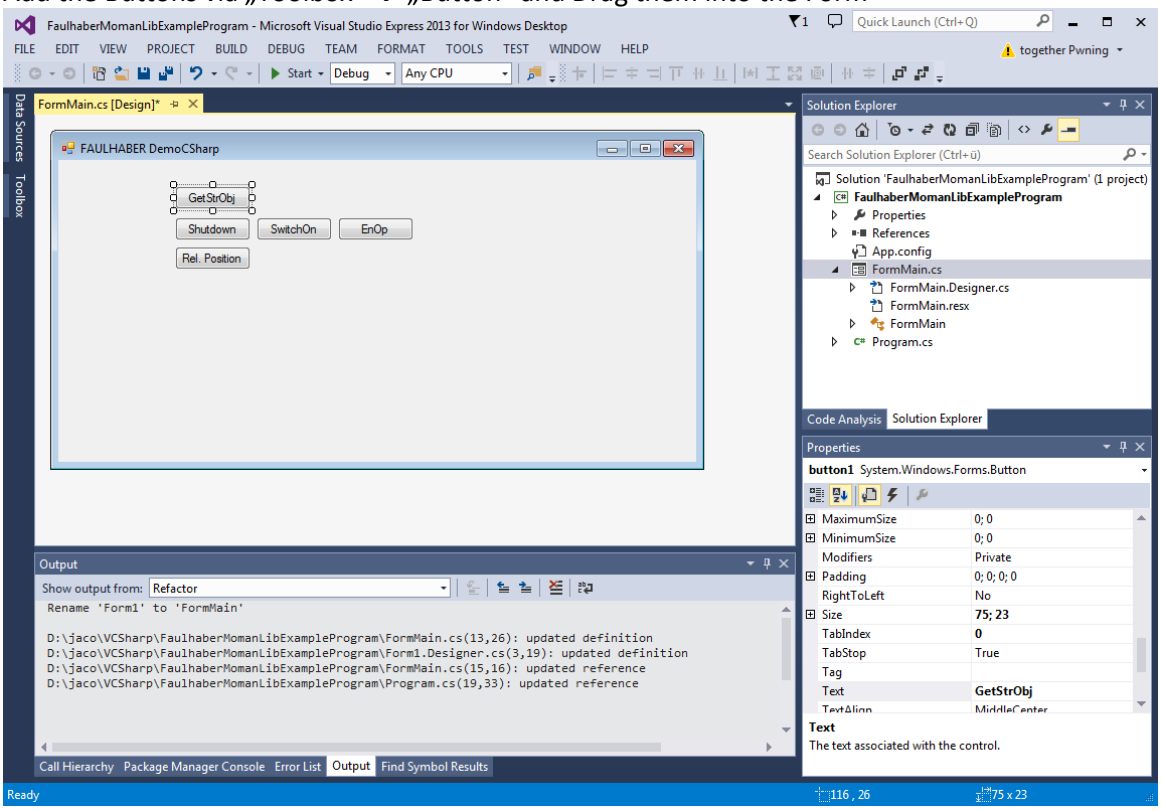


Figure 9: Creating the Buttons

4. Add the TextBox and make it multiline, the fit it in the Form with a nice border as you like.

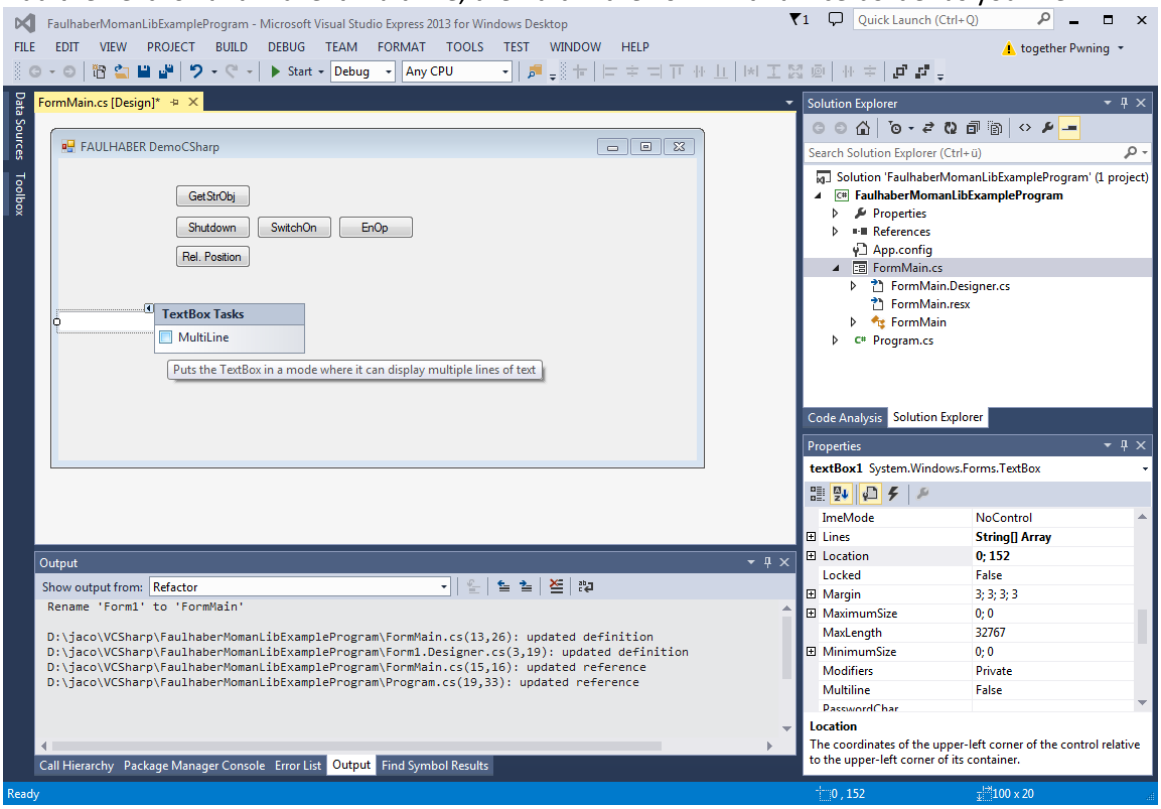


Figure 10: Creating the Textbox

5. Add the StatusStrip by dragging the element on the Form

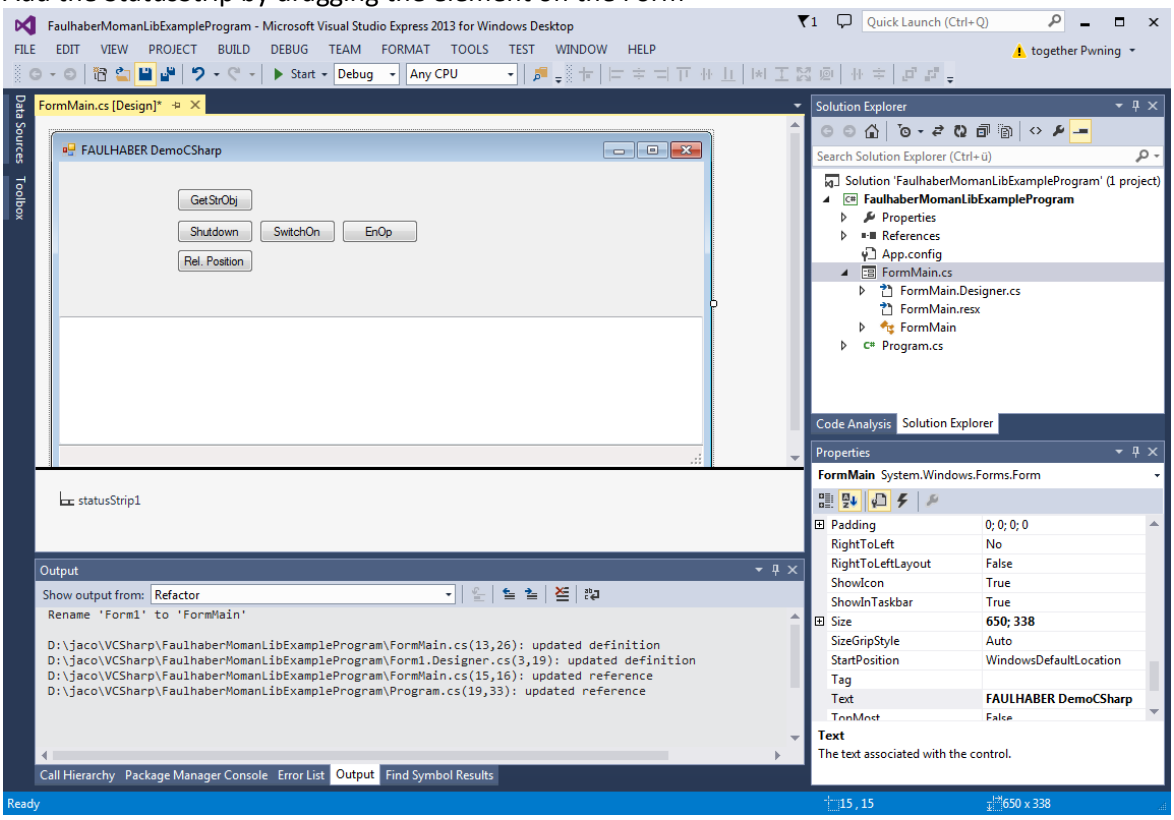


Figure 11: Adding the Status Strip

6. Adding the Labels

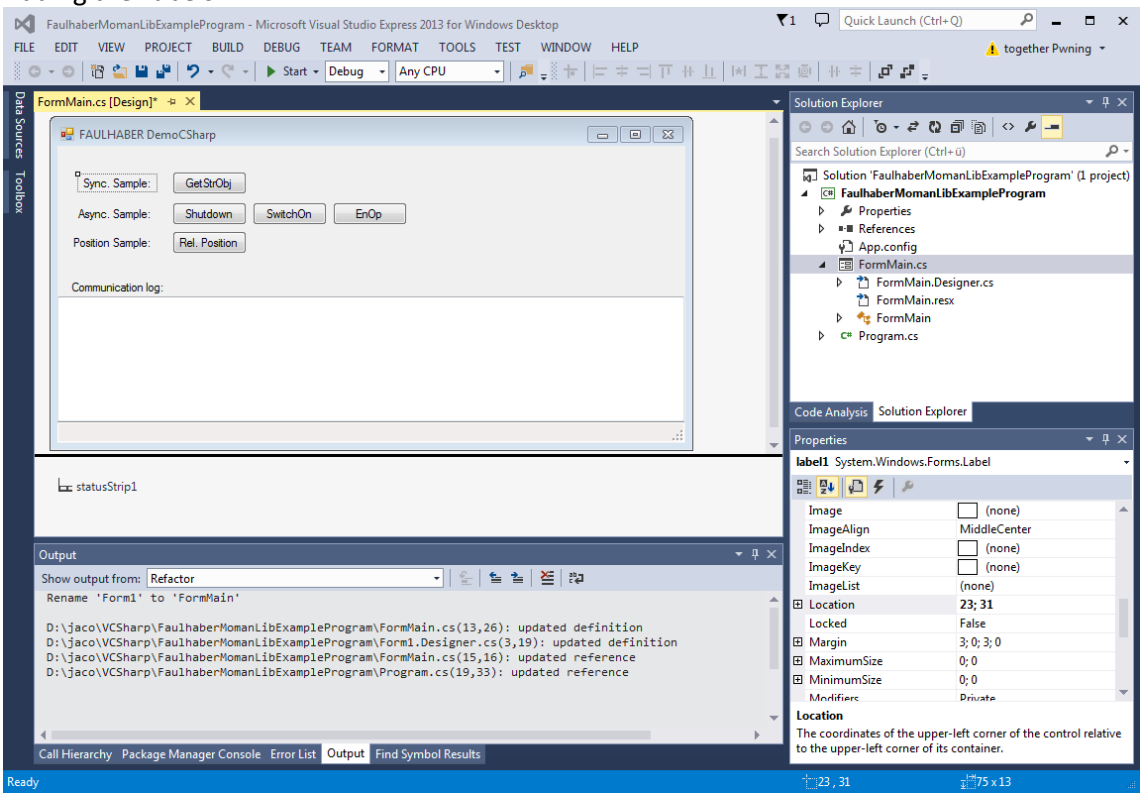


Figure 12: Adding the Labels

Connecting the MomanLib to the Project

Why do we need to do this?

To call native C++ Code in C#, we need a Wrapper (“Interface”). This is achieved by creating a method in C# for each method you want to import. When using a naming scheme, it is common praxis to name the methods of the wrapper exactly like the library names, and then rename the imported ones with one or two underscores. For some function calls you will need some advanced knowledge about marshalling and pointers, but with a little C++ knowledge you shouldn’t have a problem. In case you don’t understand what’s going on, just copy the files from the example or follow the next chapter.

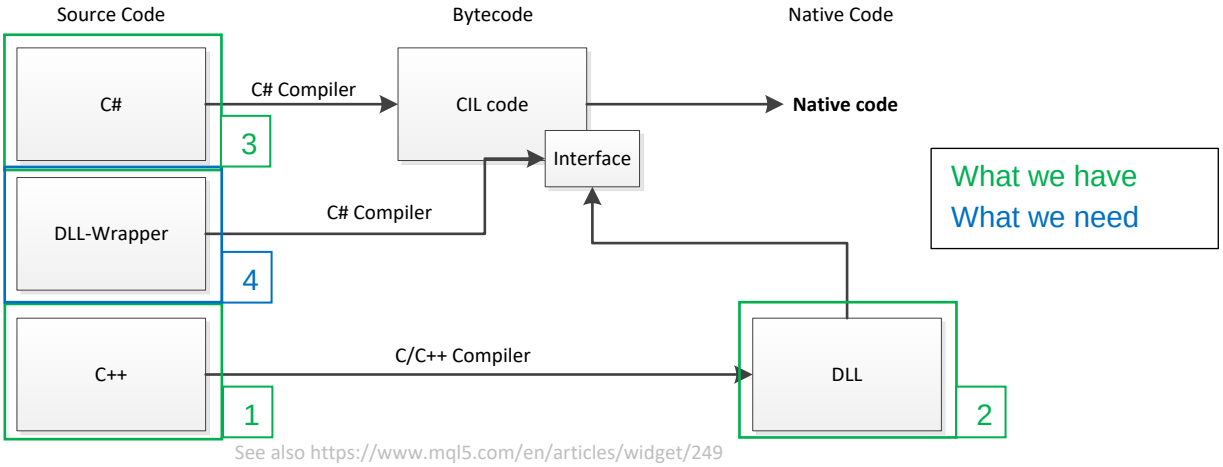


Figure 13: Overview of the C#-Architecture

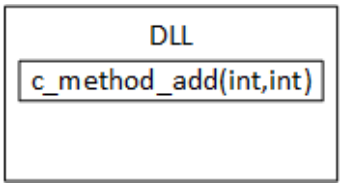
What is a Wrapper

A wrapper is a Software structure, which mimics or copies the behavior and the Methods of a Software Layer. Imagine, we would have a function in C++ like the following (simplified version):

Codelisting 1: Simplified export of an add method

```
void c_method_add(int a, int b)
{
	return a + b;
}
```

This Method would get translated and compiled to a DLL. The result would be an exported Function inside this DLL. Exported Functions inside a DLL are visible by any Program that understands the File format DLL.



So what a Program would see when loading the DLL is the following: This would describe the Path between the 1 and the 2 in Figure 13: Overview of the C#-Architecture. So what needs to be described now is the Part 3 of the Figure: How C# accesses the DLL and finally how the wrapper (Part 4) fits in this problem.

Figure 14: How the DLL-Export could look like

So the Wrapper to the previous example would be:

```
class Wrapper
{
    static int add(int a, int b)
    {
        if(true/*import successful*/)
            return Import._add(a, b);
        else
            throw new Exception("Cannot import!");
    }
    class Import
    {
        public const string cDLLPath = "C:/path/to/dl
        [DllImport(
            cDLLPath,
            EntryPoint = "c_method_add",
            CallingConvention = CallingConvention.Std
        ]
        public extern static int _add(int a, int b);
    }
}
```

- A Class named „Wrapper“
- „add“ is the method we want to import

- Checking if the import was successful
- If yes, we can use the function
- else
- We have to indicate that we have an error

- A class named „Import”

- The DLL-Path, has to be const
- The important part begins: Importing with the options: Path, how the methods real name is in the DLL (entry point), and how the method „c_method_add“ needs its arguments passed to it
- What Type is returned, and what are the arguments, as well as a method name

With this technique you have the possibility to react to failure before calling, return your own debug values, exclude some values, …
The Following chapter will explain how you do this for the MomanLib.

Creating the wrapper

The following parts of the Tutorial will start with a Task that is either ”Read“, ”Create“, or ”Copy“.

1. ”Create“ a new File: MomanLibSample.cs

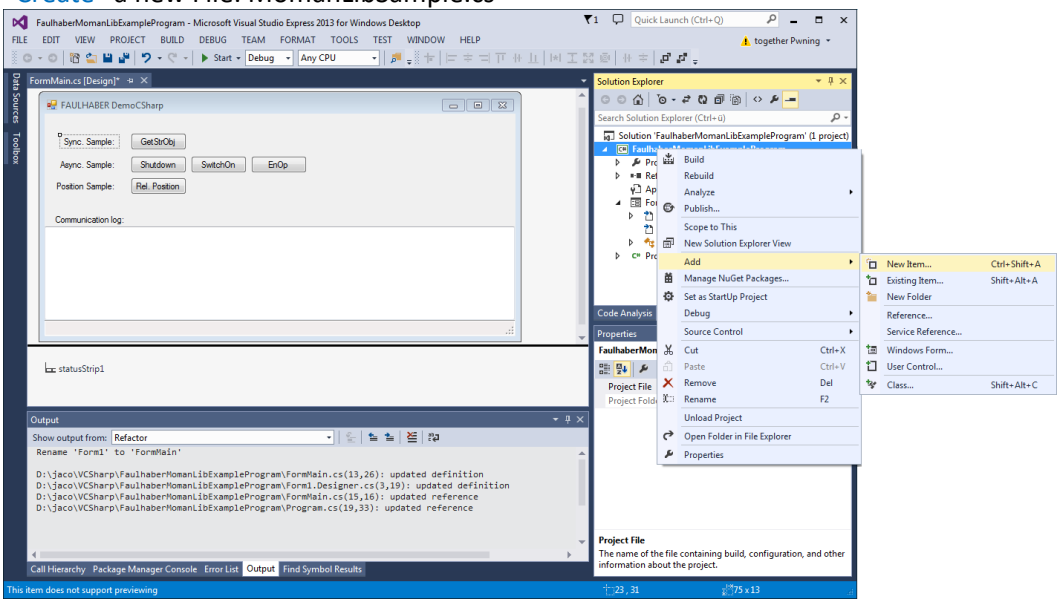


Figure 15: Adding the Wrapper-files

Because the MomanLib has some more complex types to offer than just integers, we will need to declare some Types in beforehand, before we can use the DLL-Imports. Because when we add the Imports, we have everything declared, so it will be ready to use.

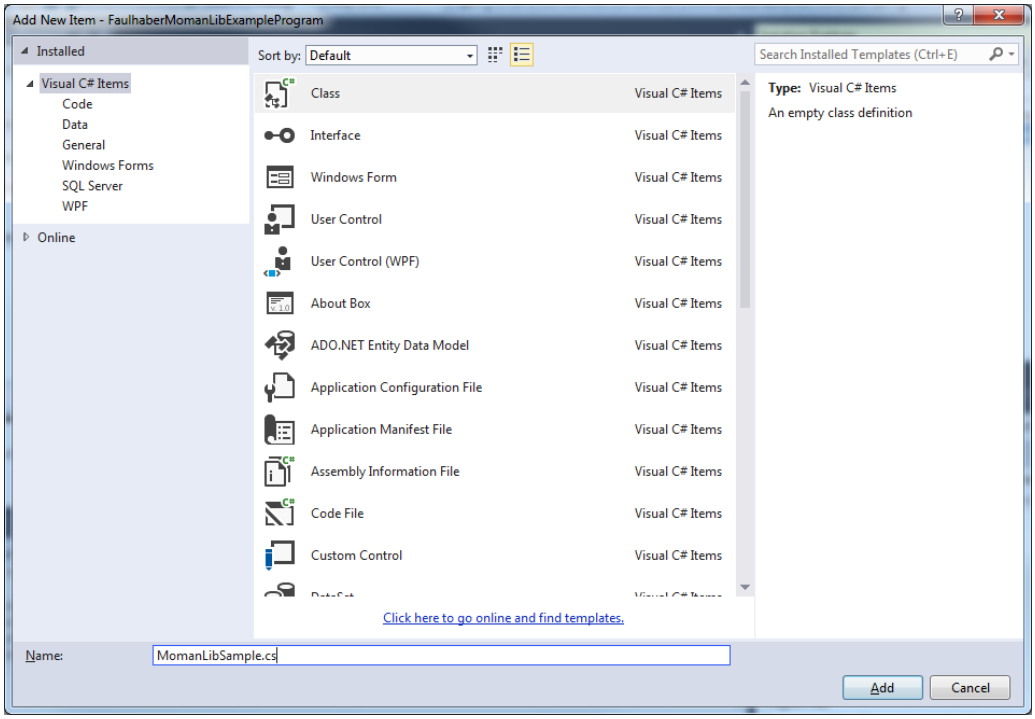


Figure 16: Detailed view of the file creation

2. ”Copy“ – Choose to
   a. Add the Library-Types manual from the MomanCMD.h files located in C:\MyProject\FaulhaberMomanLibExampleProgram\lib\MomanLib\Lib\Include

(only use the rest if you want to use the Trace-Mode[1]).

   b. Alternatively you can use *Codelisting 2*: Definition of the enums used in the library. Copy it to the File created in Step 1, but add it inside the namespace, not inside the class.

Codelisting 2: Definition of the enums used in the library

```
public delegate void tdmmProtDataCallback();
public delegate void tdmmProtTraceValuesCallback(int nodeNr, UInt32[] value,
int timecode);
public enum eMomanprot
{
    eMomanprot_ok_bootup = 2,
    eMomanprot_ok_async = 1,
    eMomanprot_ok = 0,
    eMomanprot_error = -1,
    eMomanprot_error_timeout = -2,
    eMomanprot_error_cmd = -3,
    eMomanprot_error_emcy = -4,
    eMomanprot_error_param = -5,
    eMomanprot_error_accessdenied = -6,
    eMomanprot_error_init = -7,
    eMomanprot_noData = -8
    }
```

The delegates are a Type how a method looks is described in C#- return types aswell as arguments.

The Enum is an Enumeration of Integers with names. One could say, it is collection of named numbers.
`eMomanprot` is there for indicating how the state of the Interface Initialization is, the state whilst opening the Communication and when reading an Answer by the device.

[1] The Trace-mode is used for logging and recording (only MC V3.0) data values of the controller.

```
public enum eMomancmd {
    //Device Control:
    eMomancmd_shutdown = 16,
    eMomancmd_switchon = 17,
    eMomancmd_disable = 18,
    eMomancmd_quickstop = 19,
    eMomancmd_DiOp = 20,
    eMomancmd_EnOp = 21,
    eMomancmd_faultreset = 22,
    eMomancmd_MA = 23,
    eMomancmd_MR = 24,
    eMomancmd_HS = 25
}
public enum eDecoded {
    eDecoded_none = 0,          /*!< none decoded */
    eDecoded_SOBJ = 1,          /*!< SOBJ command decoded */
    eDecoded_GOBJ = 2,          /*!< GOBJ command decoded*/
    eDecoded_Bootup = 3,
    eDecoded_NMT = 4,
    eDecoded_NMTRequest = 5,
    eDecoded_Heartbeat = 6,
    eDecoded_Statusword = 7,
};
```

`eMomancmd` contains named commands for mmSendCommand(), a method of the MomanLib

The Enum is bigger and contains more Commands like NMT, but we won‘t need them in this example.

`eDecoded` is used by mmProtDecodeCmdStr() and mmProtDecodeAnswStr() as return

3. Then ”Create / Copy“ a static class „MomanWrapperLib“ inside the namespace (see Codelisting 3: Example for inserting the static class MomanWrapperLib)
4. ”Read“ and select your library you want to use. (Choose from the Protocol folder)
   Here is USB selected, although we don‘t want to use interface-specific functions.
   Create a final string in the class with the Path to the DLL as value
   `public const string cProtDll = @"C:\MyProject\FaulhaberMomanLibExampleProgram\lib\MomanLib\Lib\Bin\Protocol\USB\CO_USB.dll";`
   You may aswell use a relative Path, but it has to match your specific path.

Codelisting 3: Example for inserting the static class MomanWrapperLib

```
using System;

namespace FaulhaberMomanLibExampleProgram
{
    class MomanWrapperLib
    {
        public const string cProtDll =
@"C:\MyProject\FaulhaberMomanLibExampleProgram\lib\MomanLib\Lib\Bin\Protocol\USB\CO_USB.dll";
    }
    //Code from Codelisting 1 [public delegate void tdmmProtDataCallback(); …]
}
```

5. “Read“ and open the MomanProt.h because only the Protocol-Wrapping is needed here. See Documentation for more. If you scroll down, you may see the function prototypes like in Codelisting 4: Excerpt of function Prototypes

Codelisting 4: Excerpt of function Prototypes

```
/* Function prototypes */
MOMANPROT_API eMomanprot __stdcall mmProtInitInterface(char* InterfaceDll,
      tdmmProtDataCallback DataReceived, tdmmProtTraceValuesCallback
TraceValuesReceived);
MOMANPROT_API void __stdcall mmProtCloseInterface(void);
MOMANPROT_API void __stdcall mmProtSetDataCallback(tdmmProtDataCallback
DataReceived);
MOMANPROT_API void __stdcall mmProtSetTraceValuesCallback(
      tdmmProtTraceValuesCallback TraceValuesReceived);
MOMANPROT_API eMomanprot __stdcall mmProtOpenCom(int port, int channel, int
baud);
MOMANPROT_API void __stdcall mmProtCloseCom(void);
MOMANPROT_API int __stdcall mmProtLoadCommandSet(int cmdType);
```

If you need more functions like listed above, you will have to translate them on your own or see the examples.

6. ”Read“. Now the DLL-Imports are needed to bind the C#-Methods to the functions provided in the DLL. To bring it on point, it is exactly what is described in Chapter “What is a Wrapper”.
This is Done with

```
[DllImport(cProtDll, CallingConvention=CallingConvention.StdCall)]
```

“Copy“ for some of the DLL-Imports we need to declare the usage of:

```
using System.Runtime.InteropServices;
```

You will need to add this line at the very beginning of the MomanWrapperLib.cs file as seen in Codelisting 3: Example for inserting the static class MomanWrapperLib

7. The final result should look like in Figure 17: Example view of the WrapperLibrary, you can „Copy“ the Function Definitions from Codelisting 5: Function declarations of the Wrapper library:

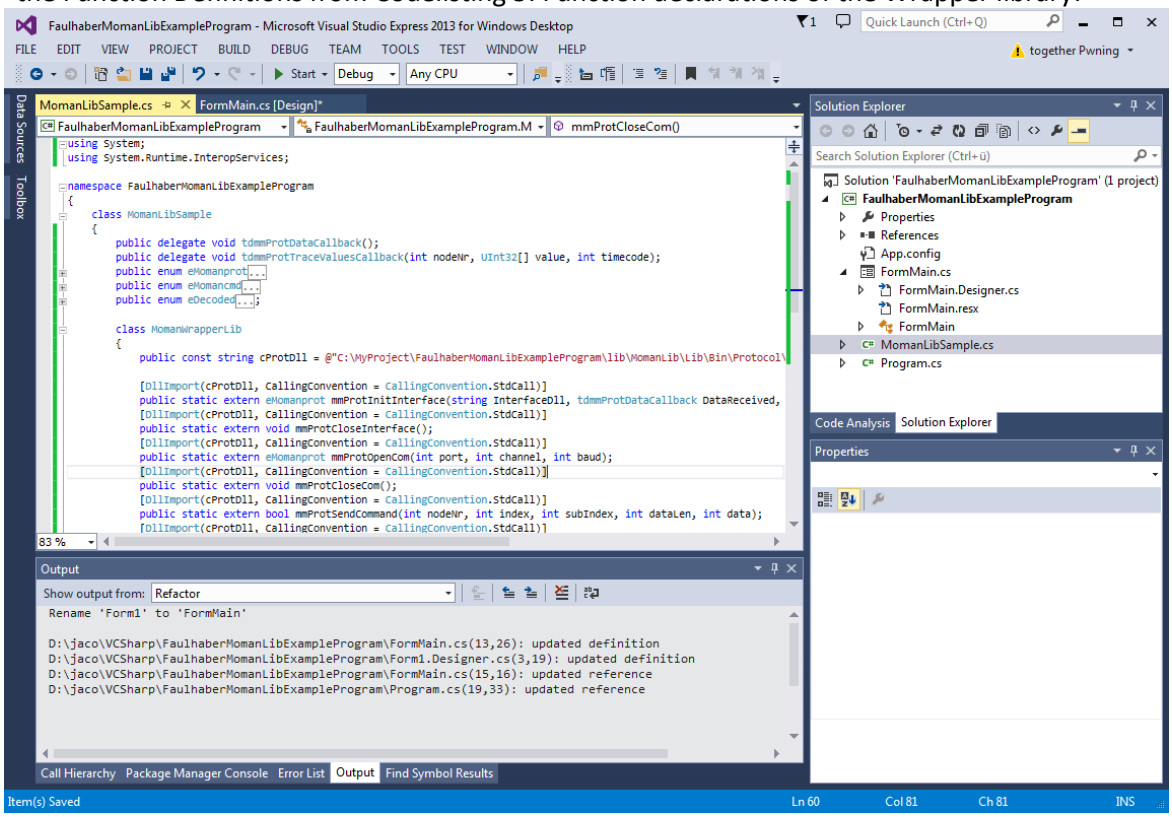


Figure 17: Example view of the WrapperLibrary

Codelisting 5: Function declarations of the Wrapper library

```
[DllImport(cProtDll, CallingConvention=CallingConvention.StdCall)]
public static extern eMomanprot mmProtInitInterface(string InterfaceDll, tdmmProtDataCallback
DataReceived, tdmmProtTraceValuesCallback TraceValuesReceived);
[DllImport(cProtDll, CallingConvention=CallingConvention.StdCall)]
public static extern void mmProtCloseInterface();
[DllImport(cProtDll, CallingConvention=CallingConvention.StdCall)]
public static extern eMomanprot mmProtOpenCom(int port, int channel, int baud);
[DllImport(cProtDll, CallingConvention=CallingConvention.StdCall)]
public static extern void mmProtCloseCom();
[DllImport(cProtDll, CallingConvention = CallingConvention.StdCall)]
public static extern bool mmProtSendCommand(int nodeNr, int index, int subIndex, int dataLen, int
data);
[DllImport(cProtDll, CallingConvention = CallingConvention.StdCall)]
public static extern eMomanprot mmProtReadAnswer(out IntPtr answData, out int nodeNr, out IntPtr
cmdString, out IntPtr receiveTelegram);
[DllImport(cProtDll, CallingConvention = CallingConvention.StdCall)]
public static extern eDecoded mmProtDecodeAnswStr([MarshalAs(UnmanagedType.LPStr)] string answStr,
out Int64 value);
[DllImport(cProtDll, CallingConvention = CallingConvention.StdCall)]
public static extern eMomanprot mmProtGetStrObj(int nodeNr, int index, int subIndex, out IntPtr
value);
[DllImport(cProtDll, CallingConvention = CallingConvention.StdCall)]
public static extern eMomanprot mmProtSetObj(int nodeNr, int index, int subIndex, int value, int
len, out uint abortCode);
[DllImport(cProtDll, CallingConvention = CallingConvention.StdCall)]
public static extern string mmProtGetAbortMessage(uint abortCode);
```

Providing an Interface to the User-Code

Why an Interface

In every application you want to separate user code / more generalizable code from specific code. The advantage is easily explained: If you want to either port the software to another platform or want to exchange some software layers. Let's say you want a new Graphical User Interface, build your own Library for the Communication with controllers or want to insert another library so that your application works with more Products, you need multiple layers. (for further reading try searching either online or in books with the keyword "Multitier architecture").

For this Application Note the Architecture with Examples from both #1 and #2 is recommended.

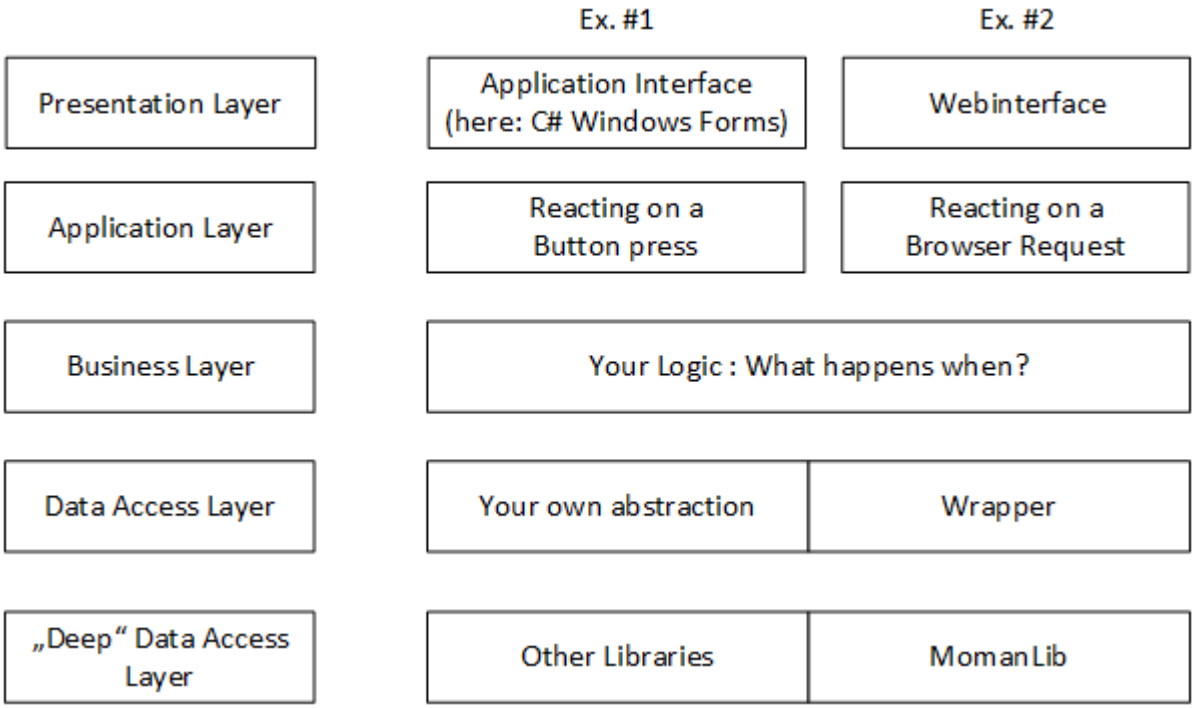


Figure 18: Architectural Overview and recommended Design

To be more specific, the Heading containing "Interface" does not refer to the C#-Way of an Interface, although it would be a professional step to translate the class definitions seen here into an interface.

What is left to do?

The following parts of the Tutorial will start with a Task that is like "Read", "Create", or "Copy".
"Read": Technically, providing an Interface would not be necessary. If you decide to use this in a bigger Project, it is strongly recommended though.

1. Ask yourself what you want to achieve
   Here: We want to:
   1. Get a Text from the Device
   2. Enable it; To be more precise: operate the CiA402
   3. Start a Positioning / Motor movement

   And not to mention the setup we need to do.
2. Then Look at the sentences and find an action and an object
   a. Get Text and Device
   b. Operate CiA402 and Device [it]
   c. Start Movement and Device
      [The Motor is replaced here with the Device because the device can do it for us]
3. Create a class for the Device and add the Methods

A sample Implementation can be found in the Examples of the Library.→
For the following we'll only need to *"copy".* the Codelisting 6: Example Code for the Interface to the Library Sample :

Codelisting 6: Example Code for the Interface to the Library Sample

```
class MomanLibSample
{
    //Used interface Dll from communication Library:
    const string cIntfDll =
@"C:\MyProject\FaulhaberMomanLibExampleProgram\lib\MomanLib\Lib\Bin\Interface\USB\MC3USB.dl
l";
    public MomanLibSample()
    {
        //check if we can find the dll
        if (!System.IO.File.Exists(MomanWrapperLib.cProtDll))
        {
            throw new DllNotFoundException();
        }
    }
    internal bool GetStrObj(int nodeNr, int index, int subIndex, out string value)
    {
        IntPtr answData;
        eMomanprot  ret  =  MomanWrapperLib.mmProtGetStrObj(nodeNr,  index,  subIndex,  out
answData);
        if (ret == eMomanprot.eMomanprot_ok)
        {
            value = Marshal.PtrToStringAnsi(answData);
            return true;
        }
        else
        {
```

```
            value = "<ERROR>";
            return false;
        }
    }

    internal bool Init(tdmmProtDataCallback SignalDataReceived)
    {
        if  (MomanWrapperLib.mmProtInitInterface(cIntfDll,  SignalDataReceived,  null)  !=
eMomanprot.eMomanprot_ok)
        {
            return false;
        }
        if (MomanWrapperLib.mmProtOpenCom(1, 0, 0) != eMomanprot.eMomanprot_ok)
        {
            return false;
        }
        return true;
    }
}
```

Adding Functionality to the User Interface

Usually one would want to log the actions / data that the program is producing. You could use a commonly used Library like log4net or similar. We will use the simpler Way and therefore we create a Logging-Method, that you can *"copy".*

Codelisting 7: Code snippet for logging in a Textbox

```
delegate void LoggingMethod(string logMessage, DateTime date);

private void Log(string logMessage)
{
    LogData_Threadsafe(logMessage, DateTime.Now);
}

private void LogData_Threadsafe(string logMessage, DateTime date)
{
    if(textBox1.InvokeRequired)
    {
        IAsyncResult           reference           =           textBox1.BeginInvoke(new
LoggingMethod(LogData_Threadsafe),logMessage,date);
        textBox1.EndInvoke(reference);
    }
    else
    {
        string dateStr = "??";
        //use ISO 8601 for date
        if(date != null)
            dateStr = date.ToString("yyyy-MM-dd\\THH:mm:ss");
        textBox1.Text += string.Format("[{0}] {1}\r\n",dateStr,logMessage);
    }
}
```

We then quickly add an initialization method by using the User Interface of Visual Studio and *"select".*Properties → Event → Load event, then *"create".*the Method by double-clicking on the dropdown as seen in the red box in Figure 19: Adding the Load Event .

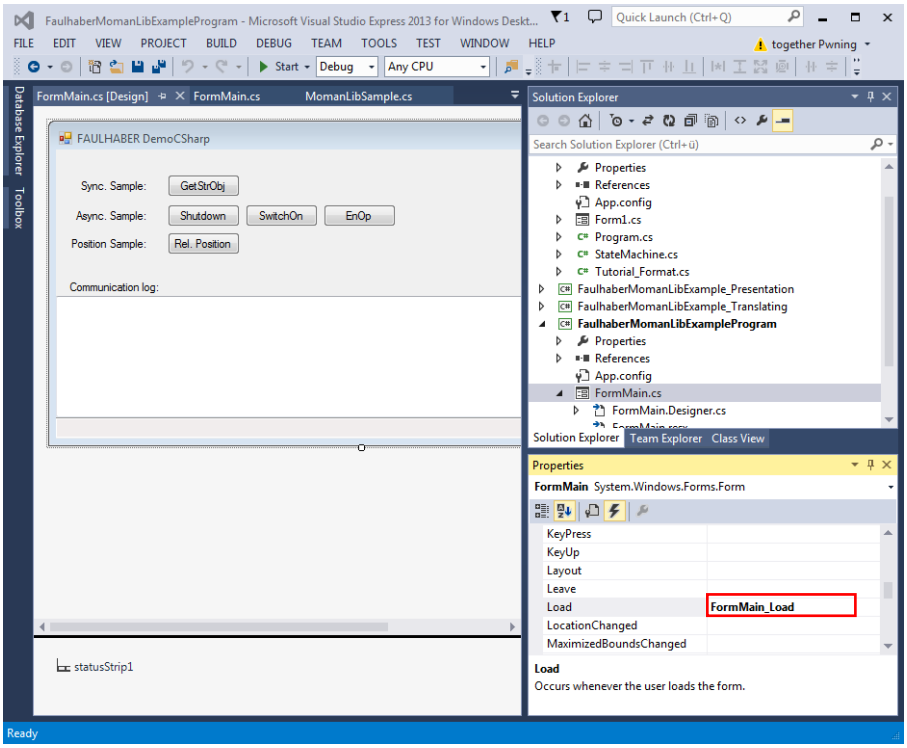


Figure 19: Adding the Load Event

You may "copy" the following Codelisting 8: Initialization of the Library inside the Library

Codelisting 8: Initialization of the Library

```
MomanLibSample library;
System.Threading.Thread ReceiveThread;
private void FormMain_Load(object sender, EventArgs e)
{
    library = new MomanLibSample();
    try
    {
        if (library.Init(CBAsyncDataReceived) == false)
            throw new Exception("Init failed!");
        ReceiveThread = new System.Threading.Thread(EventReceiver);
        ReceiveThread.Start();
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message);
        //we got an error
        this.Close();
    }
}
```

Assuming the Initialization can fail, we surround it in a try-catch-block and show a Message Box when an error is thrown. Logging here wouldn't be good if we used the textbox example like before.

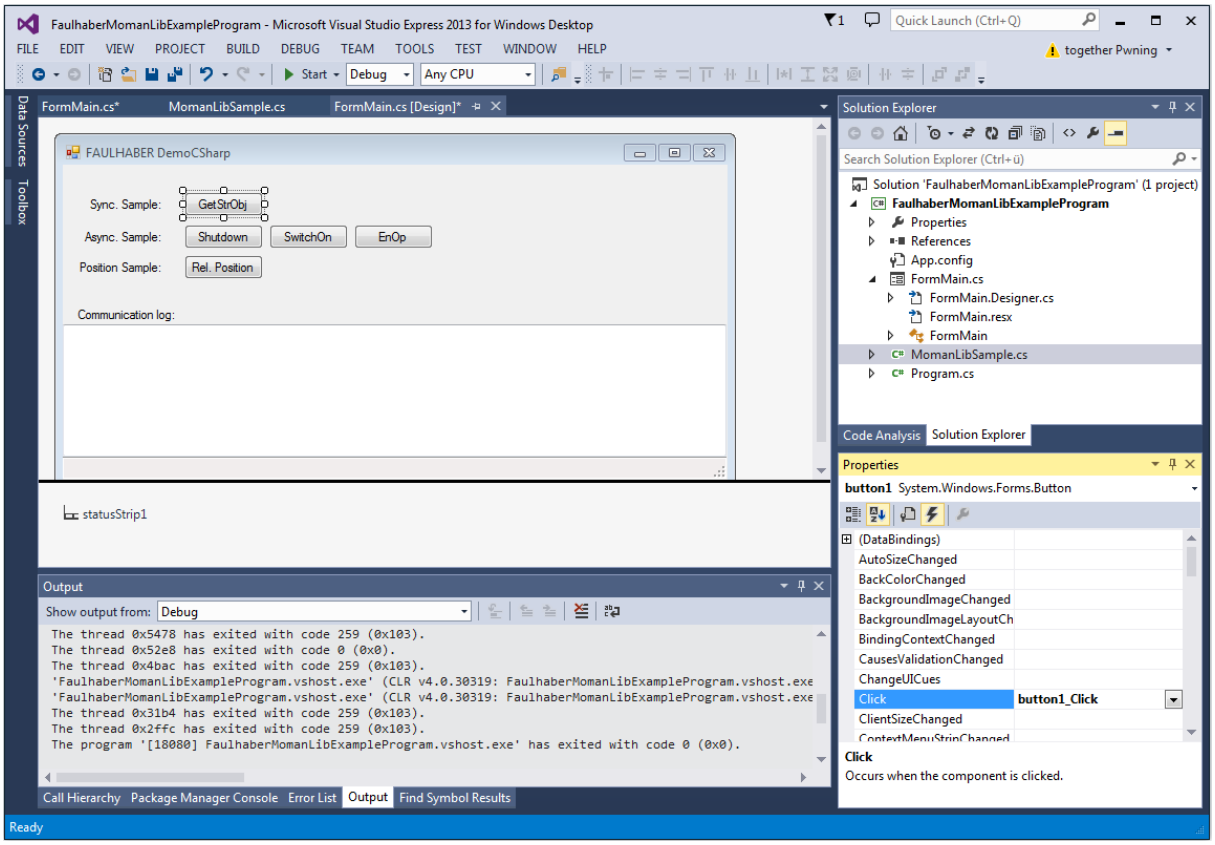


Figure 20: How to add a on_Click method to a button

Now we can "add" a function to the Button-Click by either double-clicking the Button or adding a method in the Properties → Events → Click event.

Codelisting 9: Source code for the GetStrObj-Button

```
private void button1_Click(object sender, EventArgs e)
{
    string value = "";
    library.GetStrObj(1, 0x1008, 0x00, out value);
    Log("Sync read 0x1008.00");
    Log(string.Format("Sync received: {0}",value));
}
```

This would result in something like Figure 21: The finished GUI (Graphical User Interface):

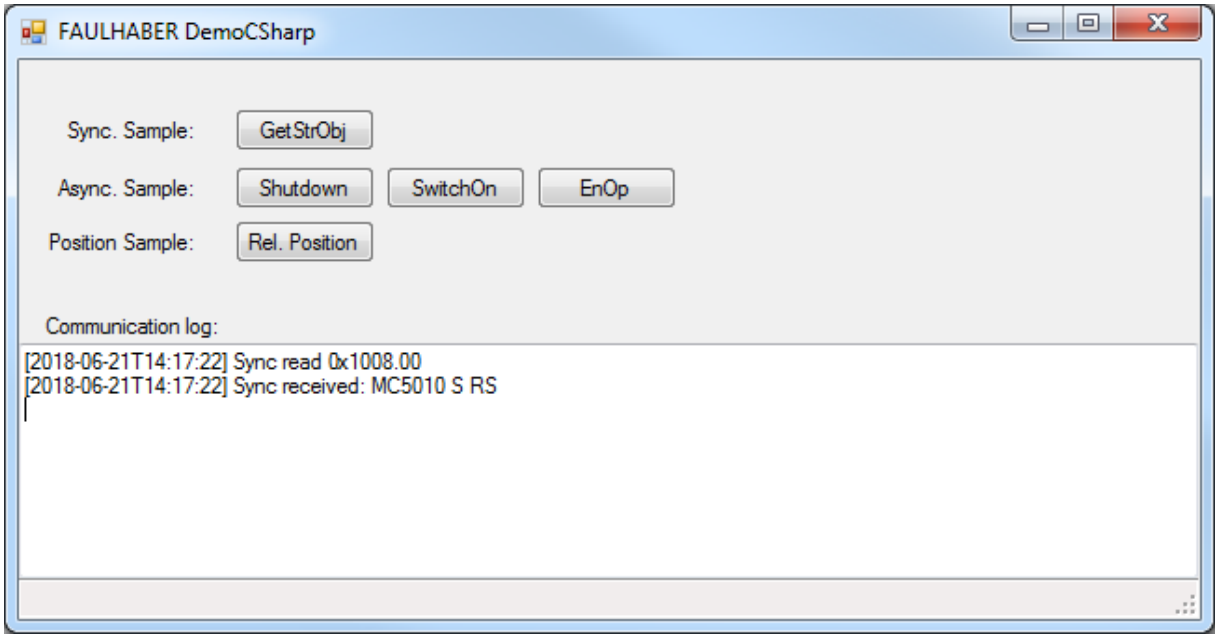


Figure 21: The finished GUI (Graphical User Interface)

Synchronous Access

You can directly call the library methods described in the Library Reference. You can find a copy in C:\MyProject\FaulhaberMomanLibExampleProgram\lib\MomanLib\Doc

Asynchronous Access

If you want to use the asynchronous Features of the Library you can create a Thread to receive a Notification when data is arrived. It is strongly advised that the Callback function called by the library does not directly process the data, because it will then block the receiving part of the library. One possible solution is the usage of System.Threading.AutoResetEvent.

Codelisting 10: Example for asynchronous access handling

```
private System.Threading.AutoResetEvent ReceiveEvent = new
System.Threading.AutoResetEvent(false);
private void CBAsyncDataReceived()
{
    ReceiveEvent.Set();
}
private void EventReceiver()
{
    while (true)
    {
        ReceiveEvent.WaitOne();
        Invoke(new MethodInvoker(AsyncDataReceived));
        ReceiveEvent.Reset();
    }
} private void AsyncDataReceived() { Log("Async Data!");/*we may call
ReadReceivedData*/ }
```

Putting it together

As this puzzling and copy-pasting with source code parts always gets confusing – the two main parts are completed here, as stated in the beginning of the chapter.
The following files are presented:

Filename	Description
FormMain.cs	The File contains: - Functionality to the User Interface
MomanLibSample.cs	The File contains: - The Copied Parts from Codelisting 2: Definition of the enums used in the library - The Class `class MomanLibSample` - The Class `class MomanWrapperLib`
Program.cs	The File contains the main function with … `Application.Run(new FormMain());`

Hint: You might have to adjust the namespaces.

FormMain.cs

Codelisting 11: Final source code of the FormMain.cs

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

namespace FaulhaberMomanLibExampleProgram
{
    public partial class FormMain : Form
    {
        MomanLibSample library;
        public FormMain()
        {
            InitializeComponent();
        }
```

```
        private void button1_Click(object sender, EventArgs e)
        {
            string value = "";
            library.GetStrObj(1, 0x1008, 0x00, out value);
            Log("Sync read 0x1008.00");
            Log(string.Format("Sync received: {0}",value));
        }

        delegate void LoggingMethod(string logMessage, DateTime date);

        private void Log(string logMessage)
        {
            LogData_Threadsafe(logMessage, DateTime.Now);
        }

        private void LogData_Threadsafe(string logMessage, DateTime date)
        {
            if(textBox1.InvokeRequired)
            {
                IAsyncResult reference = textBox1.BeginInvoke(new
LoggingMethod(LogData_Threadsafe),logMessage,date);
                textBox1.EndInvoke(reference);
            }
            else
            {
                string dateStr = "??";
                //use ISO 8601 for date
                if(date != null)
                    dateStr = date.ToString("yyyy-MM-dd\\THH:mm:ss");
                textBox1.Text += string.Format("[{0}] {1}\r\n",dateStr,logMessage);
            }
        }

        private System.Threading.AutoResetEvent ReceiveEvent = new
System.Threading.AutoResetEvent(false);
        private void CBAsyncDataReceived()
        {
            ReceiveEvent.Set();
        }
        private void EventReceiver()
        {
            while (true)
            {
                ReceiveEvent.WaitOne();
                Invoke(new MethodInvoker(AsyncDataReceived));
                ReceiveEvent.Reset();
            }
        }
```

```
        private void AsyncDataReceived()
        {
            Log("Async Data!");//we may call ReadReceivedData
        }

        System.Threading.Thread ReceiveThread;
        private void FormMain_Load(object sender, EventArgs e)
        {
            library = new MomanLibSample();
            try
            {
                if (library.Init(CBAsyncDataReceived) == false)
                    throw new Exception("Init failed!");
                ReceiveThread = new System.Threading.Thread(EventReceiver);
                ReceiveThread.Start();
            }
            catch(Exception ex)
            {
                MessageBox.Show(ex.Message);
                //we got an error
                this.Close();
            }
        }

        private void FormMain_FormClosed(object sender, FormClosedEventArgs e)
        {
            ReceiveThread.Abort();
            ReceiveThread.Join();
        }
    }
}
```

MomanLibSample.cs

Codelisting 12: Final source code of the MomanLibSample.cs

```
using System;
using System.Runtime.InteropServices;

namespace FaulhaberMomanLibExampleProgram
{

    public delegate void tdmmProtDataCallback();
    public delegate void tdmmProtTraceValuesCallback(int nodeNr, UInt32[] value, int
timecode);
    public enum eMomanprot
    {
        eMomanprot_ok_bootup = 2,
        eMomanprot_ok_async = 1,
        eMomanprot_ok = 0,
        eMomanprot_error = -1,
        eMomanprot_error_timeout = -2,
        eMomanprot_error_cmd = -3,
        eMomanprot_error_emcy = -4,
        eMomanprot_error_param = -5,
        eMomanprot_error_accessdenied = -6,
        eMomanprot_error_init = -7,
        eMomanprot_noData = -8
    }
    public enum eMomancmd
    {
        //Device Control:
        eMomancmd_shutdown = 16,
        eMomancmd_switchon = 17,
        eMomancmd_disable = 18,
        eMomancmd_quickstop = 19,
        eMomancmd_DiOp = 20,
        eMomancmd_EnOp = 21,
        eMomancmd_faultreset = 22,
        eMomancmd_MA = 23,
        eMomancmd_MR = 24,
        eMomancmd_HS = 25
    }
    public enum eDecoded
    {
        eDecoded_none = 0,              /*!< none decoded */
        eDecoded_SOBJ = 1,              /*!< SOBJ command decoded */
        eDecoded_GOBJ = 2,              /*!< GOBJ command decoded*/
        eDecoded_Bootup = 3,
        eDecoded_NMT = 4,
        eDecoded_NMTRequest = 5,
        eDecoded_Heartbeat = 6,
```

```
        eDecoded_Statusword = 7,
    };

    class MomanWrapperLib
    {
        public const string cProtDll =
@"C:\MyProject\FaulhaberMomanLibExampleProgram\lib\MomanLib\Lib\Bin\Protocol\USB\CO_USB.dll
";

        [DllImport(cProtDll, CallingConvention = CallingConvention.StdCall)]
        public static extern eMomanprot mmProtInitInterface(string InterfaceDll,
tdmmProtDataCallback DataReceived, tdmmProtTraceValuesCallback TraceValuesReceived);
        [DllImport(cProtDll, CallingConvention = CallingConvention.StdCall)]
        public static extern void mmProtCloseInterface();
        [DllImport(cProtDll, CallingConvention = CallingConvention.StdCall)]
        public static extern eMomanprot mmProtOpenCom(int port, int channel, int baud);
        [DllImport(cProtDll, CallingConvention = CallingConvention.StdCall)]
        public static extern void mmProtCloseCom();
        [DllImport(cProtDll, CallingConvention = CallingConvention.StdCall)]
        public static extern bool mmProtSendCommand(int nodeNr, int index, int subIndex,
int dataLen, int data);
        [DllImport(cProtDll, CallingConvention = CallingConvention.StdCall)]
        public static extern eMomanprot mmProtReadAnswer(out IntPtr answData, out int
nodeNr, out IntPtr cmdString, out IntPtr receiveTelegram);
        [DllImport(cProtDll, CallingConvention = CallingConvention.StdCall)]
        public static extern eDecoded mmProtDecodeAnswStr([MarshalAs(UnmanagedType.LPStr)]
string answStr, out Int64 value);
        [DllImport(cProtDll, CallingConvention = CallingConvention.StdCall)]
        public static extern eMomanprot mmProtGetStrObj(int nodeNr, int index, int
subIndex, out IntPtr value);
        [DllImport(cProtDll, CallingConvention = CallingConvention.StdCall)]
        public static extern eMomanprot mmProtSetObj(int nodeNr, int index, int subIndex,
int value, int len, out uint abortCode);
        [DllImport(cProtDll, CallingConvention = CallingConvention.StdCall)]
        public static extern string mmProtGetAbortMessage(uint abortCode);
    }

    class MomanLibSample
    {
        //Used interface Dll from communication Library:
        const string cIntfDll =
@"C:\MyProject\FaulhaberMomanLibExampleProgram\lib\MomanLib\Lib\Bin\Interface\USB\MC3USB.dl
l";
        public MomanLibSample()
        {
            //check if we can find the dll
            if (!System.IO.File.Exists(MomanWrapperLib.cProtDll))
            {
                throw new DllNotFoundException();
            }
        }
```

```
        internal bool GetStrObj(int nodeNr, int index, int subIndex, out string value)
        {
            IntPtr answData;
            eMomanprot ret = MomanWrapperLib.mmProtGetStrObj(nodeNr, index, subIndex, out
answData);
            if (ret == eMomanprot.eMomanprot_ok)
            {
                value = Marshal.PtrToStringAnsi(answData);
                return true;
            }
            else
            {
                value = "<ERROR>";
                return false;
            }
        }

        internal bool Init(tdmmProtDataCallback SignalDataReceived)
        {
            if (MomanWrapperLib.mmProtInitInterface(cIntfDll, SignalDataReceived, null) !=
eMomanprot.eMomanprot_ok)
            {
                return false;
            }
            if (MomanWrapperLib.mmProtOpenCom(1, 0, 0) != eMomanprot.eMomanprot_ok)
            {
                return false;
            }
            return true;
        }
    }
}
```

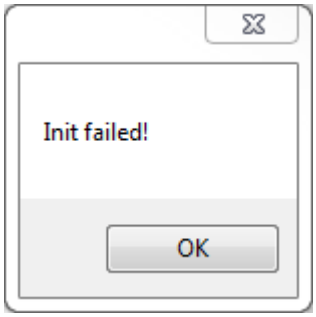
Testing

The Example provided above should produce a closely comparable result to the example project delivered with the Library. Whilst using the Form, it should produce a result described in the following list, but some requirements have to be matched:

- Your device has to have the right voltage connected to the right terminals – check first
- Your device should have the communication Port connected to your PC / Device.
- Your device should be discoverable in the latest Version Motion manager. (Here: Motion Manager 6.4 is used)
- Make sure that the Node-Id of the device you are using is the one you are seeing in the motion manager – or the example (Fieldname: cNodeNr). If you created the example yourself, you may edit an argument of the call
- Make sure you have the right DLL-Paths for both the Interface- DLL and the Protocol-DLL.

```
library.GetStrObj(1, 0x1008, 0x00, out value);
```

Referring to Figure 22: The Demo at use, with an MC V3.0 (MC5010 S CO) over USB with a configured and connected Motor



- When Starting:
  - No Error should be noticeable, the Demo shows a string

```
FAULHABER communication API loaded
```

  - If the Device is shown in Motion Manager, but it still shows then you need to close the Motion Manager

- When using the Interface Element
  1. Pressing the Button 1:
     a. The Program Should output something like

```
[2018-10-05T12:00:00] Sync read 0x1008.00
[2018-10-05T12:00:00] Sync received: MC5010
```

     b. If it does not, but something like

```
[2018-10-05T12:00:00] Sync read 0x1008.00
[2018-10-05T12:00:00] Sync received: <ERROR>
```

  2. Pressing the Buttons 2, 3 or 4
     a. These Buttons should activate the State machine of the Controller, visible by the blinking speed of the „Status LED“ (Refer to the Device Manual) as well as a text in the Communication log (7), one of

```
[2018-10-05T12:00:00] Send SHUTDOWN
[2018-10-05T12:00:00] Send SWITCHON
[2018-10-05T12:00:00] Send ENOP
```

  3. Pressing the Button 5
     a. This should to nothing in this Demo, but the example should behave with a text like below as well as a relative position by 1000 Increments.

```
[2018-10-05T12:00:00] Execute MOVE RELATIVE
```

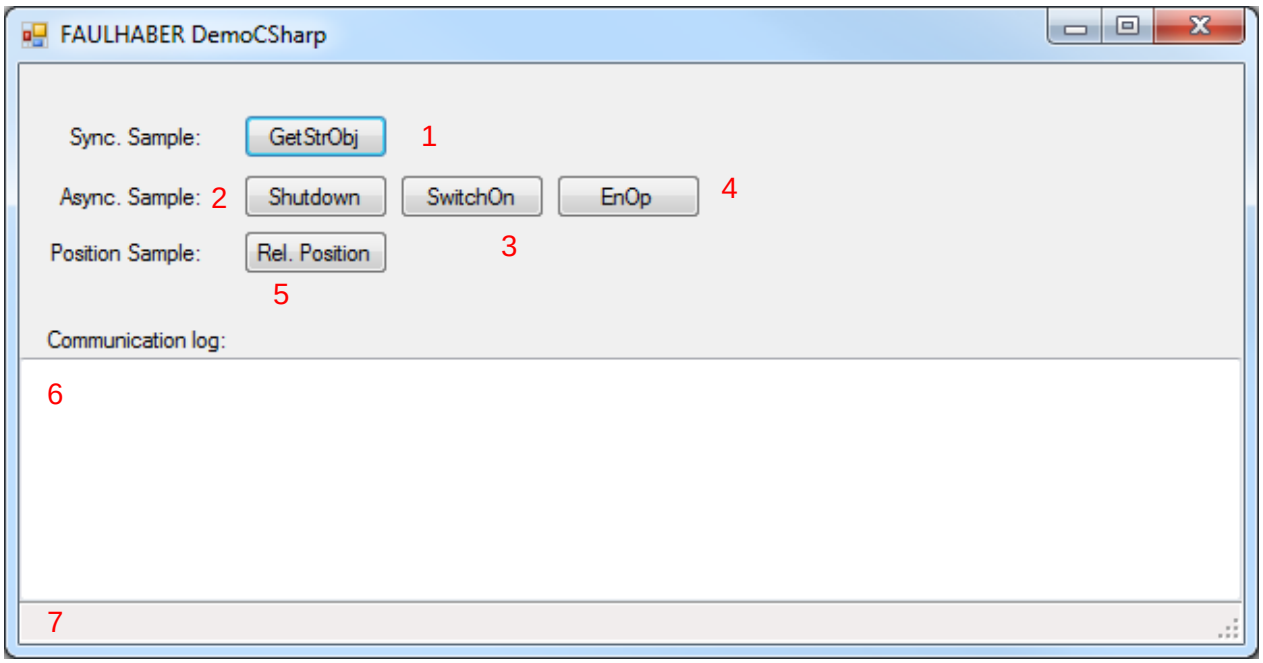


Figure 22: The Demo at use

Table of Figures

Table of Source Code

Rechtliche Hinweise

Urheberrechte. Alle Rechte vorbehalten. Ohne vorherige ausdrückliche schriftliche Zustimmung der Dr. Fritz Faulhaber & Co. KG darf diese Application Note oder Teile dieser unabhängig von dem Zweck insbesondere nicht vervielfältigt, reproduziert, gespeichert (z.B. in einem Informationssystem) oder be- oder verarbeitet werden.
Gewerbliche Schutzrechte. Mit der Veröffentlichung, Übergabe/Übersendung oder sonstigen Zur-Verfügung-Stellung dieser Application Note werden weder ausdrücklich noch konkludent Rechte an gewerblichen Schutzrechten, übertragen noch Nutzungsrechte oder sonstige Rechte an diesen eingeräumt. Dies gilt insbesondere für gewerbliche Schutzrechte, die mittelbar oder unmittelbar den beschriebenen Anwendungen und/oder Funktionen dieser Application Note zugrunde liegen oder mit diesen in Zusammenhang stehen.
Kein Vertragsbestandteil; Unverbindlichkeit der Application Note. Die Application Note ist nicht Vertragsbestandteil von Verträgen, die die Dr. Fritz Faulhaber GmbH & Co. KG abschließt, und der Inhalt der Application Note stellt auch keine Beschaffenheitsangabe für Vertragsprodukte dar, soweit in den jeweiligen Verträgen nicht ausdrücklich etwas anderes vereinbart ist. Die Application Note beschreibt unverbindlich ein mögliches Anwendungsbeispiel. Die Dr. Fritz Faulhaber GmbH & Co. KG übernimmt insbesondere keine Gewährleistung oder Garantie dafür und steht auch insbesondere nicht dafür ein, dass die in der Application Note illustrierten Abläufe und Funktionen stets wie beschrieben aus- und durchgeführt werden können und dass die in der Application Note beschriebenen Abläufe und Funktionen in anderen Zusammenhängen und Umgebungen ohne zusätzliche Tests oder Modifikationen mit demselben Ergebnis umgesetzt werden können. Der Kunde und ein sonstiger Anwender müssen sich jeweils im Einzelfall vor Vertragsabschluss informieren, ob die Abläufe und Funktionen in ihrem Bereich anwendbar und umsetzbar sind.
Keine Haftung. Die Dr. Fritz Faulhaber GmbH & Co. KG weist darauf hin, dass aufgrund der Unverbindlichkeit der Application Note keine Haftung für Schäden übernommen wird, die auf die Application Note und deren Anwendung durch den Kunden oder sonstigen Anwender zurückgehen. Insbesondere können aus dieser Application Note und deren Anwendung keine Ansprüche aufgrund von Verletzungen von Schutzrechten Dritter, aufgrund von Mängeln oder sonstigen Problemen gegenüber der Dr. Fritz Faulhaber GmbH & Co. KG hergeleitet werden.
Änderungen der Application Note. Änderungen der Application Note sind vorbehalten. Die jeweils aktuelle Version dieser Application Note erhalten Sie von Dr. Fritz Faulhaber GmbH & Co. KG unter der Telefonnummer +49 7031 638 345 oder per Mail von mcsupport@faulhaber.de.

Legal notices

Copyrights. All rights reserved. This Application Note and parts thereof may in particular not be copied, reproduced, saved (e.g. in an information system), altered or processed in any way irrespective of the purpose without the express prior written consent of Dr. Fritz Faulhaber & Co. KG.
Industrial property rights. In publishing, handing over/dispatching or otherwise making available this Application Note Dr. Fritz Faulhaber & Co. KG does not expressly or implicitly grant any rights in industrial property rights nor does it transfer rights of use or other rights in such industrial property rights. This applies in particular to industrial property rights on which the applications and/or functions of this Application Note are directly or indirectly based or with which they are connected.
No part of contract; non-binding character of the Application Note. The Application Note is not a constituent part of contracts concluded by Dr. Fritz Faulhaber & Co. KG and the content of the Application Note does not constitute any contractual quality statement for products, unless expressly set out otherwise in the respective contracts. The Application Note is a non-binding description of a possible application. In

particular Dr. Fritz Faulhaber & Co. KG does not warrant or guarantee and also makes no representation that the processes and functions illustrated in the Application Note can always be executed and implemented as described and that they can be used in other contexts and environments with the same result without additional tests or modifications. The customer and any user must inform themselves in each case before concluding a contract concerning a product whether the processes and functions are applicable and can be implemented in their scope and environment.

No liability. Owing to the non-binding character of the Application Note Dr. Fritz Faulhaber & Co. KG will not accept any liability for losses arising from its application by customers and other users. In particular, this Application Note and its use cannot give rise to any claims based on infringements of industrial property rights of third parties, due to defects or other problems as against Dr. Fritz Faulhaber GmbH & Co. KG.

Amendments to the Application Note. Dr. Fritz Faulhaber & Co. KG reserves the right to amend Application Notes. The current version of this Application Note may be obtained from Dr. Fritz Faulhaber & Co. KG by calling +49 7031 638 345 or sending an e-mail to mcsupport@faulhaber.de.