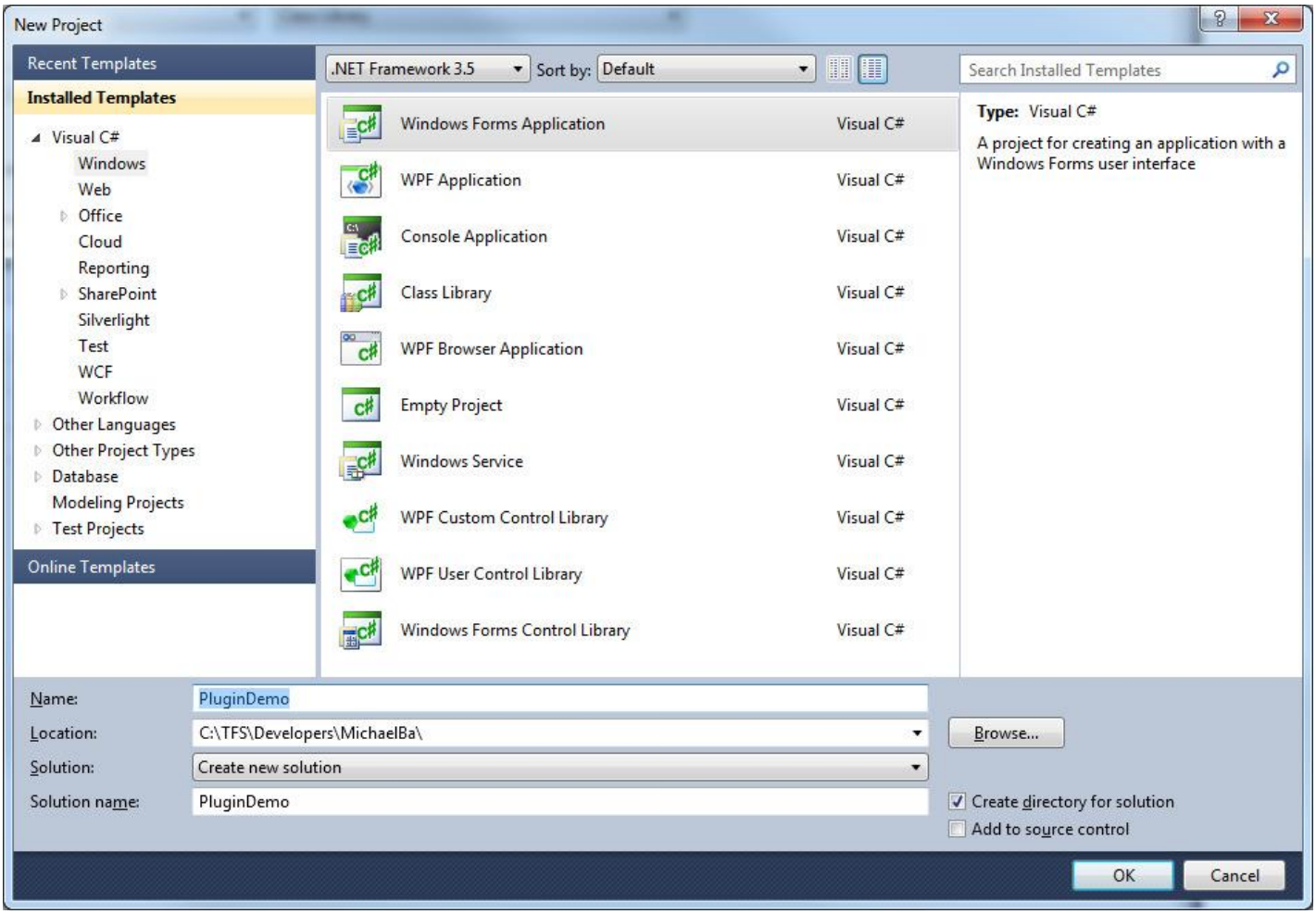
Plugin development for the Beckhoff HMI

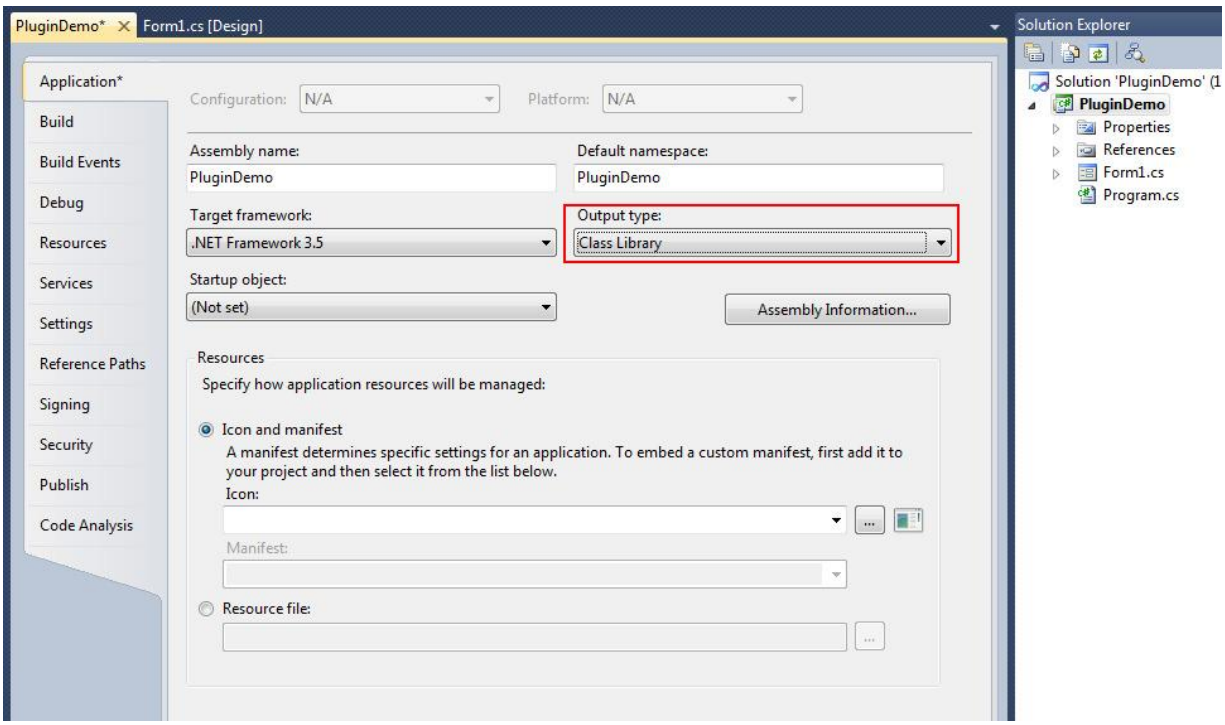
Michael Balsfulland
CNC Applikation
Telefon: + 49 (0)5246 963 236
Telefax: + 49 (0)5246 963 9 236
m.balsfulland@beckhoff.com

23.2.2012 **V 1.1**

In Visual Studio start a new project of type „Windows Forms Application“:

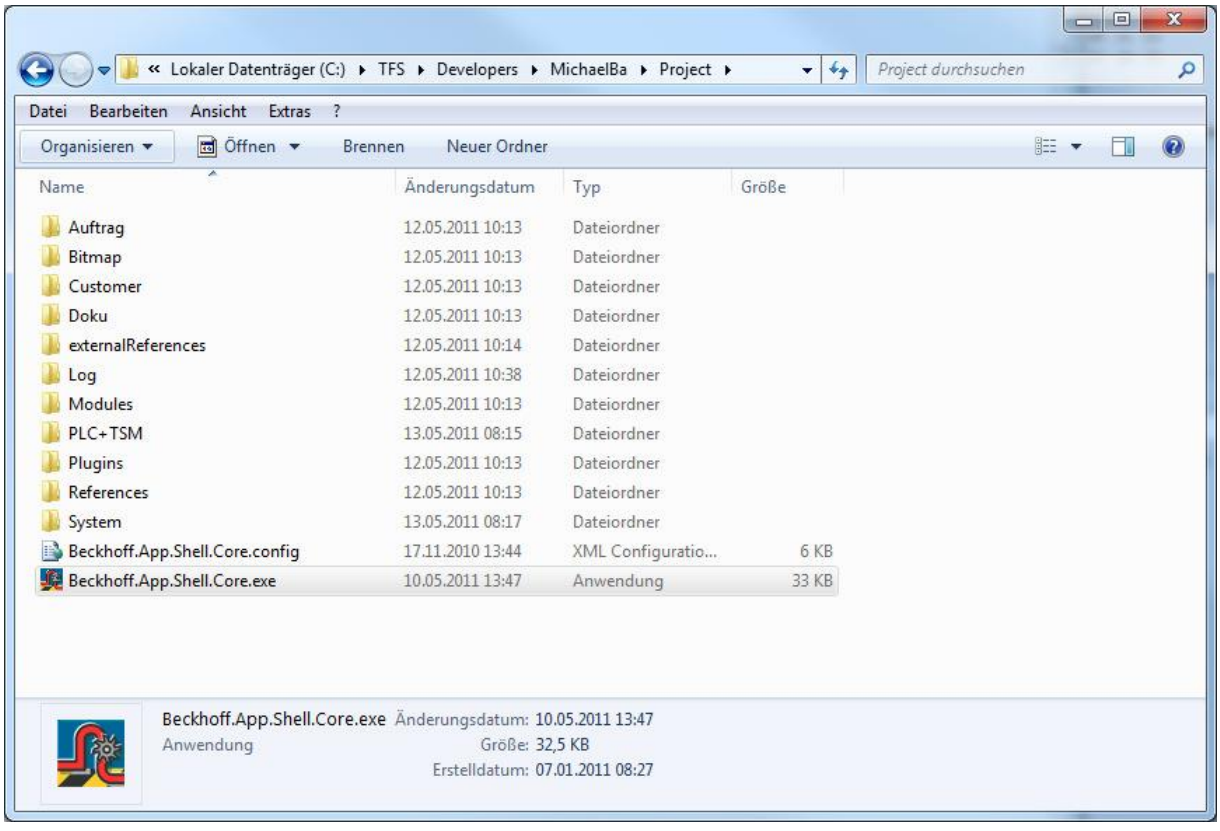


Then in the project properties, set the "Output type" to “Class library”:

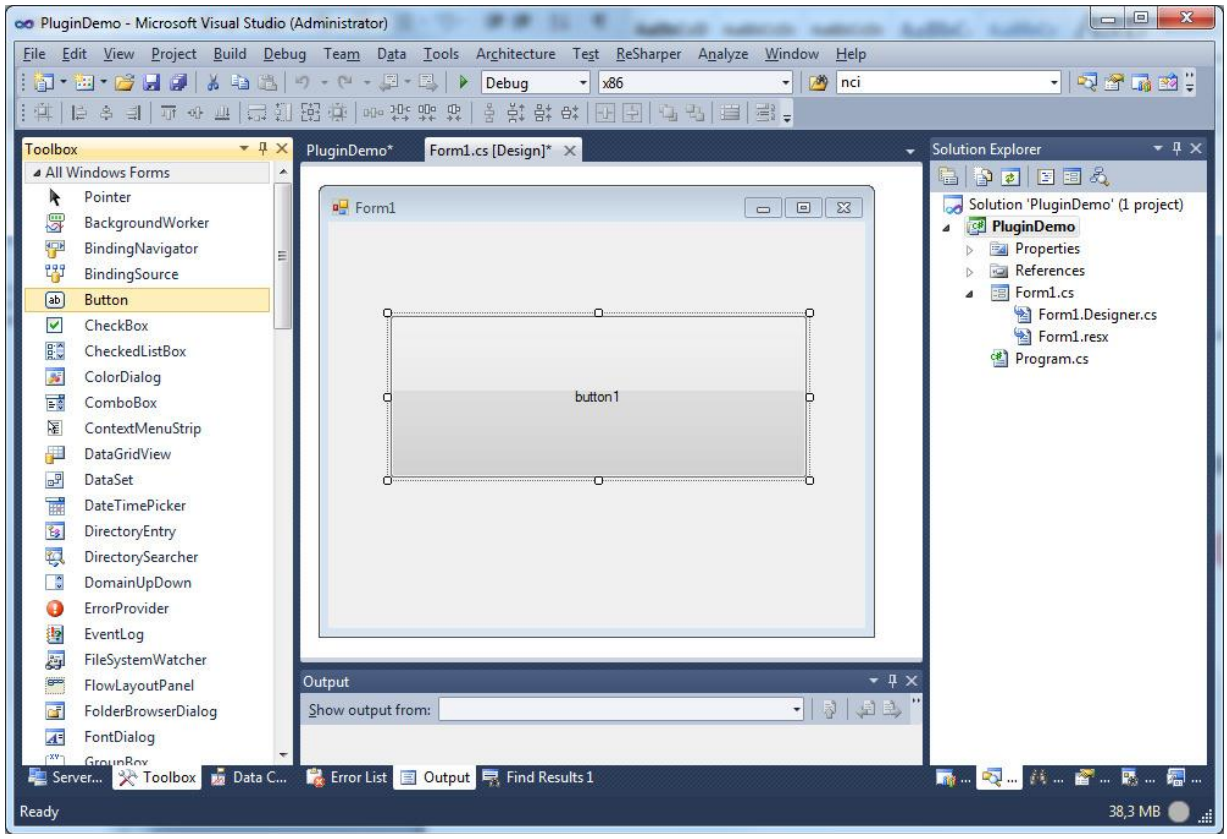


In this example, the following paths are set:

- The HMI executable is located under „C:\TFS\Developer\MichaelBa\Project“
- The project path to the plugin: „C:\TFS\Developer\MichaelBa\PluginDemo“



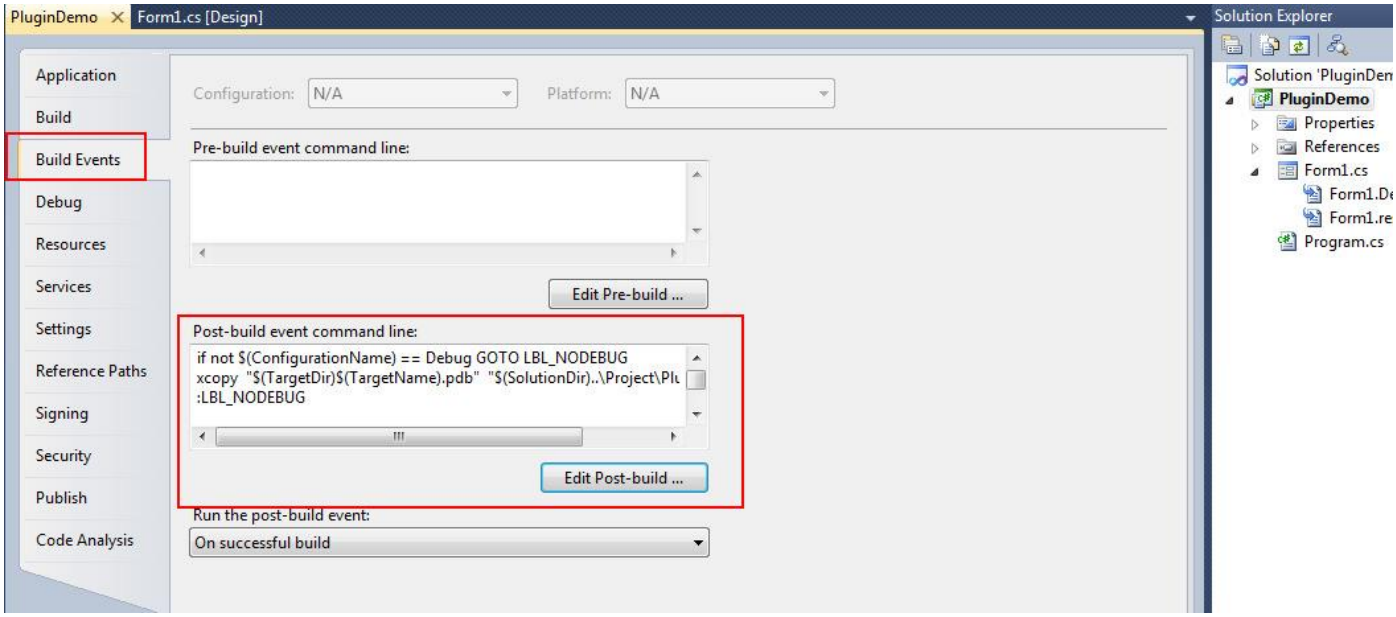
In order to test, drag a button to the form:



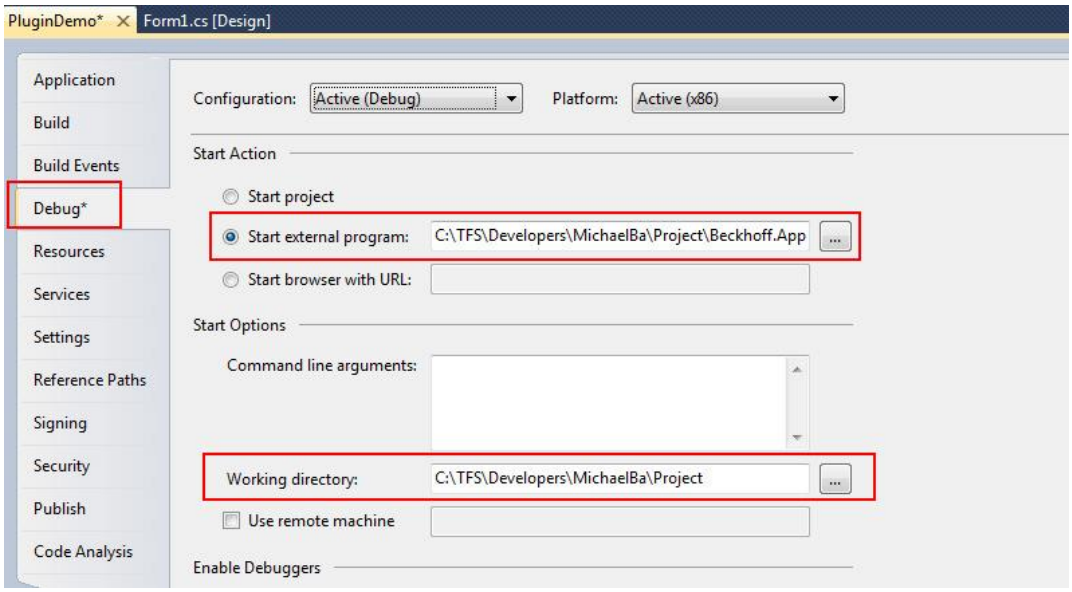
So that the PluginDemo can be launched in the debugger, the DLL must be copied in to the proper folder with a "PostbuildCommand" and the "Beckhoff.App.Shell.Exe" should be started from within Visual Studio. The "Form1" from PluginDemo.dll must be loaded in to the Menumanger and called in the HMI. To do this, the following steps are necessary:

- Enter the following in the properties of the project in the "build events" in the "post-built event command line"

```
if not $(ConfigurationName) == Debug GOTO LBL_NODEBUG
xcopy  "$(TargetDir)$(TargetName).pdb"  "$(SolutionDir)..\Project\Plugins" /r/y/c/d
:LBL_NODEBUG
rem copies build into local assembly cache
xcopy  "$(TargetDir)$(TargetFileName)"   "$(SolutionDir)..\Project\Plugins" /r/y/c/d
```



- In “debug” enter the appropriate working path for "Beckhoff.App.Shell.Core.Exe":



In the free Express version of Visual Studio, there is no convenient way to set an external program to debug.

This setting can be worn however "by hand" with a text editor in the .csproj file of the project.

The red lines are to complement and to adapt:

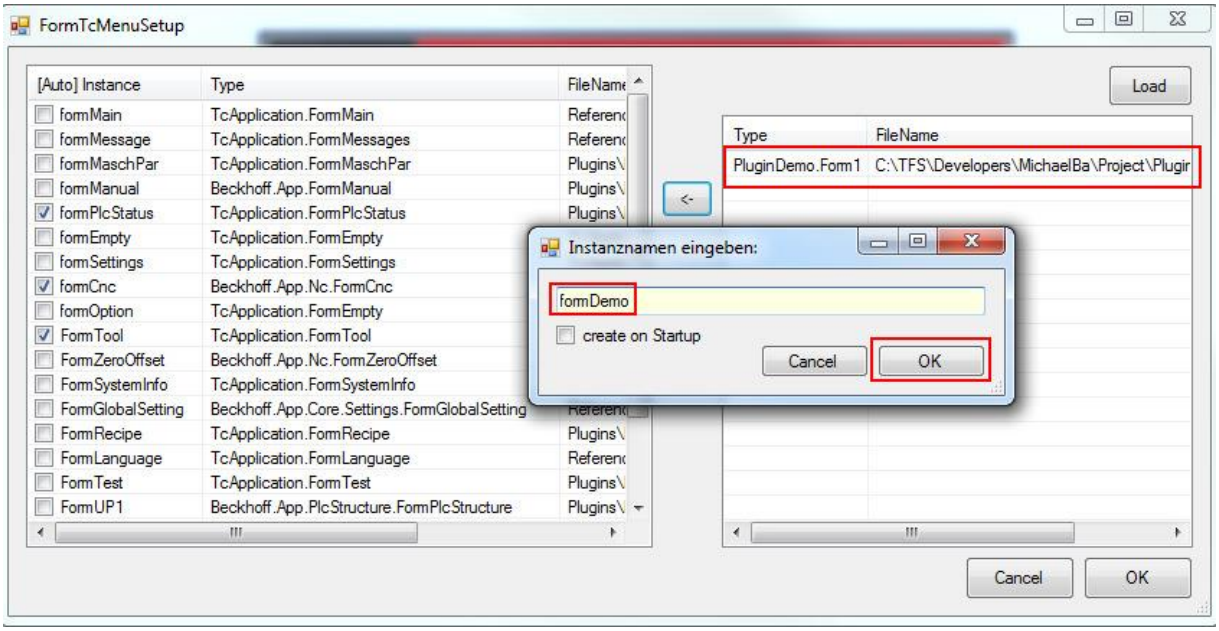
Excerpt from the .csproj file:

```
<PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|x86' ">
    <PlatformTarget>x86</PlatformTarget>
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
    <StartAction>Program</StartAction>
    <StartProgram>C:\HMI 2.3\Beckhoff.App.Shell.Core.exe</StartProgram>
    <StartWorkingDirectory>C:\HMI 2.3</StartWorkingDirectory>
</PropertyGroup>
```

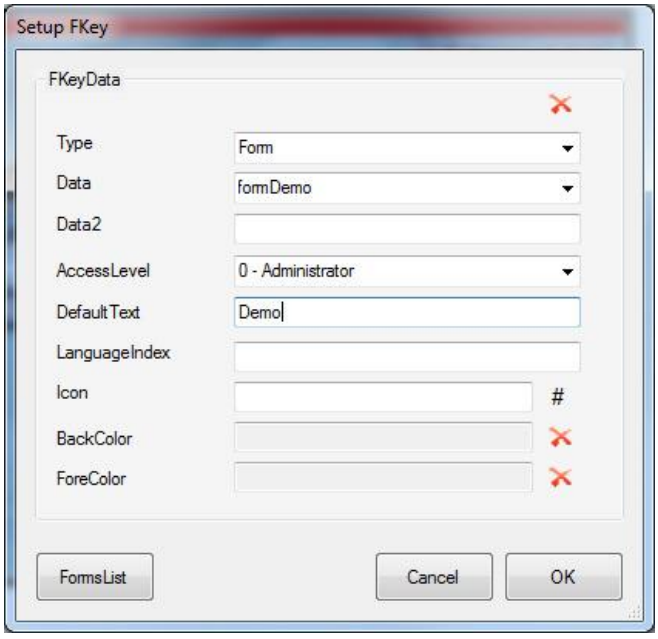
<F5>'Start Debugging' can now be used to start the HMI.

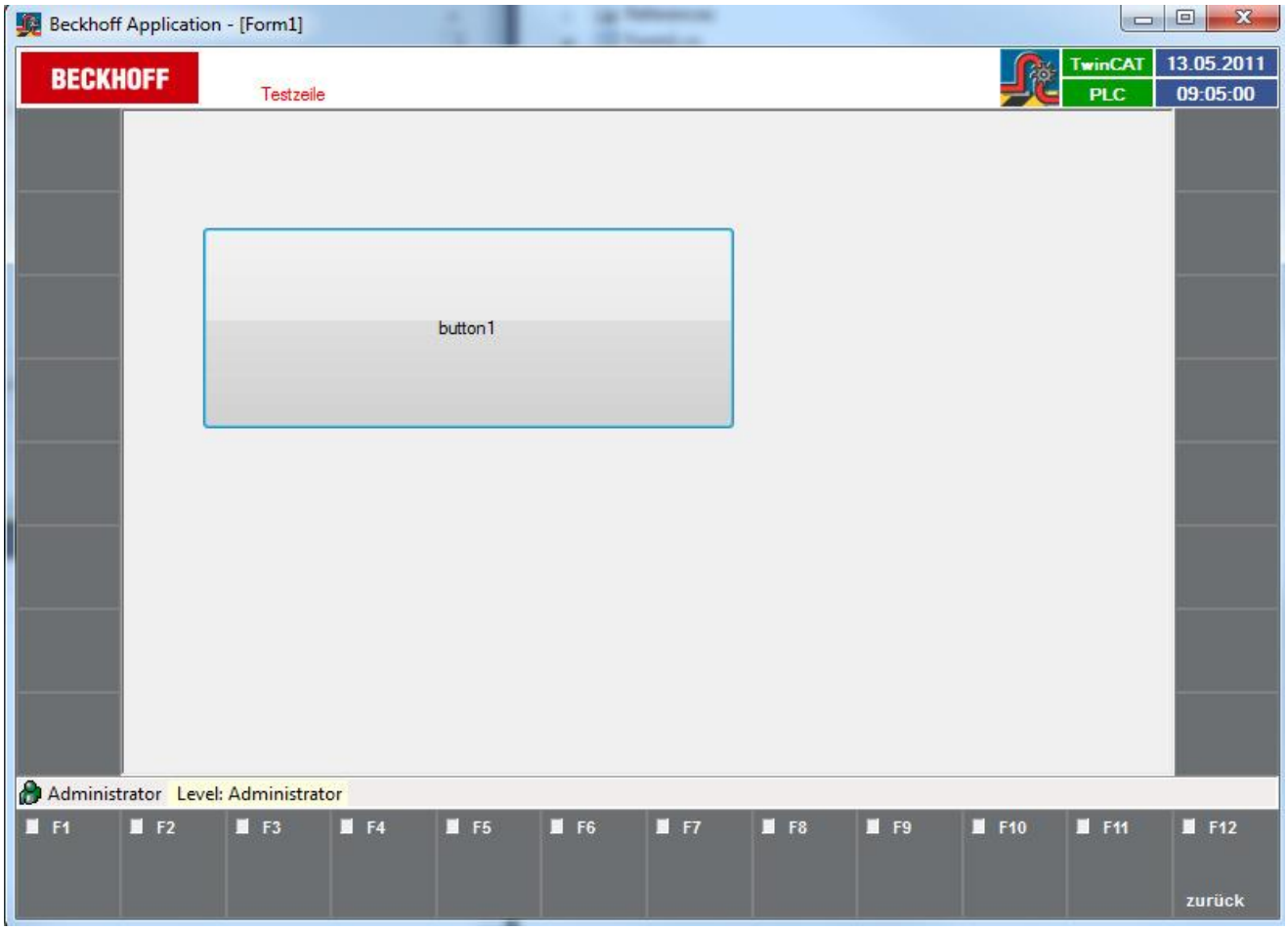
Mit <CTRL><ALT>“klick auf eine Taste“ wird der Menumanager in den Konfigurationsmodus gebracht. Um eine Taste mit dem Aufruf unseres „Form1“ aus der PluginDemo.dll zu belegen sind folgende Schritte auszuführen (vgl. Doku Menumanager):
To put the menu Manager in configuration mode "click a button" while pressing <CTRL> <ALT>. To configure a button with the call to our "Form1" from the PluginDemo.dll

- Select "Form List" and then do "Load" to make the Form1 plugin demo available to the menu manager.



- The call to assign the "formDemo" a button instance:

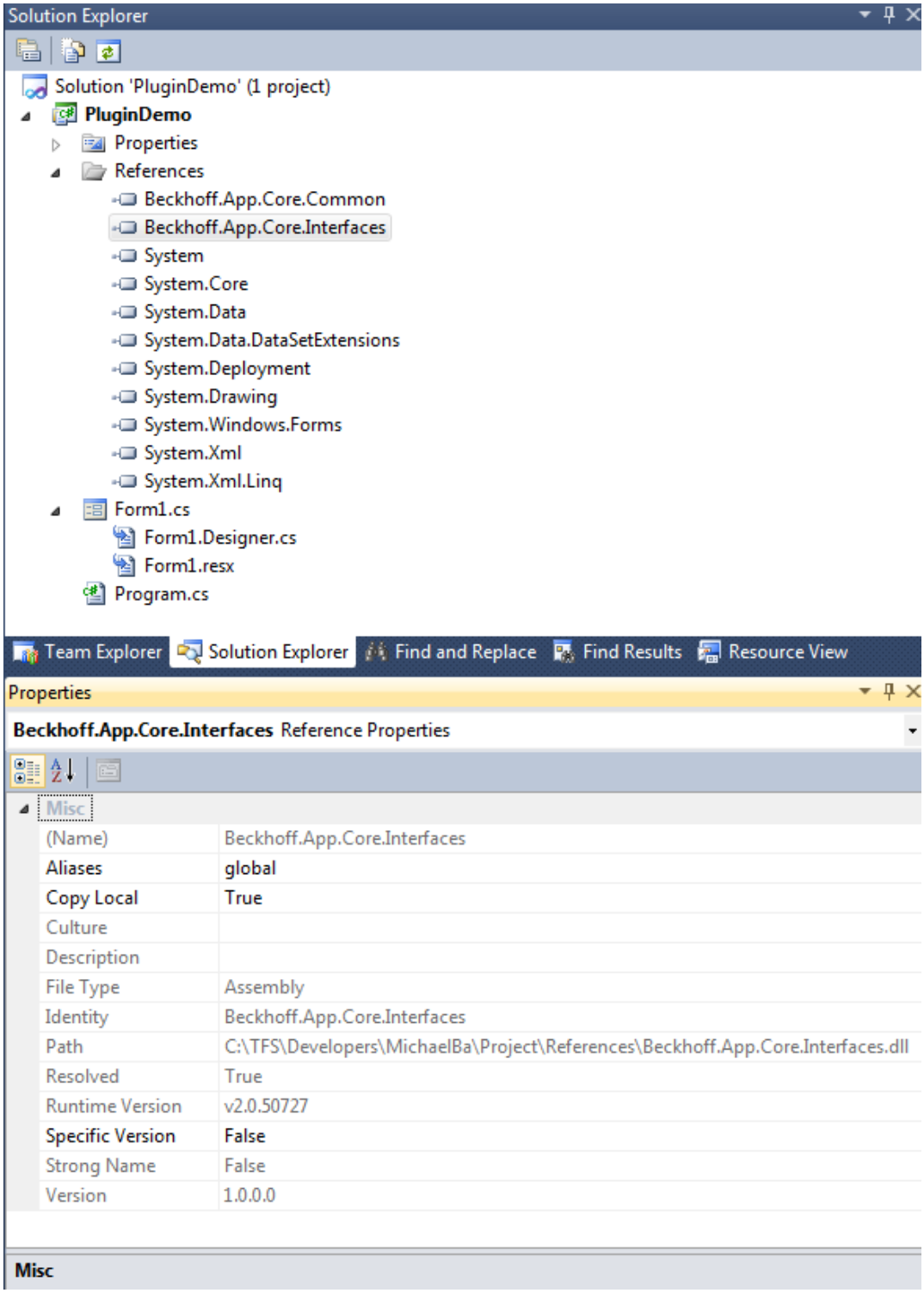



Beckhoff Application - [Form1]
BECKHOFF
Testzeile
TwinCAT
PLC
13.05.2011
09:05:00
button1
Administrator
Level: Administrator
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
zurück

The connection to the HMI is made through several interfaces and classes. To do so, following DLLs should be reference from the folder "References”:

- Beckhoff.App.Core.Interface
- Beckhoiff.App.Core.Common

Depending on the desired function, additional libraries may need to be referenced.



Default Key assignment definitions of a form

To set the default layout of a form in the menu manager, that is used when you first import the form, a "public static" method must be defined in the form that returns a "Dictionary<int, IBAFkeyData="">" as the return value. In this dictionary the default assignment is stored and is requested from the menu Manager when needed.

Example: Here is the button 5 "callMethod" registered with the button 6 a "PlcToggleVarAndFeedbackImage":

```
/// <summary>
/// Gets the default F key default assingment.
/// </summary>
/// <returns></returns>
public static Dictionary<int, IBAFKeyData> GetDefaultFKeyAssingment()
{
    var fkeyAssign = new Dictionary<int, IBAFKeyData>();
    var fkey = new BAFKeyDataStruct
    {
        AccessLevel = 2,
        EventType = "callMethod",
        Data = "CncStart",
        DefaultText = "Start",
        Icon = @"\Bitmap\State\ChStart.ico"
    };
    fkeyAssign.Add(5, fkey);


    fkey = new BAFKeyDataStruct
    {
        AccessLevel = 2,
        EventType = "PlcToggleVarAndFeedbackImage",
        Data = ".PlcAxisEnable",
        Data2 = @"\Bitmap\State\Enable2.ico",
        DefaultText = "Enable",
        Icon = @"\Bitmap\State\Enable.ico"
    };
    fkeyAssign.Add(6, fkey);
     return fkeyAssign;
 }
```

How are instances of global objects to the new form?

- The constructor of the form is filled in automatically by unity with the appropriate parameters.
- If the parameter "String InstanceName" exists in the constructor of the current instance name is injected there.
- If the parameter "String ParameterMenuManager" exists in the constructor of the "Data2" from the key assignment of calling FKeys parameter there.
- Properties are automatically filled when instantiating the form from the menu Manager with instances of the pass sending, if these are available types in the General list of the object. (Here is however not guaranteed, when the properties are injected)
- All "Public void" methods with zero or one parameters can be accessed via the menu Manager by "callMethod" and thus be placed on any function buttons.

- Example Constructor:

```
public FormTest()
{
    InitializeComponent();
}

public FormTest(TcAdsServer adsServerCNC,
        IBALanguage language,
        IBASettings settings,
        IBAMainFrame mainFrame,
        IBAAdsServer adsServer,
        IBAMenu menu,
        IBALogger log,
        IBAContainerFacade container,
        IBAMessageService msgService,
        IBAUserAdmin userAdmin,
        string instanceName,
        string ParameterMenuManager)
    : this()
{

    _language = language;
    _settings = settings;
    _mainFrame = mainFrame;
    _adsServer = adsServer;

    _settings.SettingsChangedEvent += _settings_SettingsChangedEvent;
    _language.LanguageChangedEvent += new EventHandler(_language_LanguageC
hangedEvent);

    _nciClient = _adsServer.GetAdsClient<BAAdsNciClient>("NCI");

usw…….
```

The menu Manager by "outside" announce new event types

- Types can be disclosed from 'outside' the menu manager via a new event.
- To do this, the attribute `TcMenuEvent` must be defined in a class for a method. The event name is specified as a parameter of the attribute.
- The following parameters of the method are supported: a string or five parameters, or a parameter of type TcMenuEventParameter
- About the method of the menu Manager AddMenuExtension (object o); will be announced the class then the Menumanger.

Bsp:

```
using System;
using System.Collections.Generic;
using System.Text;
using Beckhoff.App;
namespace TcApplication
{
    class TcMenuPlcConnect
    {
        [TcMenuEvent("PlcSetVar")]
        public void SetVarInPlc(string plcVar)
        {
            System.Windows.Forms.MessageBox.Show("PlcSetVar aufgerufen mit dem
Parameter: " + plcVar);
       }
        [TcMenuEvent("PlcToggleVar")]
        public void ToggleVarInPlc(string plcVar)
        {
            System.Windows.Forms.MessageBox.Show("PlcToggleVar aufgerufen mit dem
Parameter: " + plcVar);
        }
    }
}


Irgendwo im Code:
Somewhere in Code


TcMenuPlcConnect menuPlcConnet = new TcMenuPlcConnect();

menuLocal.AddMenuExtension(menuPlcConnet);
```

- To get more information about the call button, it is also possible to implement a method with 5 parameters (example):

```
[TcMenuEvent("PlcToggleVarAndFeedback")]
public void ToggleVarInPlcWithColor(string plcVar, string plcVar2, TwinCAT.App.TcFKey fkey, int index, TcMenuEvent.EventReason reason)
```

- Significance of the parameter:
  - plcVar: Parameter from Data1
  - plcVar2: Parameter from Data2
  - fKey: Key group that triggered the event
  - index: Key index of Events
  - reason: Reason of Events:
    KeyDown: Key is pressed
    KeyUp: Key is released
    ElementShow: Button is displayed
    ElementHide: Button is hidden

- Other possibilities:

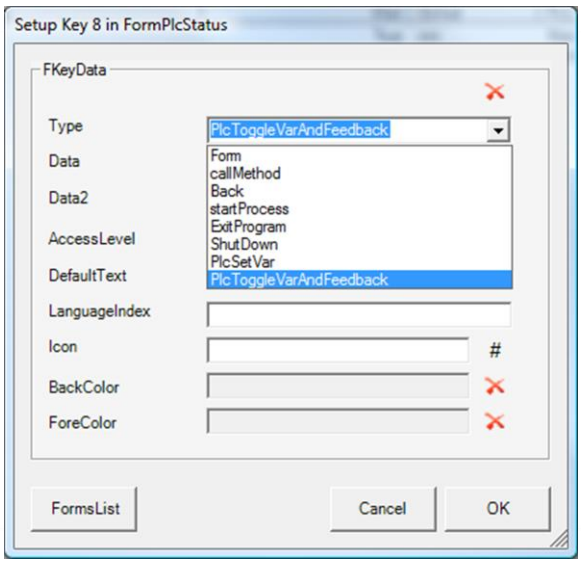
```
[TcMenuEvent("PlcToggleVarAndFeedback")]
public void ToggleVarInPlcWithColor(TcMenuEventParameter param)
    switch (param.Reason)
    {
        case TcMenuEvent.EventReason.KeyUp: ……..
```

The class contains TcMenuEventParameter with the elements described above. The class TcMenuEventParameter additionally contains the parameter

- InstanceName: Instance name of the current form

BECKHOFF

- Now the Event Types "PlcSetVar" and "PlcToggleVarAndFeedback" are known in the menu manager and can be selected. If now a menu button is pressed, the method is called and the parameters from the Data and Data2 field are passed.
  (These two types are the standard already implemented)



Remap the Keys from the outside at runtime

BECKHOFF

To change the keyboard layout at runtime, there is in IBAMenu methods

```
/// <summary>
/// Gets the F key assignment
/// </summary>
/// <param name="token">The token (key index).</param>
/// <returns></returns>
IBAFKeyData GetFKeyAssign(int token);

/// <summary>
/// Sets the F key assignment.
/// </summary>
/// <param name="data">The data.</param>
/// <param name="token">The token.</param>
void SetFKeyAssign(IBAFKeyData data, int token);
```

With GetFKeyAssign, the assignment of a key (identified by their ID Ex 1 = F1, 13 = ALT F1) can be read. In IBAFKeyData all the information of a key are stored. (Color, label, function). With SetFKeyAssign the assignment can be given to the Menumanager. This should be called (in Windows Forms) after the forms Visible has gone to TRUE (Event Visible Changed). Immediately after calling the method, the new values are active.

Example for confirming the ALT-F3 key with the command "start process" and CMD.EXE:

```
var data = _menu.GetFKeyAssign(15);
if (data != null)
{
    data.BackColor = Color.Salmon;
    data.DefaultText = "start CMD";
    data.EventType = "startProcess";
    data.Data = "cmd.exe";
    data.Data2 = "";

    _menu.SetFKeyAssign(data, 15);
}
```