

## Plugin für das Beckhoff HMI entwickeln

Michael Balsfulland

CNC Applikation

Telefon: + 49 (0)5246 963 236

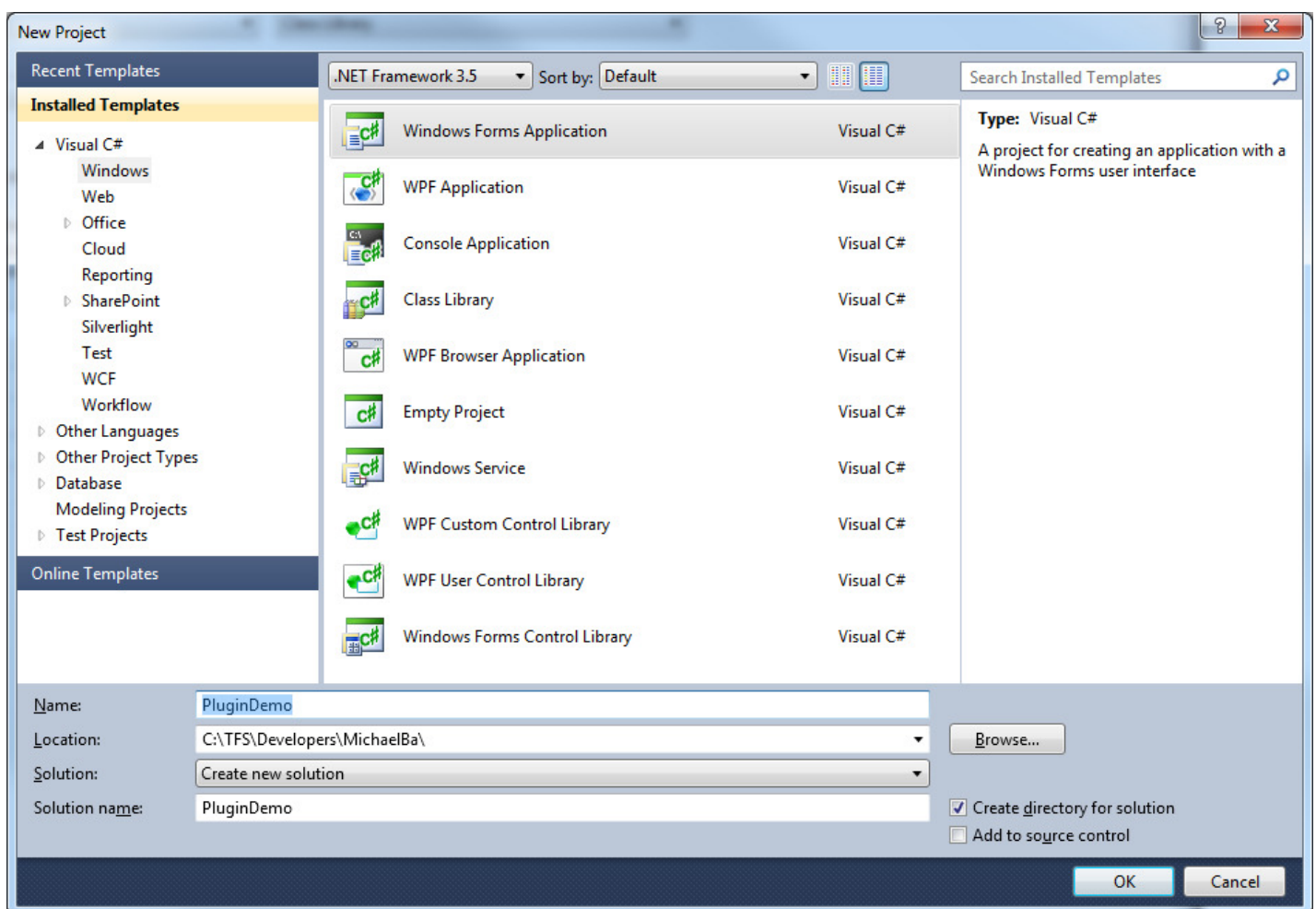
Telefax: + 49 (0)5246 963 9 236

[m.balsfulland@beckhoff.com](mailto:m.balsfulland@beckhoff.com)

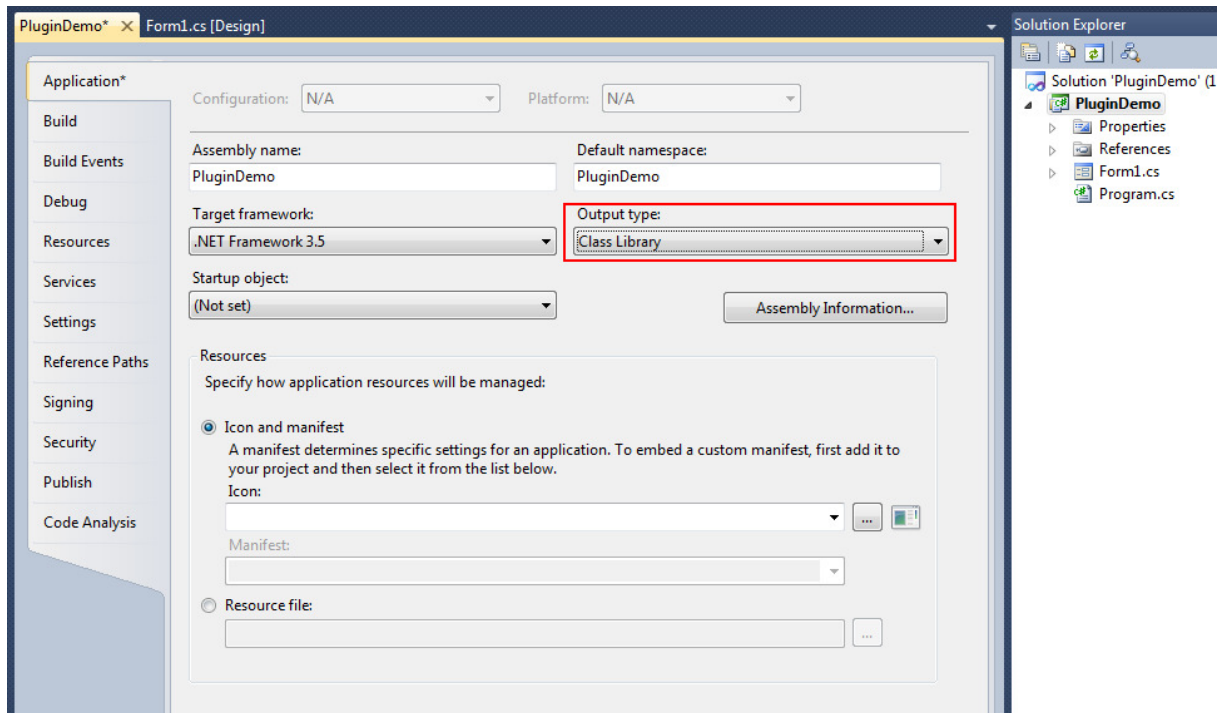
23.2.2012

V 1.1

Zum Starten im VisualStudio 2010 ein neues „Windows Forms Application“ Projekt anlegen:

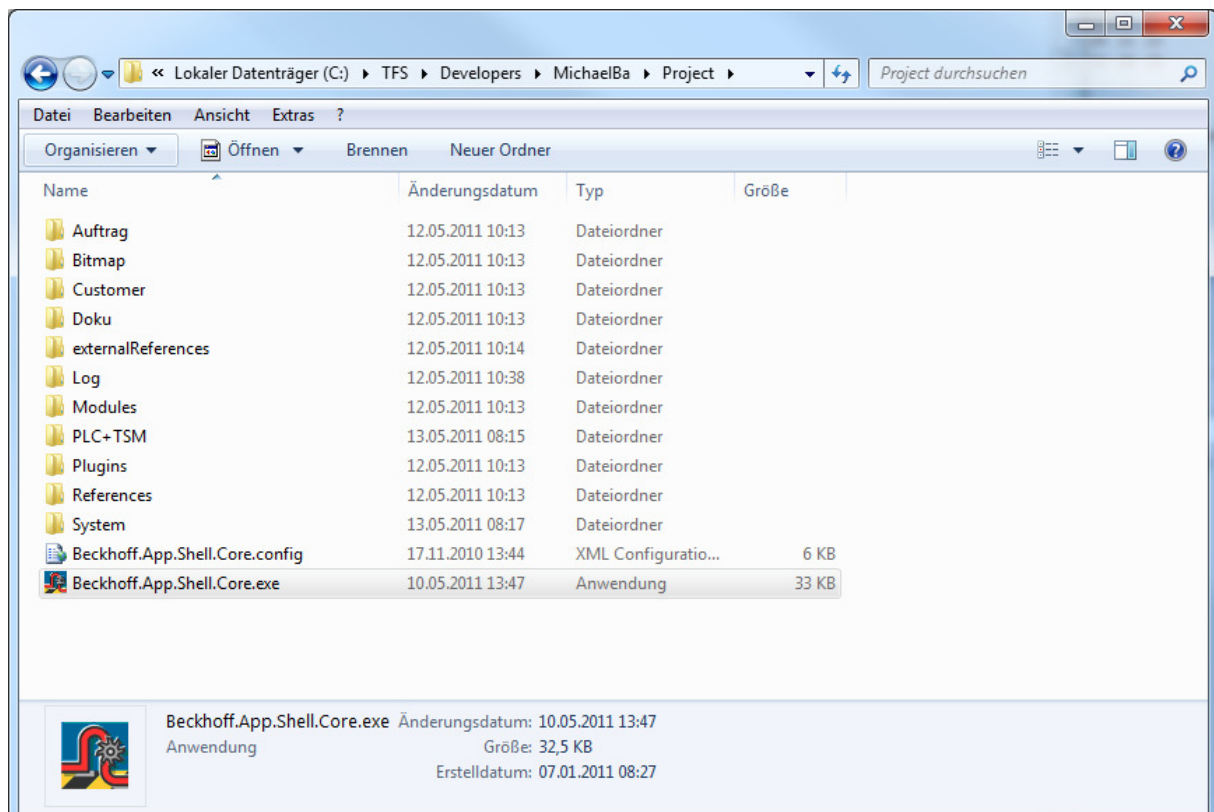


Dann in den Projekteigenschaften den „Output type“ auf Class Library einstellen:

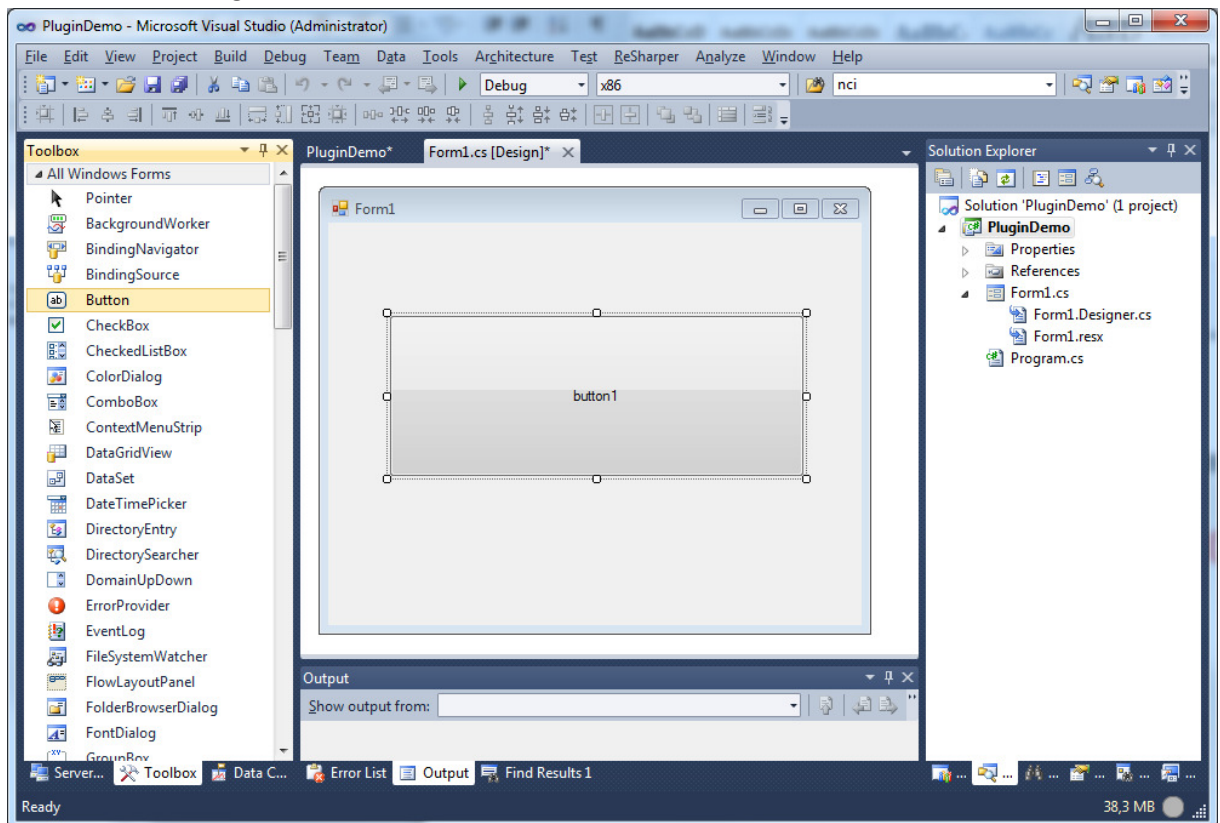


In diesem Beispiel sind folgende Pfade eingestellt:

- Die ausführbare HMI liegt unter „C:\TFS\Developer\MichaelBa\Project“
- Der Projektpfad zum Plugin: „C:\TFS\Developer\MichaelBa\PluginDemo“



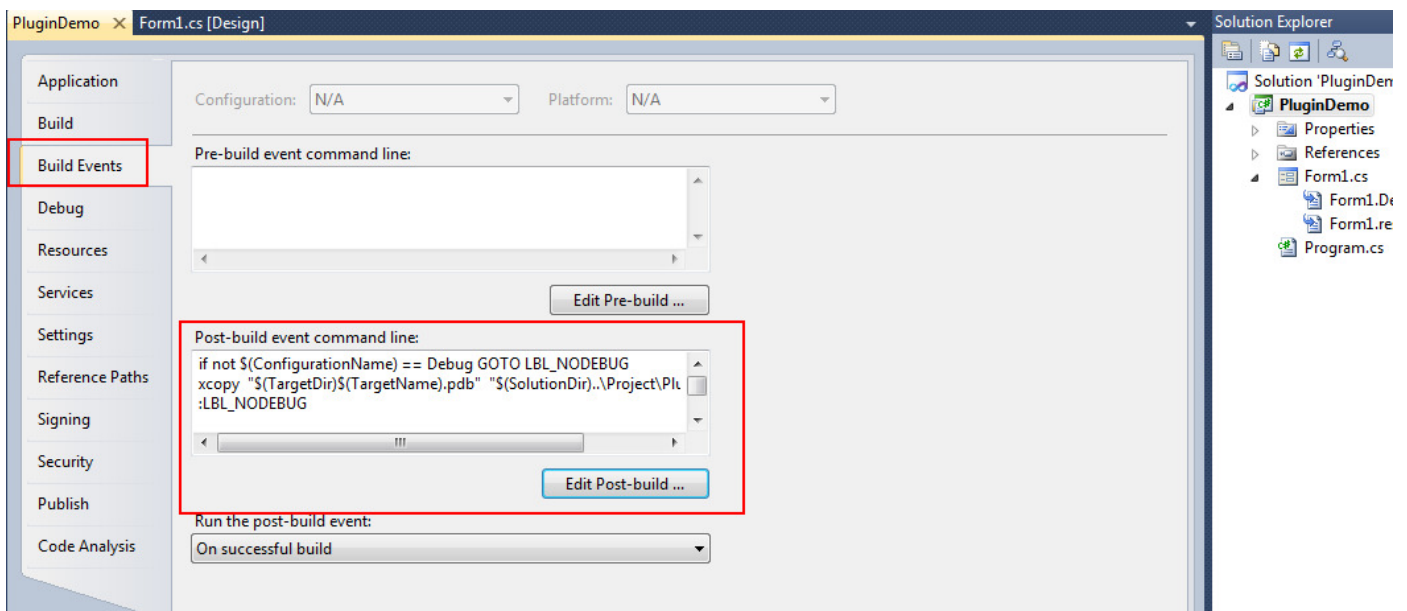
Zwecks Test einen großen Button auf die Form ziehen:



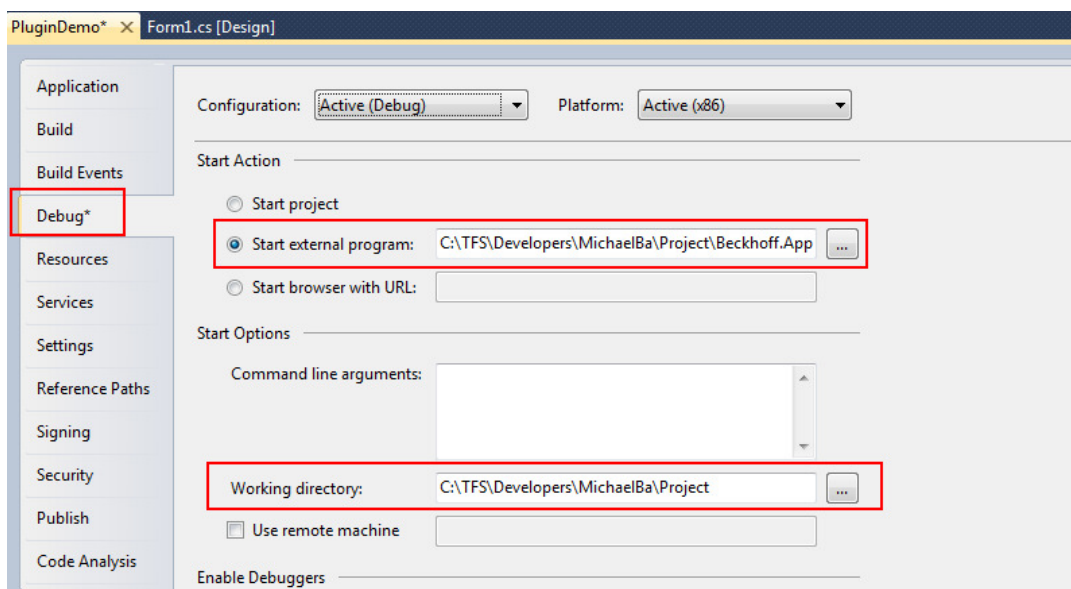
Damit das PlugIDemo im Debugger gestartet werden kann, muss die DLL mit einem „PostbuildCommand“ in das Verzeichnis Plugins der ausführbaren HMI kopiert werden und die „Beckhoff.App.Shell.Exe“ sollte vom Visual Studio aus gestartet werden. In der HMI muss das „Form1“ aus PluginDemo.dll im Menumanager bekannt gemacht und aufgerufen werden. Dazu sind folgende Schritte nötig:

- In den Eigenschaften des Projects in den „Build Events“ in der „Post-built event command line“ folgendes eintragen

```
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
```



- Im Bereich „Debug“ „Beckhoff .App.Shell.Core.Exe“ als auszuführendes Programm mit entsprechendem Arbeitspfad eintragen:



In der kostenlosen Express Version vom Visual Studio gibt es keine komfortable Möglichkeit, ein externes Programm zum Debuggen einzustellen.

Diese Einstellung kann allerdings „per Hand“ mit einem Texteditor in das .csproj File des Projekts eingetragen werden.

Die rot markierten Zeilen sind zu ergänzen und anzupassen:

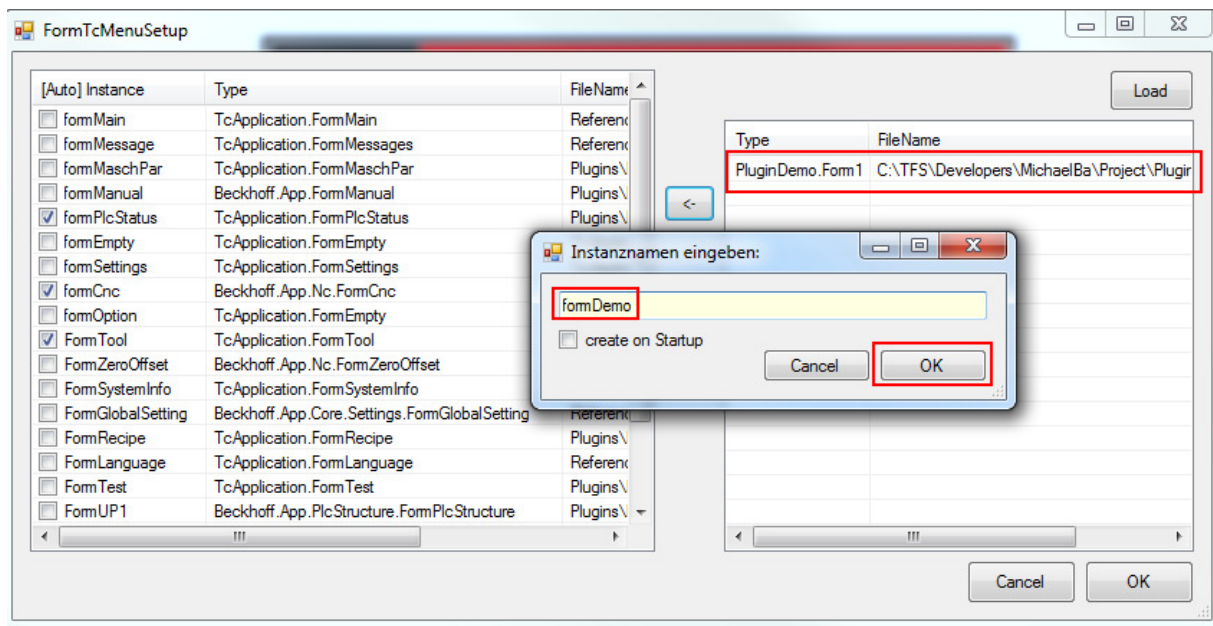
Auszug aus der .csproj Datei:

```
<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>
```

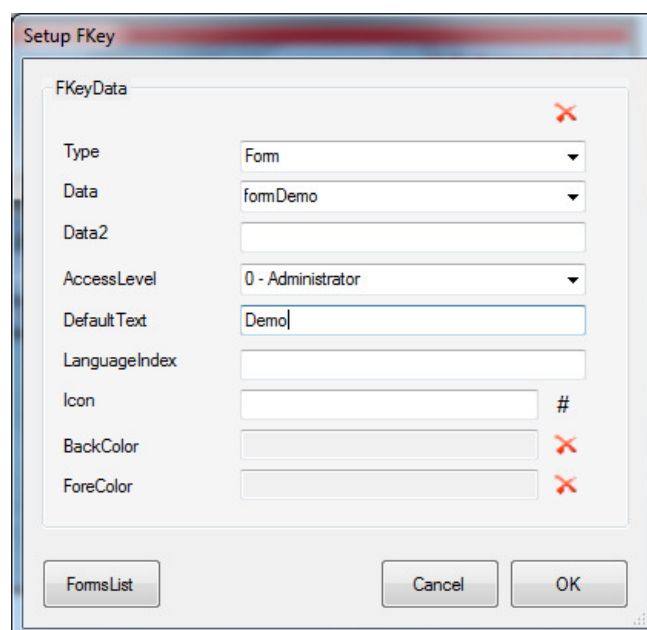
Über <F5> „Start Debugging“ kann jetzt die HMI gestartet werden.

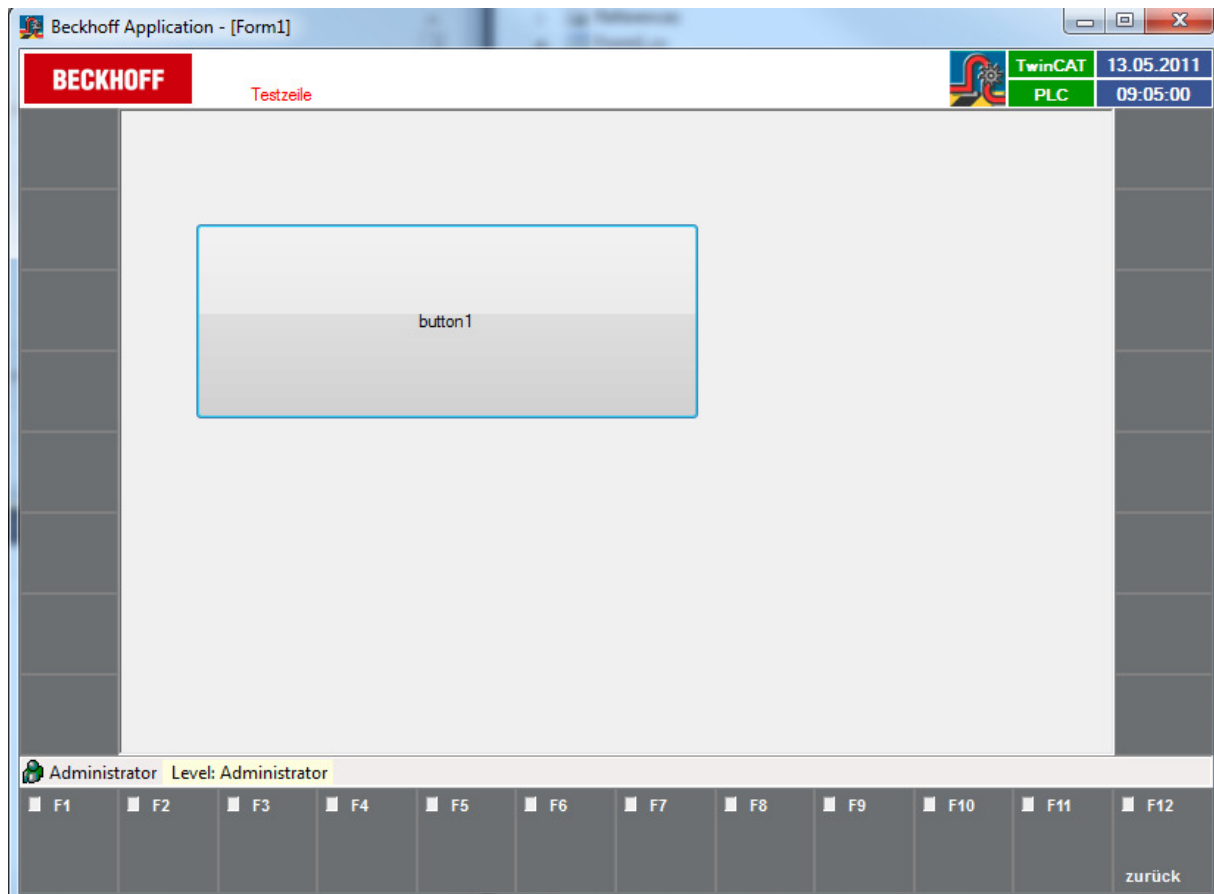
Mit <CTRL><ALT>„klick auf eine Taste“ wird der Menumanager in den Konfigurationsmodus gebracht. Um eine Taste mit dem Aufruf unseres „Form1“ aus der PluginDemo.dll zu belegen sind folgende Schritte auszuführen (vgl. Doku Menumanager):

- Über „FormsList“ und dann „Load“ das Form1 aus PluginDemo dem Menumanager bekannt machen.



- Den Aufruf der Instanz „formDemo“ einer Taste zuweisen:



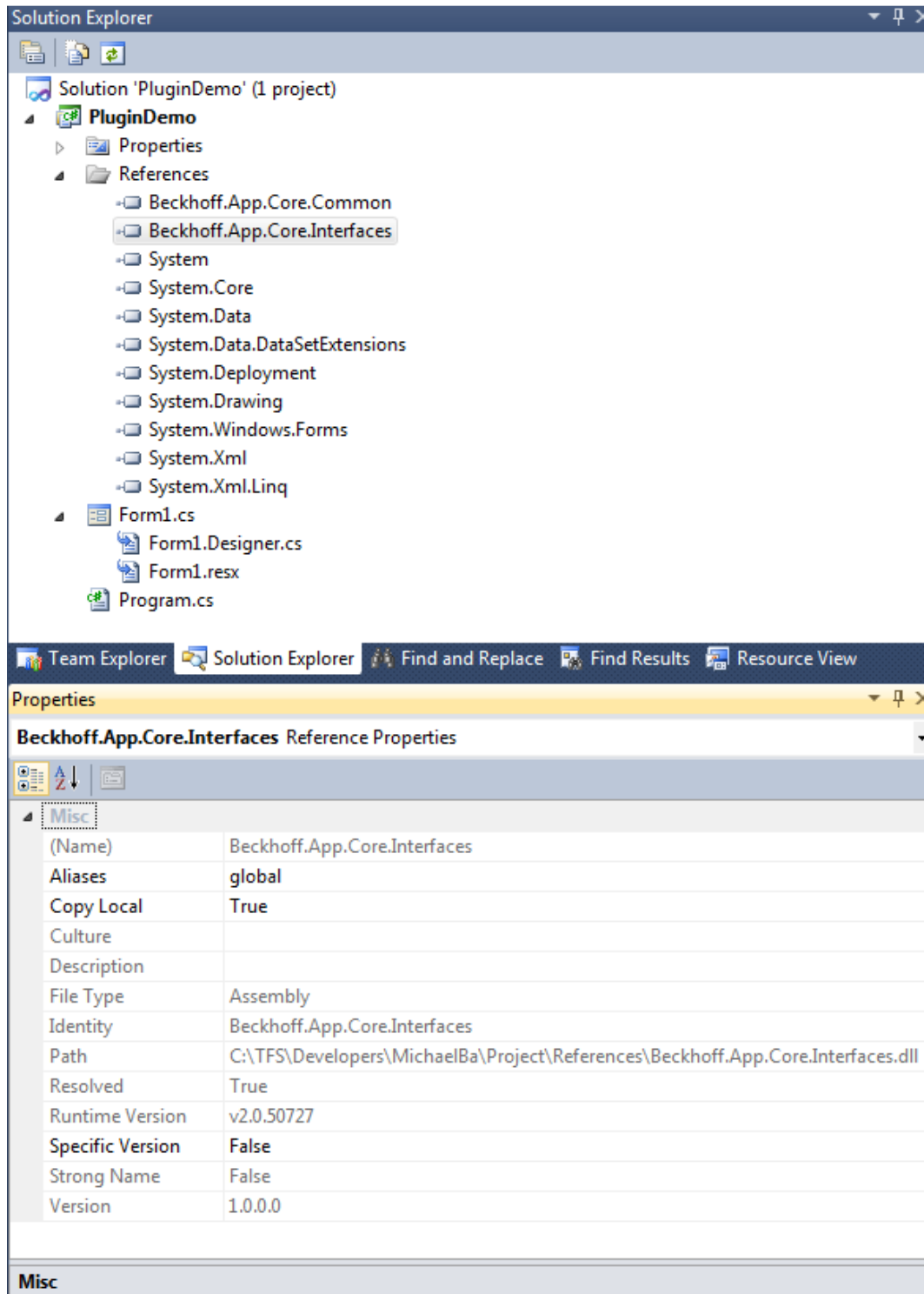




Die Verbindung zur HMI wird durch einige Interfaces und Klassen hergestellt. Dazu sollten folgende DLLs aus dem Ordner „References“ der HMI referenziert werden:

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

Je nach gewünschter Funktion müssen ggf. noch weitere Bibliotheken referenziert werden.



The screenshot displays the Visual Studio IDE. The top pane shows the 'Solution Explorer' for a project named 'PluginDemo'. Under the 'References' folder, two external references are listed: 'Beckhoff.App.Core.Common' and 'Beckhoff.App.Core.Interfaces'. Below these, several system assemblies are listed, including 'System', 'System.Core', 'System.Data', 'System.Data.DataSetExtensions', 'System.Deployment', 'System.Drawing', 'System.Windows.Forms', 'System.Xml', and 'System.Xml.Linq'. The bottom pane shows the 'Properties' window for the selected 'Beckhoff.App.Core.Interfaces' reference. The 'Misc' tab is active, displaying a table of properties for this reference.

Beckhoff.App.Core.Interfaces Reference Properties	
(Name)	Beckhoff.App.Core.Interfaces
Aliases	global
Copy Local	True
Culture	
Description	
File Type	Assembly
Identity	Beckhoff.App.Core.Interfaces
Path	C:\TFS\Developers\MichaelBa\Project\References\Beckhoff.App.Core.Interfaces.dll
Resolved	True
Runtime Version	v2.0.50727
Specific Version	False
Strong Name	False
Version	1.0.0.0



## Standard Tastenbelegung einer Form definieren

Um die Standardbelegung einer Form im Menumanager festzulegen, die beim ersten Importieren der Form benutzt wird, muss in der Form eine „public static“ Methode definiert werden, die als Rückgabewert ein „Dictionary<int, IBAFKeyData>“ zurück gibt. In diesem Dictionary wird die Standardbelegung gespeichert und bei Bedarf vom Menumanager angefragt.

Beispiel: Hier wird in der Taste 5 ein „callMethod“ eingetragen und in der Taste 6 ein „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;
}
```

## Wie kommen Instanzen von globalen Objekten zum neuen Form?

- Der Konstruktor der Form wird per Unity mit den entsprechenden Parametern automatisch gefüllt.
- Ist im Konstruktor der Parameter „String InstanceName“ vorhanden, wird dort der aktuelle Instanzname injiziert.
- Ist im Konstruktor der Parameter „String ParameterMenuManager“ vorhanden, wird dort der Parameter „Data2“ aus der Tastenbelegung des aufrufenden FKeys übergeben.
- Properties werden automatisch beim Instanzieren der Form vom Menumanager mit Instanzen der passenden Typen gefüllt, falls diese in der allgemeinen Objektliste verfügbar sind. (Hier ist allerdings nicht sichergestellt, wann die Properties injiziert werden)
- Alle „Public void“ Methoden mit keinem oder einem Parameter können über den Menumanager per „callMethod“ aufgerufen werden und somit auf beliebige Funktionstasten gelegt werden.
- Beispiel Konstruktoren:

```
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.....
```

## Dem Menumanager von „außen“ neue Eventtypen bekannt geben

- Dem Menumanager können von „außen“ neue Event - Typen bekannt gemacht werden.
- Dazu muss in einer Klasse für eine Methode das Attribut `TcMenuEvent` definiert werden. Als Parameter des Attributs wird der Eventname angegeben.
- Folgende Parameter der Methode werden unterstützt: ein String oder fünf Parametern oder einem Parameter vom Typ `TcMenuEventParameter`
- Über die Methode des Menumanagers `AddMenuExtension(Object o);` wird dem Menumanger die Klasse dann bekannt gegeben.

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:

```
TcMenuPlcConnect menuPlcConnet = new TcMenuPlcConnect();
menuLocal.AddMenuExtension(menuPlcConnet);
```

- Um mehr Informationen über die aufrufende Taste zu bekommen, besteht auch die Möglichkeit eine Methode mit 5 Parametern zu implementieren (Beispiel):

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

- Bedeutung der Paramter:
  - plcVar: Parameter aus Data1
  - plcVar2: Parameter aus Data2
  - fKey: Tastengruppe, die das Event auslöst hat
  - index: Tastenindex des Events
  - reason: Grund des Events:
    - KeyDown: Taste wurde gedrückt
    - KeyUp: Taste wurde losgelassen
    - ElementShow: Taste wurde angezeigt
    - ElementHide: Taste wurde versteckt

- Weitere Möglichkeit:

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

```

wobei die Klasse TcMenuEventParamter die oben beschriebenen Elemente enthält.

- Nun sind die EventTypen „PlcSetVar“ und „PlcToggleVarAndFeedback“ im Menumanager bekannt und können ausgewählt werden. Wird nun eine Menutaste betätigt, so wird die Methode aufgerufen und die Parameter aus dem Data und Data2 Feld werden übergeben. (Diese beiden Typen sind im Standard bereits implementiert)

Setup Key 8 in FormPlcStatus

FKeyData

Type: PlcToggleVarAndFeedback

Data: Form, callMethod, Back, startProcess, ExitProgram, ShutDown, PlcSetVar, PlcToggleVarAndFeedback

Data2:

AccessLevel:

DefaultText:

LanguageIndex:

Icon: #

BackColor: ✖

ForeColor: ✖

FormsList

Cancel

OK

## Tastenbelegung von außen zur Laufzeit ändern

Um die Tastenbelegung zur Laufzeit zu ändern, gibt es im IBAMenu die Methoden

```
/// <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);
```

Mit GetFKeyAssign kann die Belegung einer Taste

(identifiziert durch ihre ID Bsp 1 = F1, 13 = ALT F1) ausgelesen werden. In IBAFKeyData sind alle Informationen einer Taste abgelegt. (Farbe, Beschriftung, Funktion).

Mit SetFKeyAssign kann eine eigene Belegung dem Menumanager bekannt gegeben werden. Diese sollte (in Windows Forms) aufgerufen werden, nachdem Visible der Form auf TRUE gegangen ist (Event VisibleChanged).

Unmittelbar nach dem Aufruf der Methode werden die neuen Werte aktiv.

Beispiel zum Belegen der ALT-F3 Taste mit dem Kommando „startProcess“ und 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);
}
```