Tutorial - C# Anwendung mit Microsoft Visual Studio 2010

Inhaltsverzeichnis:

1 Einführung

In diesem Einführungsbeispiel werden Sie Schritt für Schritt eine Windows Anwendung zur Kommunikation mit einer HEIDENHAIN Steuerung erstellen. Dafür benötigen Sie nichts weiter, als die RemoTools SDK und eine Installation von Microsoft Visual Studio 2010. Für die Arbeit mit diesem Beispiel ist die kostenlos bei Microsoft erhältliche „express edition“ von Microsoft Visual Studio völlig ausreichend. Um Abweichungen vom hier beschriebenen Vorgehen zu vermeiden empfehlen wir ihnen dieses Beispiel mit Visual Studio 2010 und .NET Framework 4 zu erstellen.

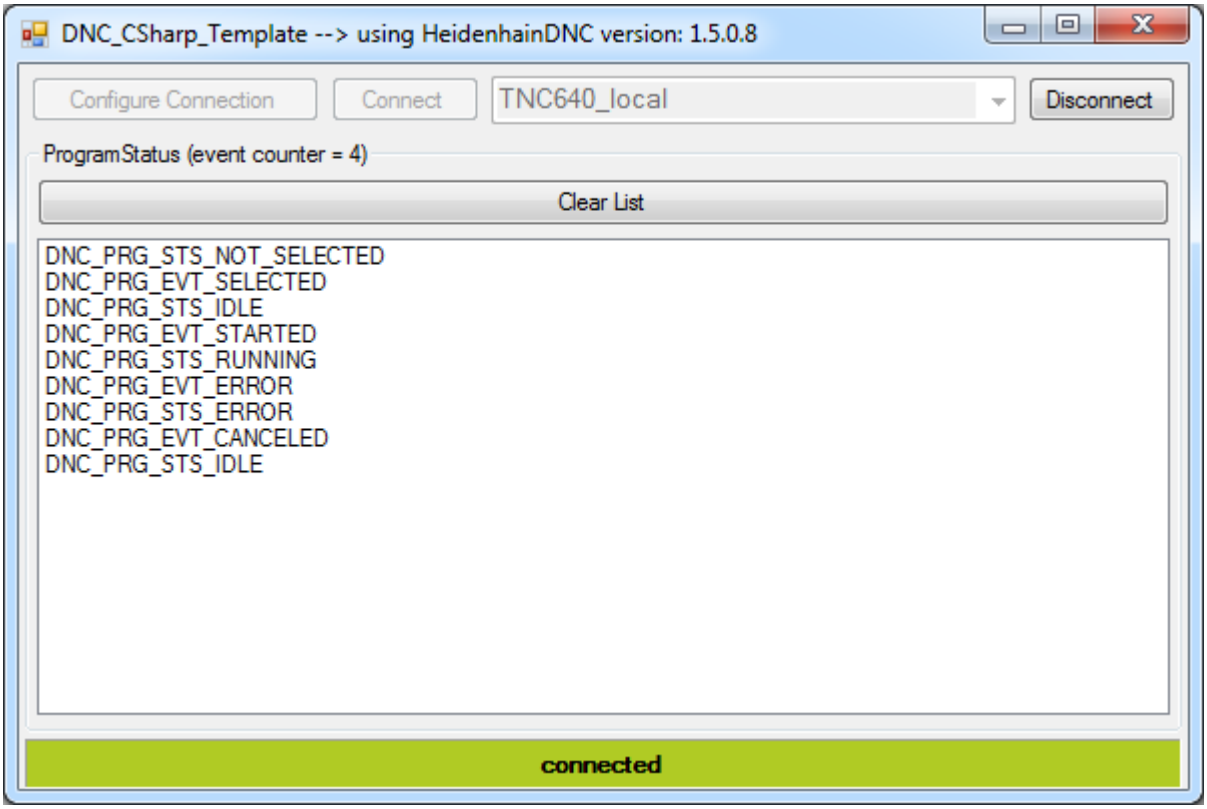

Abbildung 1

2 Anlegen des Projekts

Legen Sie bitte zuerst ein neues Visual C# „Windows Forms Application“ Projekt mit .NET 4.0 an. Führen Sie dazu bitte die in den folgenden Bildschirmausschnitten dargestellten Anweisungen durch:

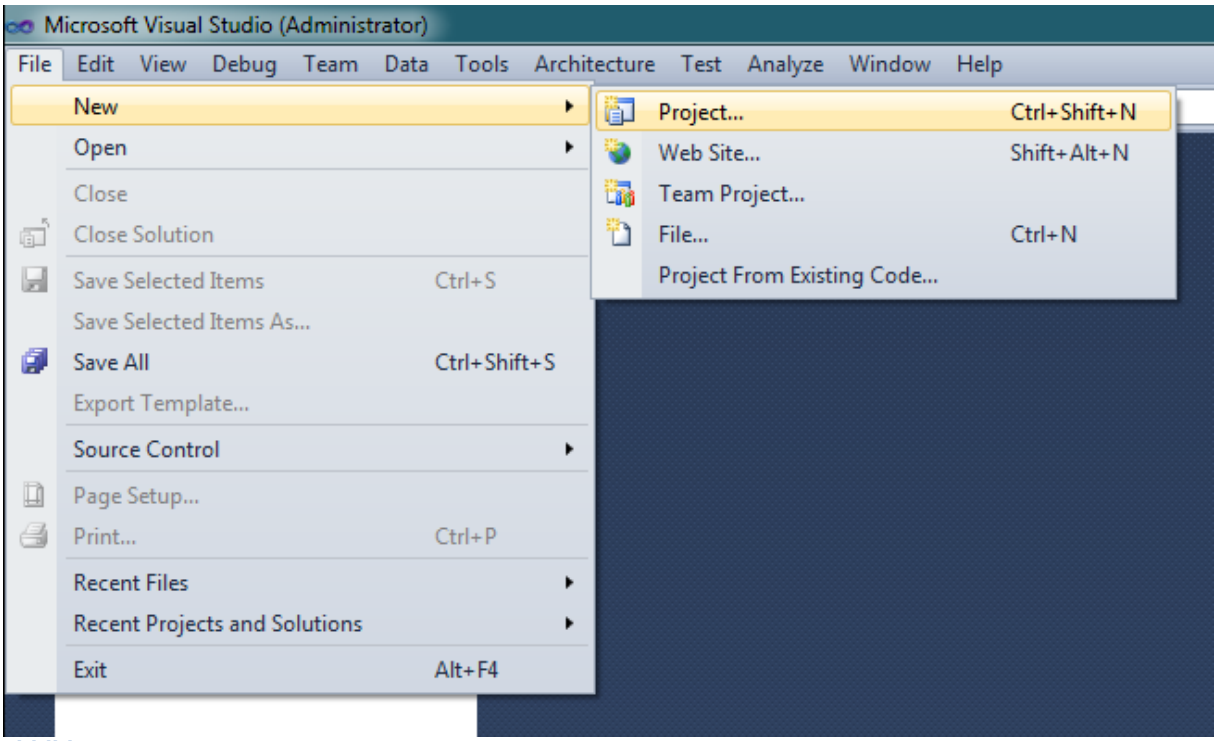

Abbildung 2

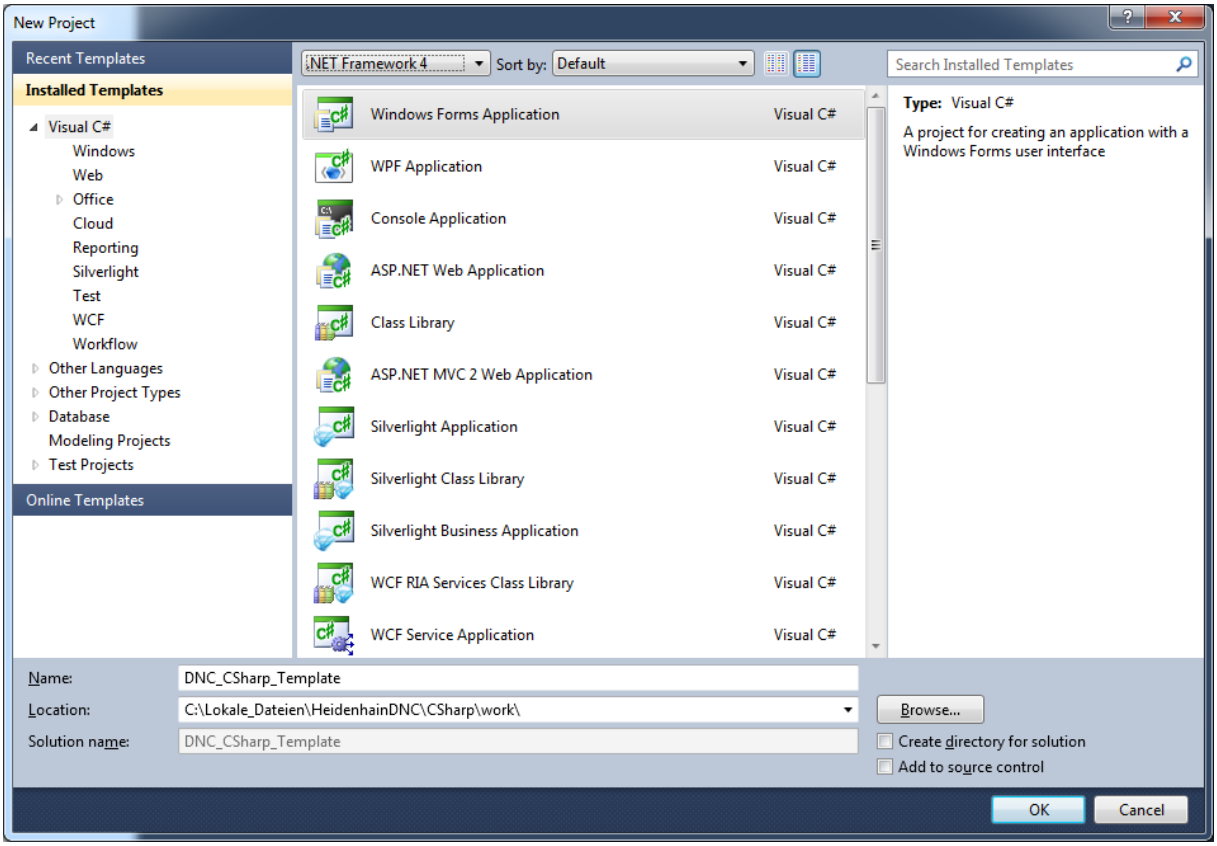

Abbildung 3

Nach Verlassen des in Abbildung 3 dargestellten Dialogs, wird ein Projekt mit dem Namen „DNC_CSharp_Template“ angelegt und geöffnet. Jetzt sollten Sie in etwa die in Abbildung 4

dargestellte Ansicht erhalten. Sollte der „Solution Explorer“ hier links oder die „Properties“ Leiste hier rechts fehlen, können Sie diese über das Menü „View“ einblenden.

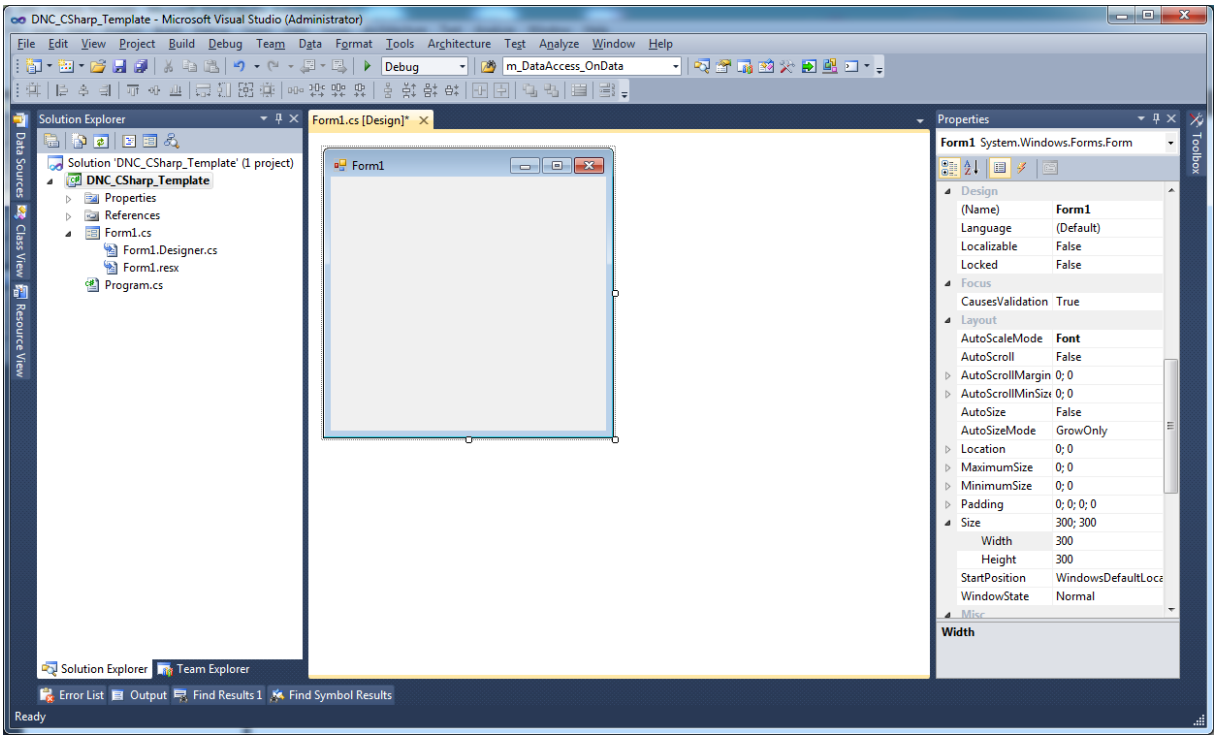

Abbildung 4

3 Einbinden der HeidenhainDNC Komponente

Für die Kommunikation mit einer Heidenhain Steuerung müssen Sie zuerst die HeidenhainDNC COM Komponente als Referenz in Ihr Projekt hinzufügen. Führen Sie dazu bitte die in folgenden Bildschirmausschnitten dargestellten Anweisungen durch.

Sollten Sie die in Abbildung 6 gezeigte Referenz auf die entsprechende HeidenhainDNC COM Komponente nicht finden, ist sie möglicherweise noch nicht installiert. Installieren Sie in diesem Fall HeidenhainDNC neu. Weitere Informationen dazu finden Sie auch im Hilfesystem HeidenhainDNC.chm.

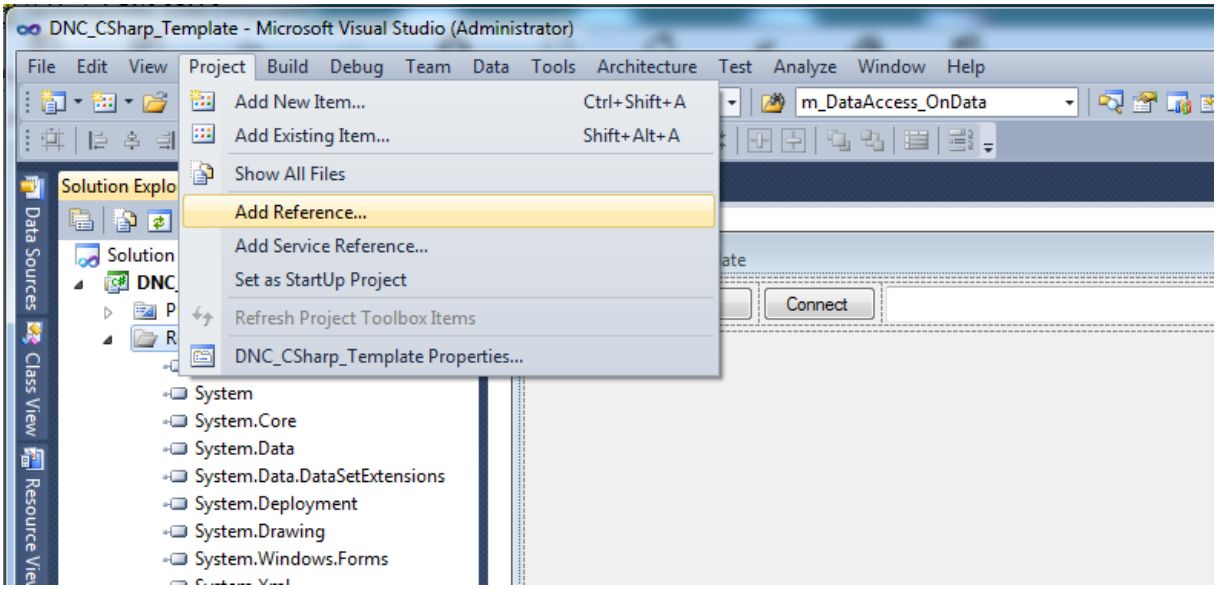

Abbildung 5

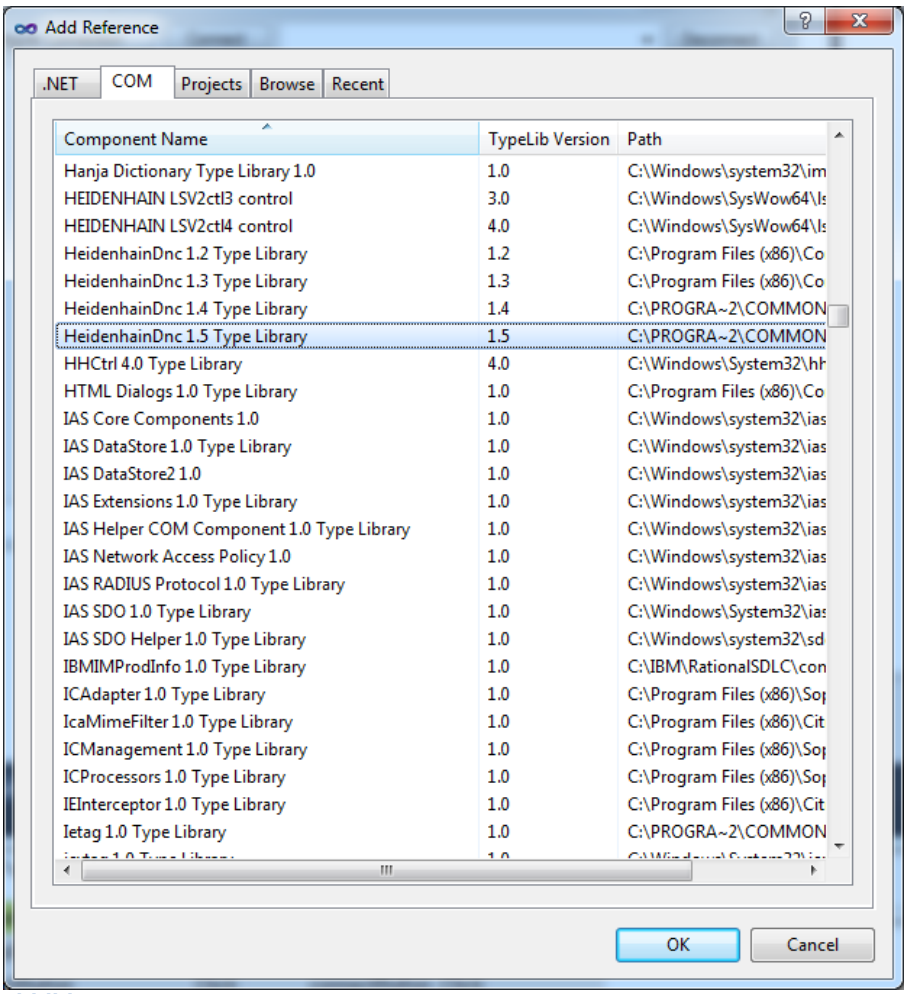

Abbildung 6

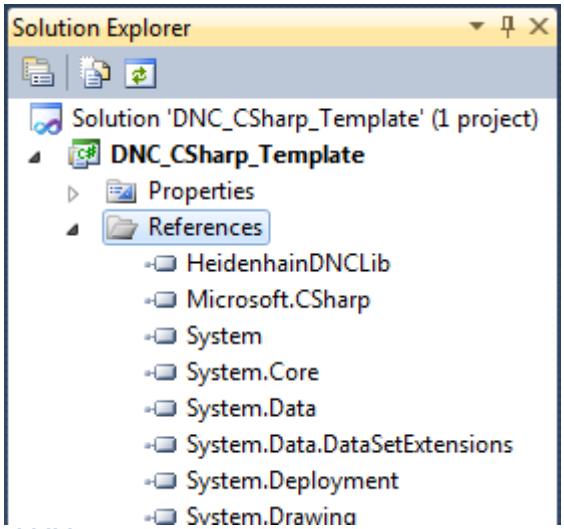

Abbildung 7

4 Statische Hilfsklasse MyHelper

Um das Projekt etwas übersichtlicher zu gestalten legen wir einige Hilfsmethoden an. Einige lagern wir in eine separate statische Klasse aus. Dazu legen Sie über das Menü Project eine neue Klasse an (siehe Abbildung 8).

Alternativ können Sie die fertige Quelletextdatei dem beigelegten Beispielprojekt entnehmen und über „Add Existing Item“ dem Projekt hinzufügen.

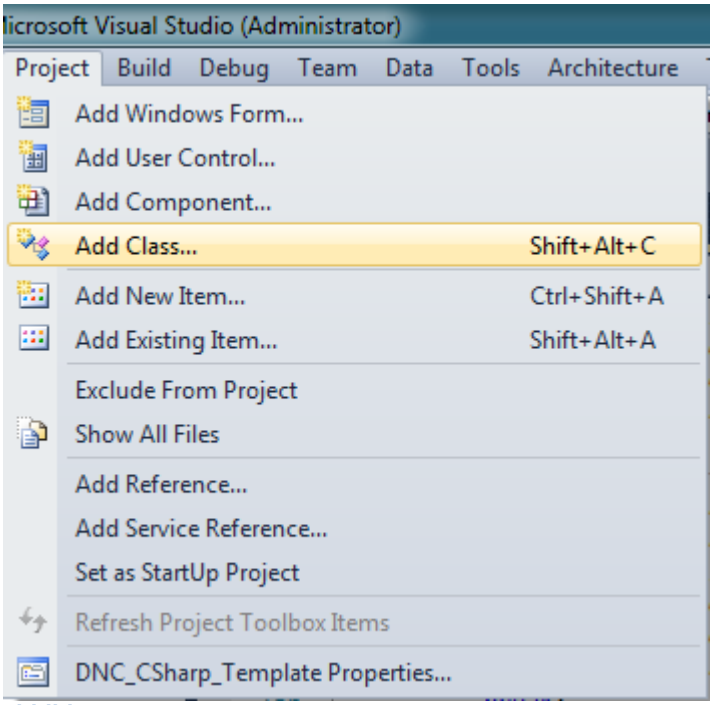

Abbildung 8

Die Klasse nennen Sie bitte MyHelper und bestätigen mit Okay. In der Klasse werden zwei neue Methoden angelegt.
CheckDncVersion um herauszufinden ob die COM Komponente von HeidenhainDNC korrekt installiert ist, und ob die DNC Version den Mindestanforderungen Ihrer Applikation entspricht.
ShowException und ShowComException um eine einheitliche und vereinfachte Darstellung von eventuell auftretenden Ausnahmen zu gewährleisten.

```
using System;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using System.Linq;

namespace DNC_CSharp_Template
{
```

4.1 Installation und Version der DNC Komponente prüfen

```
public static string CheckDncVersion(int mayor = 0, int minor = 0, int revision = 0, int build
= 0)
{
  string strVersionComInterface = null;
  HeidenhainDNCLib.JHMachine machine = null;
  try
  {
    machine = new HeidenhainDNCLib.JHMachine();
    strVersionComInterface = machine.GetVersionComInterface();
    int[] iaVersion = strVersionComInterface.Split('.').Select(n =>
Convert.ToInt32(n)).ToArray();
```

```
    // check if installed COM component meets minimum application requirement
    if ((iaVersion[0] < mayor) || (iaVersion[1] < minor) || (iaVersion[2] < revision) ||
(iaVersion[3] < build))
    {
      string strRequiredVersionComInterface = mayor.ToString() + "." + minor.ToString() + "."
+
                                              revision.ToString() + "." + build.ToString();
      // --- close application, because HeidenhainDNC version dos not meer minimum application
requirement
      MessageBox.Show("Your HeidenhainDNC version does not meet minimum application
requirement!" + Environment.NewLine + Environment.NewLine +
                      "Installed version: " + strVersionComInterface + Environment.NewLine +
                      "Required version : " + strRequiredVersionComInterface +
Environment.NewLine + Environment.NewLine +
                      "The application has to be closed." + Environment.NewLine,
                      "Verify HeidenhainDNC version",
                      MessageBoxButtons.OK,
                      MessageBoxIcon.Error);
      Application.Exit();
    }
  }
  catch (COMException)
  {
    // --- close application, because can't use HeidenhainDNC COM component (reinstall ?)
    MessageBox.Show("Can't use HeidenhainDNC COM interface!" + Environment.NewLine +
                    "The application has to be closed." + Environment.NewLine +
                    "Please reinstall HeidenhainDNC.",
                    "Verify HeidenhainDNC version",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
    Application.Exit();
  }
  catch (Exception)
  {
    // --- something went wrong by verifying HeidenhainDNC version --------
    DialogResult dr;
    dr = MessageBox.Show("Problem verifying HeidenhainDNC version!" + Environment.NewLine +
            "Possibly the application can't be executed properly." + Environment.NewLine +
            "Do you want to proceed anyway?" + Environment.NewLine,
            "Verify HeidenhainDNC version",
            MessageBoxButtons.YesNo,
            MessageBoxIcon.Question);
    if (dr == System.Windows.Forms.DialogResult.No)
      Application.Exit();
  }
  finally
  {
    if (machine != null)
      Marshal.ReleaseComObject(machine);
  }
  return strVersionComInterface;
}
```

4.2 Spezifische COM Ausnahmen melden

```
public static void ShowComException(int ErrorCode, string ClassName = null, string MethodName
= null)
{
  // --- build caption for COM exception message box ----------------------
  string strMessageBoxCaption = "COM Exception";
  if (!String.IsNullOrEmpty(ClassName) && !String.IsNullOrEmpty(MethodName))
    strMessageBoxCaption = ClassName + " --> " + MethodName;

  // --- build text for COM exception message box -------------------------
```

```
  string strExceptionMessage = "HRESULT: 0x" + Convert.ToString(ErrorCode, 16) +
Environment.NewLine;
  HeidenhainDNCLib.DNC_HRESULT hr = (HeidenhainDNCLib.DNC_HRESULT)ErrorCode;
  strExceptionMessage += Enum.GetName(typeof(HeidenhainDNCLib.DNC_HRESULT), hr);

  // --- show message box ----------------------------------------------
  MessageBox.Show(strExceptionMessage, strMessageBoxCaption, MessageBoxButtons.OK,
MessageBoxIcon.Error);
}
```

4.3 Allgemeine Ausnahmen melden

```
public static void ShowException(Exception ex, string ClassName = null, string MethodName =
null)
{
  // --- build caption for COM exception message box ----------------------
  string strMessageBoxCaption = "Exception";
  if (!String.IsNullOrEmpty(ClassName) && !String.IsNullOrEmpty(MethodName))
    strMessageBoxCaption = ClassName + " --> " + MethodName;

  // --- build text for COM exception message box -------------------------
  string strExceptionMessage = ".NET Exception:" + Environment.NewLine +
                               ex.Message + Environment.NewLine;

  // --- show message box ----------------------------------------------
  MessageBox.Show(strExceptionMessage, strMessageBoxCaption, MessageBoxButtons.OK,
MessageBoxIcon.Error);
}
```

5 Erstellen des Dialogs

In diesem Kapitel werden Sie Schritt für Schritt die Oberfläche des Hauptfensters Ihrer Anwendung zusammenstellen und konfigurieren.

5.1 Hauptfenster

Ändern Sie im „Solution Explorer“ den Namen des Hauptfensters von Form1.cs in FrmMain.cs. Passen Sie dann dessen Eigenschafften wie folg an:

Eigenschaft	Wert
(Name)	FrmMain
Text	DNC_CSharp_Template
Size.Width	600
Size.Heigth	400

TIPP:
Alle vom Standard abweichenden Parameter im Properties Fenster werden Fett angezeigt.

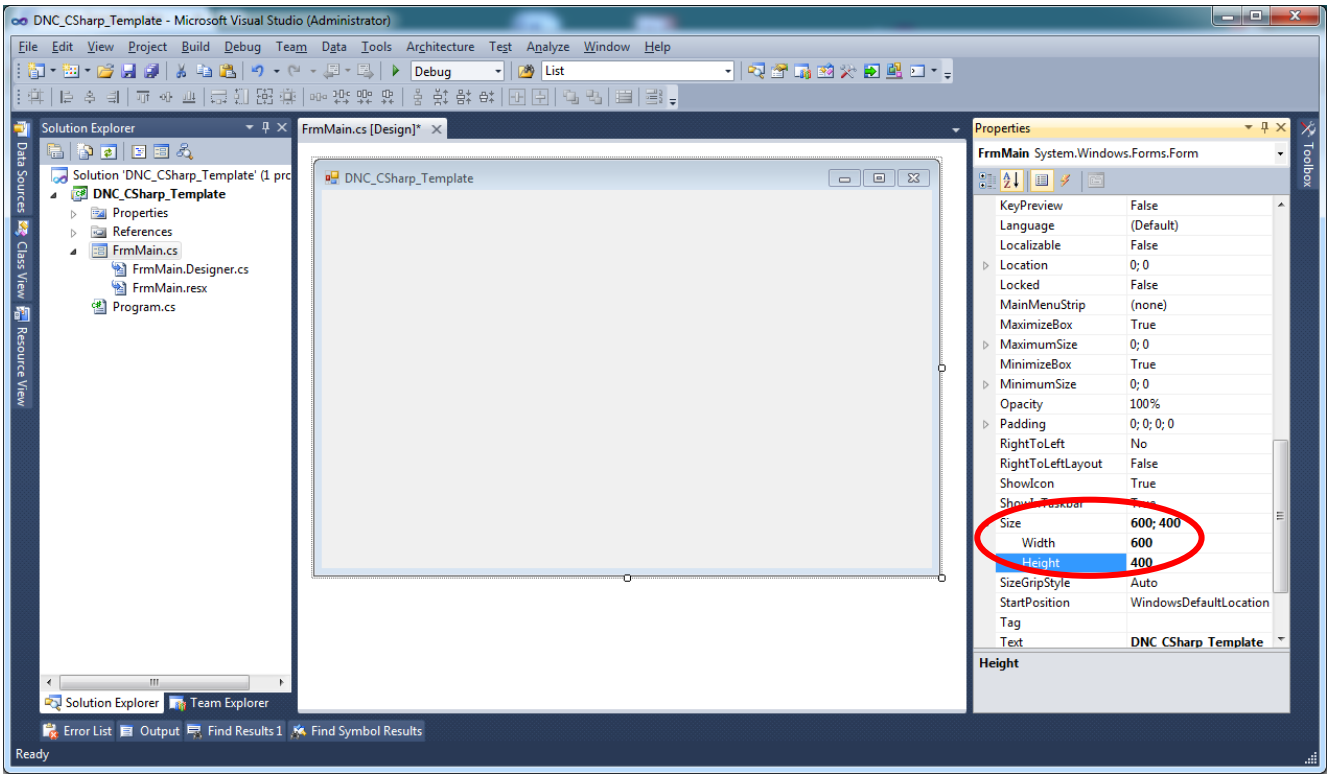

Abbildung 9

Um eine dynamische Bildaufteilung zu erreichen verwenden wir ein TableLayoutPanel.Damit teilen wir das Hauptfenster in drei Bereiche auf:

- Verbindungshandling
- Bedienung
- Verbindungsinformation

Das TableLayoutPanel finden Sie in der Toolbox unter der Rubrik „Containers“.

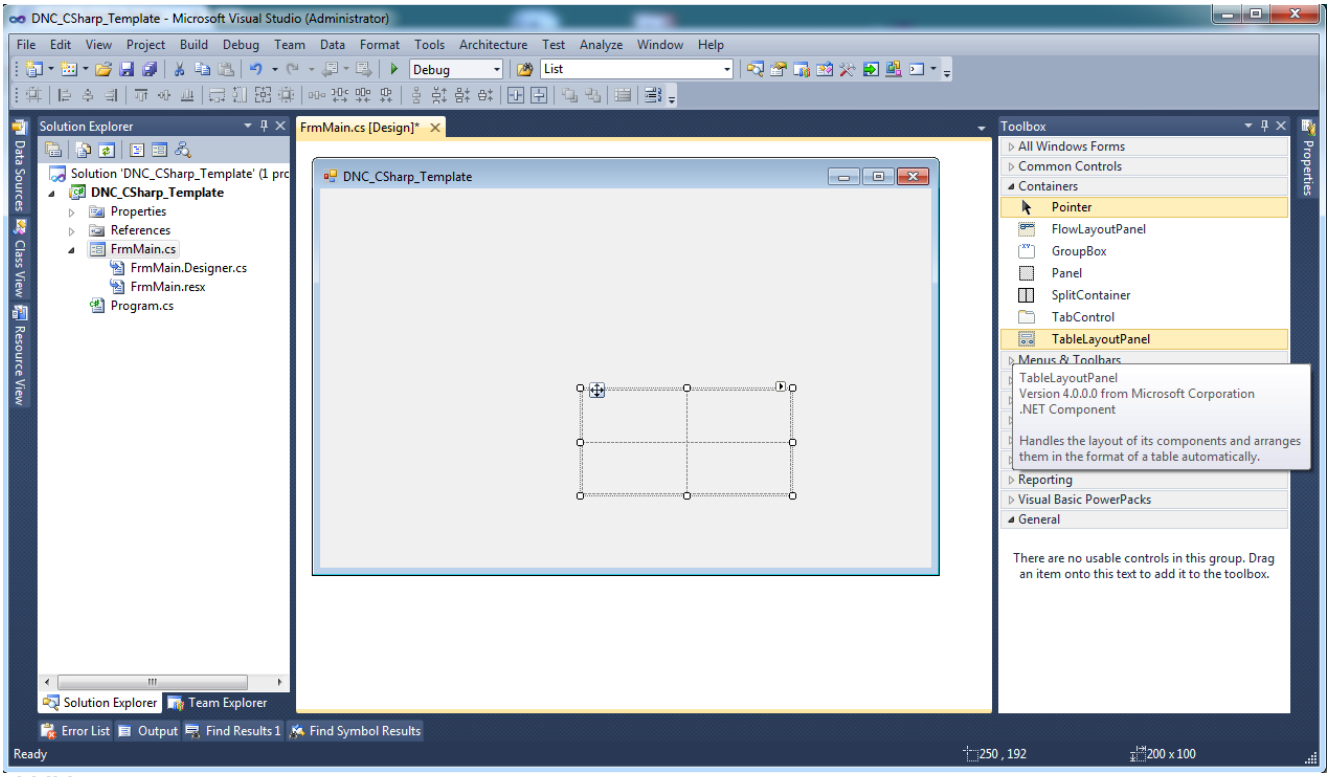

Abbildung 10

Bitte passen Sie die Eigenschaften des TableLayoutPanels wie folgt an:

Eigenschaft	**Wert**		
(Name)	tlpMain		
ColumnCount	1		
RowCount	3		
Dock	Fill		
Rows	Row1	Absolute	35
	Row2	Percent	100
	Row3	Absolute	25

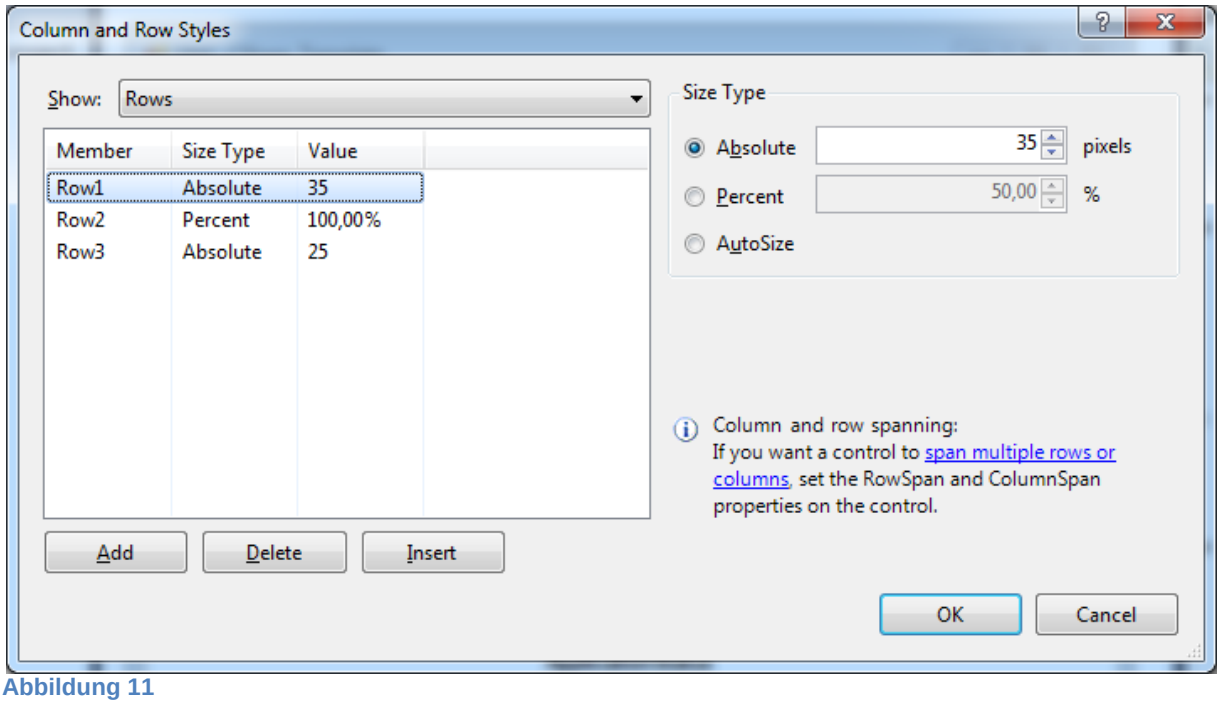

Abbildung 11

5.2 Bereich Verbindungshandling

In die erste Zeile (oben) wird ein weiteres TableLayoutPanel hinzugefügt.
Bitte passen Sie die Eigenschaften des neuen TableLayoutPanels wie folgt an:

Eigenschaft	Wert
(Name)	tlpConnectionHandling
ColumnCount	4
RowCount	1
Dock	Fill
Columns	Column1 Absolute 150 Column2 Absolute 80 Column3 Percent 100 Column4 Absolute 80

Füllen Sie die Spalten des TableLayoutPanels wie folgt mit den entsprechenden Steuerelementen. Spalte 1,2 & 4 jeweils mit einem Button und Spalte 3 mit einer ComboBox. Diese Steuerelemente finden Sie wieder in Ihrer Toolbox unter der Rubrik „Common Controls“.

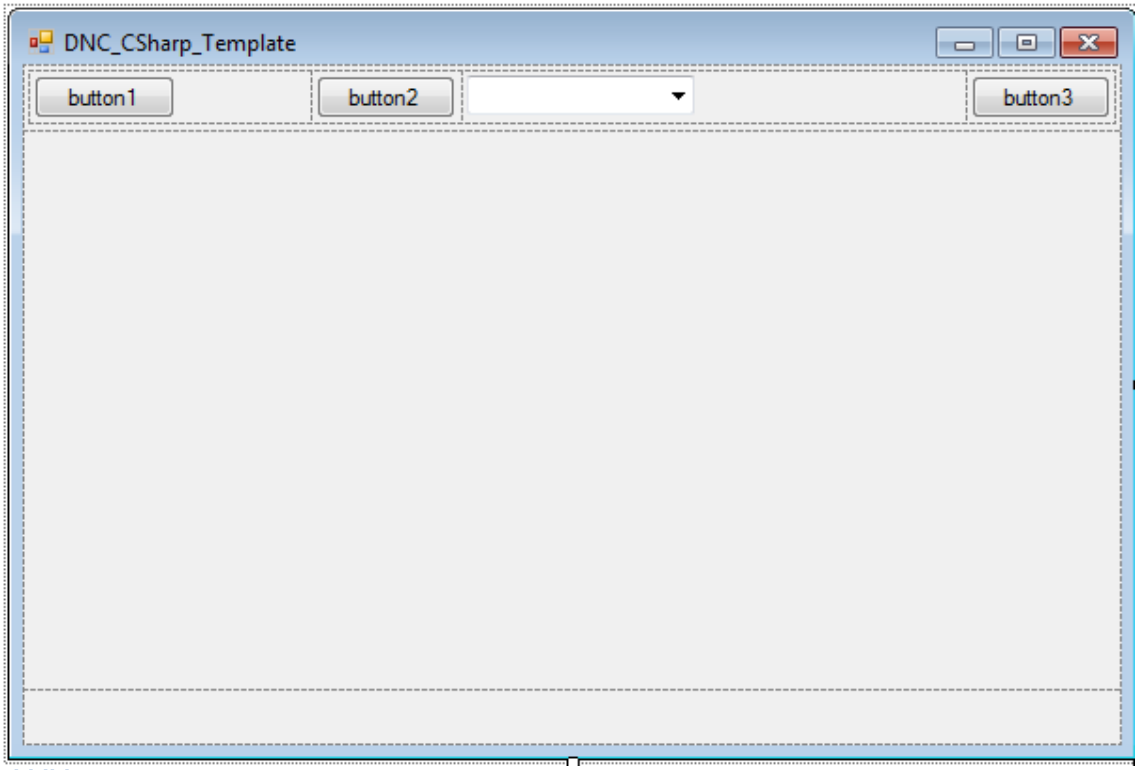

Abbildung 12

Bitte passen Sie die Eigenschaften der neuen Steuerelemente wie folgt an:

Spalte 1: Configure Connection Button

Eigenschaft	Wert
(Name)	btnConfigureConnection
Text	Configure Connection
Dock	Fill

Spalte 2: Connect Button

Eigenschaft	Wert
(Name)	btnConnect
Text	Connect
Dock	Fill

Spalte 3: Connection List ComboBox

Eigenschaft	Wert
(Name)	cbxConnectionList
Dock	Fill
Font	Microsoft Sans Serif; 8,25pt

Spalte 4: Disconnect Button

Eigenschaft	Wert
(Name)	btnDisconnect
Text	Disconnect
Dock	Fill

5.3 Bereich Verbindungsstatus

In die letzte Zeile (unten) wird ein Label eingefügt.
Bitte passen Sie die Eigenschaften des neuen Labels wie folgt an:

Eigenschaft	Wert
(Name)	lblApplicationStatus
Text	Application Status
AutoSize	False
TextAlign	MiddleCenter
Dock	Fill
Font	Microsoft Sans Serif; 8,25pt; style=Bold
BorderStyle	Fixed3D
BackColor	ControlLightLight

Jetzt sollte das Hauptfenster im Designer wie unter Abbildung 13 dargestellt werden.
Wenn Sie wollen, können Sie die Anwendung jetzt auch mit „start Debuggin“ im Menü „Debug“ starten. Hier können Sie auch prüfen, wie sich die Steuerelemente beim Ändern der Größe des Fenster anpassen.

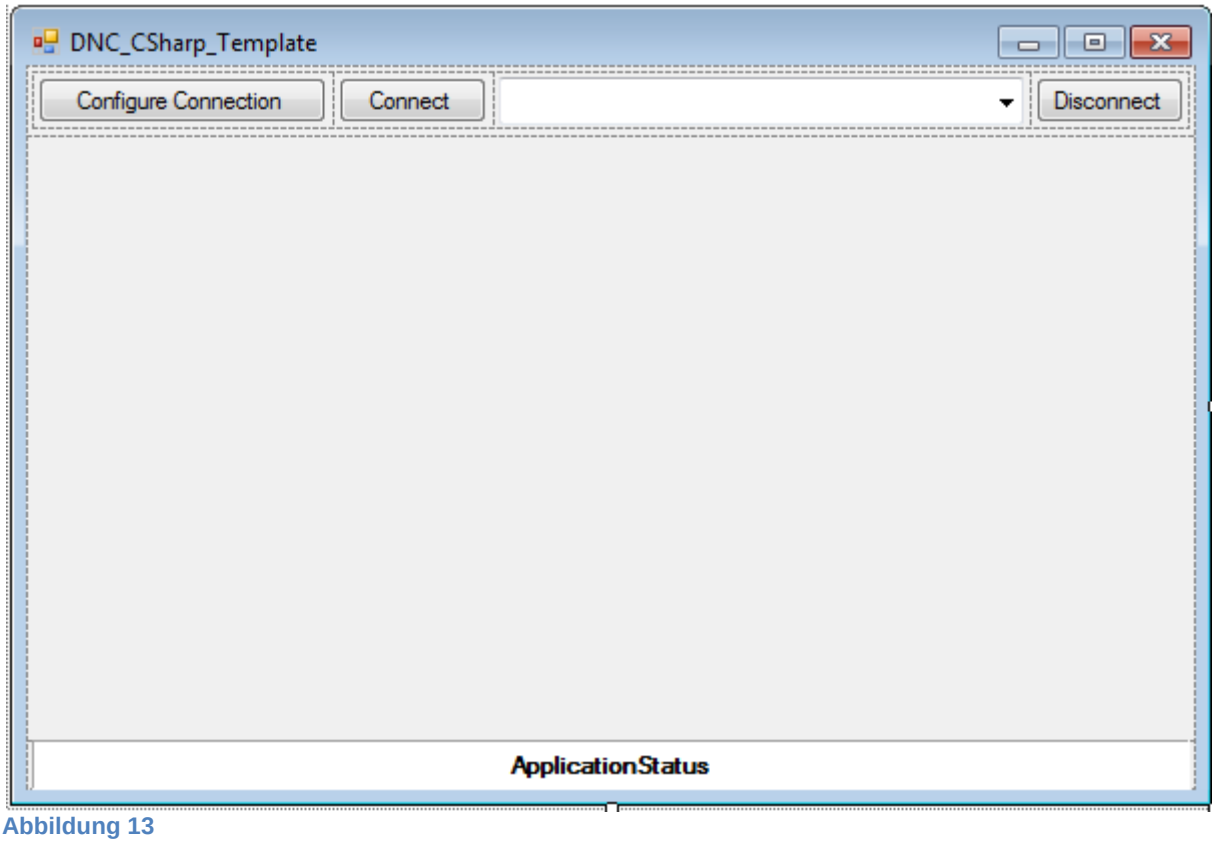

Abbildung 13

6 Verbindungshandling

6.1 Namensraum

Zur Verwendung der HeidenhainDNC COM Komponente werden Methoden aus dem Namensraum System.Runtime.InteropServices benötigt. Zur Unterstützung bei der Fehlersuche verwenden wir zusätzlich die Namensräume System.Diagnostics und System.Reflection. Fügen Sie diese Namensräume der Datei FrmMain.cs hinzu.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
using System.Reflection;
using System.Runtime.InteropServices;
```

6.2 Eigenschaften und Felder

Im der Klasse des Hauptfensters werden einige Variablen benötigt.
Bitte deklarieren Sie diese direkt am Kopf der Klasse (FrmMain.cs) des Hauptfensters.
Für die Variable m_DncConnectionState wird noch ein Property hinzugefügt (get / set Methode)

Variablentyp	Bezeichner	Zweck
`static readonly Color`	`JHGREEN`	Grüne Farbe für den Verbindungsstatus
`HeidenhainDNCLib.JHMachine`	`m_Machine`	Hauptschnittstelle für das Verbindungshandling mit der CNC
`ApplicationState`	`m_AppState`	Verbindungsstatus der Anwendung zur CNC

TIPP:
Mit den Schlüsselworten #region „{Region Name}“ und #Endregion können zusammengehörige Eigenschaften und Methoden gruppiert werden. Somit können z.B. alle Eventhandler Methoden oder privaten Methoden usw. zusammengefasst werden.

```
public partial class FrmMain : Form
{
  #region "type defs"
  public enum ApplicationState { disconnected, connecting, connected, disconnecting }
  public static readonly Color JHGREEN = Color.FromArgb(176, 203, 36);
  #endregion

  #region "fields"
  private HeidenhainDNCLib.JHMachine m_Machine = null;
  private ApplicationState m_AppState = ApplicationState.disconnected;
  #endregion

  #region "properties"
  private ApplicationState AppState
  {
    get
    {
      return m_AppState;
    }
    set
    {
      m_AppState = value;
```

```
      UpdateGUI();  // always update gui after changing connection state
    }
  }
  #endregion
```

6.3 Private Methoden

6.3.1 Aktualisieren der Benutzeroberfläche

Fügen Sie eine neue private Methode in der Klasse FrmMain ein, um die Benutzeroberfläche zu aktualisieren.

```
private void UpdateGUI()
{
  this.ResumeLayout();
  // --- update dnc connection state ---------------------------------------
  switch (AppState)
  {
    case ApplicationState.connecting:
      lblApplicationStatus.BackColor = Color.Yellow;
      lblApplicationStatus.Text = "connecting";
      cbxConnectionList.Enabled = false;
      btnConfigureConnection.Enabled = false;
      btnConnect.Enabled = false;
      btnDisconnect.Enabled = true;
      break;
    case ApplicationState.connected:
      lblApplicationStatus.BackColor = JHGREEN;
      lblApplicationStatus.Text = "connected";
      cbxConnectionList.Enabled = false;
      btnConfigureConnection.Enabled = false;
      btnConnect.Enabled = false;
      btnDisconnect.Enabled = true;
      break;
    case ApplicationState.disconnecting:
      lblApplicationStatus.BackColor = Color.Yellow;
      lblApplicationStatus.Text = "disconnecting";
      cbxConnectionList.Enabled = false;
      btnConfigureConnection.Enabled = false;
      btnConnect.Enabled = false;
      btnDisconnect.Enabled = true;
      break;
    case ApplicationState.disconnected:
      lblApplicationStatus.BackColor = Color.Red;
      lblApplicationStatus.Text = "disconnected";
      cbxConnectionList.Enabled = true;
      btnConfigureConnection.Enabled = true;
      btnConnect.Enabled = true;
      btnDisconnect.Enabled = false;
      break;
  }
  lblApplicationStatus.Refresh();
}
```

6.3.2 Verbindungsabbau

Da der Disconnect von mehreren Stellen ausgeführt werden soll, lagern wir ihn ebenfalls in eine eigene private Methode aus.

```
private void Disconnect()
{
  try
  {
    if (m_Machine != null)
    {
      AppState = ApplicationState.disconnecting;
```

```
      // --- 1. disconnect from TNC ---------------------------------------
      m_Machine.Disconnect();
      AppState = ApplicationState.disconnected;
      // --- 2. release machine object ------------------------------------
      Marshal.ReleaseComObject(m_Machine);
      m_Machine = null;
    }
  }
  catch (COMException cex)
  {
    AppState = ApplicationState.connected;
    string strClassName = this.GetType().Name;
    string strMethodName = MethodInfo.GetCurrentMethod().Name;
    MyHelper.ShowComException(cex.ErrorCode, strClassName, strMethodName);
  }
}
```

6.3.3 Aktualisierung der Verbindungen in der ComboBox

In der ComboBox im Kopf unseres Hauptfensters werden alle konfigurierten Verbindungen angezeit. Diese müssen beim Start der Anwendung und nach jedem konfigurieren (Configure Connctions“ erneut ausgelesen werden. Dies übernimmt die private Methode UpdateConnectionList. Über den String selectedConnection kann, falls erforderlich, eine Verbindung in der Combobox vorgewählt werden.

```
private void UpdateConnectionList(string selectedConnection)
{
  cbxConnectionList.Items.Clear();

  HeidenhainDNCLib.JHMachine machine = null;
  HeidenhainDNCLib.IJHConnectionList connectionList = null;
  HeidenhainDNCLib.IJHConnection connection = null;
  try
  {
    // --- list all configured connection in combo box --------------------
    machine = new HeidenhainDNCLib.JHMachine();
    connectionList = machine.ListConnections();

    cbxConnectionList.BeginUpdate();
    for (int i = 0; i < connectionList.Count; i++)
    {
      connection = connectionList[i];

      cbxConnectionList.Items.Add(connection.name);

      if (connection != null)
        Marshal.ReleaseComObject(connection);
    }

    // --- preselect connection in combobox -------------------------------
    if (cbxConnectionList.Items.Count > 0)
    {
      if (String.IsNullOrEmpty(selectedConnection))
      {
        cbxConnectionList.SelectedIndex = 0;
      }
      else
      {
        cbxConnectionList.SelectedIndex =
cbxConnectionList.FindStringExact(selectedConnection);
      }
      cbxConnectionList.EndUpdate();
    }
  }
  catch (COMException cex)
```

```
  {
    string strClassName = this.GetType().Name;
    string strMethodName = MethodInfo.GetCurrentMethod().Name;
    MyHelper.ShowComException(cex.ErrorCode, strClassName, strMethodName);
  }
  catch (Exception ex)
  {
    string strClassName = this.GetType().Name;
    string strMethodName = MethodInfo.GetCurrentMethod().Name;
    MyHelper.ShowException(ex, strClassName, strMethodName);
  }
  finally
  {
    if (machine != null)
      Marshal.ReleaseComObject(machine);
    if (connectionList != null)
      Marshal.ReleaseComObject(connectionList);
    if (connection != null)
      Marshal.ReleaseComObject(connection);
  }
}
```

6.4 Eventhandler

Die Eventhandler Methoden der Steuerelemente können ebenfalls über den Designer hinzugefügt werden. Das wird über die Properties Ansicht durchgeführt. Dazu wählen Sie zuerst im Kopf der Properties Ansicht das entsprechende Steuerelement aus und stellen die Ansicht dann von Properties auf Events um (siehe Blitz). Bitte Fügen Sie alle in den folgenden Tabellen angegebenen Eventhandle Methoden hinzu.

TIPP:
Durch einen Doppelklick in die rechte Spalte (Abbildung 14) erzeugt Visual Studio automatisch einen passenden Namen für die Eventhandler Methode.

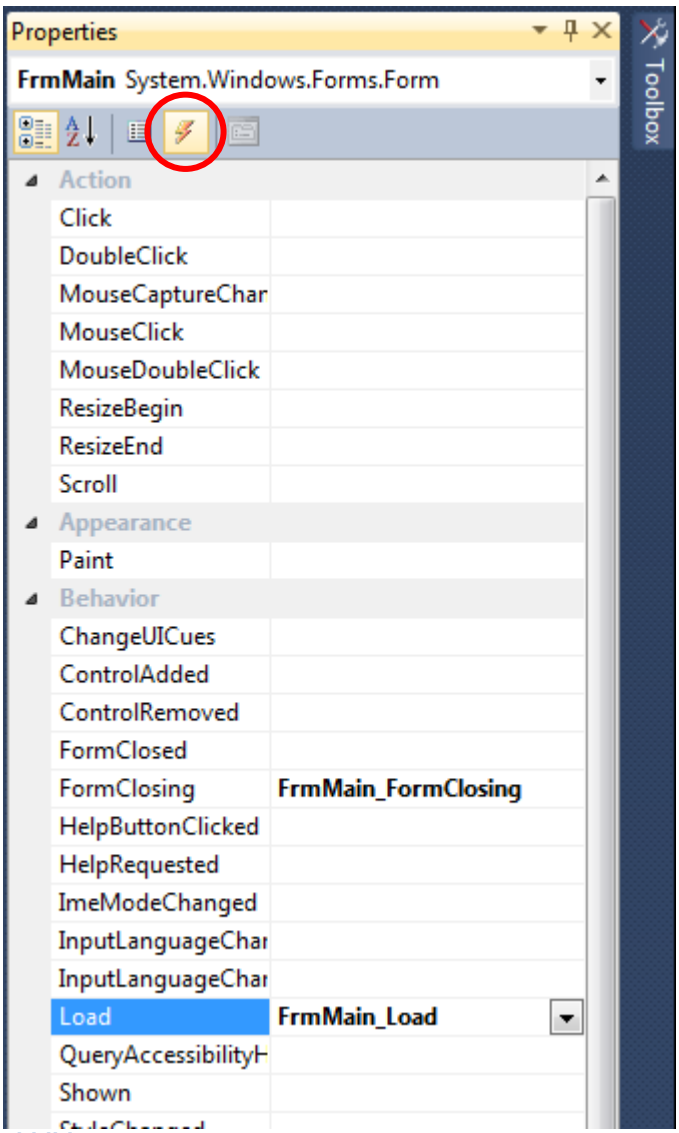

Abbildung 14

Steuerelement: FrmMain

Ereigniss	Handler Methode
FormClosing	FrmMain_FormClosing
Load	FrmMain_Load

Steuerelement: btnConnect

Ereigniss	Handler Methode
Click	btnConnect_Click

Steuerelement: btnDisonnect

Ereigniss	Handler Methode

Click	btnDisonnect_Click

Steuerelement: btnConfigureConnection

Ereigniss	Handler Methode
Click	btnConfigureConnection _Click

6.4.1 Konfigurieren der Verbindungen

Über den Button „Configure Connection“ können Sie einen Dialog zur Einsicht, Manipulation oder zum Anlegen neuer Verbindungen öffnen. Den Namen der in diesem Dialog selektierten Verbindung können Sie im Objekt „oConnectionName“ abfragen. In diesem Beispiel wird die Verbindung in der ComboBox über „UpdateConnectionList“ vorgewählt.

```
#region "event handler methods"
private void btnConfigureConnection_Click(object sender, EventArgs e)
{
  HeidenhainDNCLib.JHMachine machine = null;
  try
  {
    machine = new HeidenhainDNCLib.JHMachine();
    // object claimed by C# COM wrapper (ConnectionName)
    object oConnectionName = cbxConnectionList.SelectedIndex.ToString();
    this.Enabled = false;   // Disable the dialog while the Connections dialog is active
    // use "ref" in front of second parameter for allowing the method ConfigureConnection to
read and write
    // value of objConnectionName

    machine.ConfigureConnection(HeidenhainDNCLib.DNC_CONFIGURE_MODE.DNC_CONFIGURE_MODE_ALL,
ref oConnectionName);
    this.Enabled = true;    // Enable the dialog again
    // check if a connection was chosen (only possible if second parameter of
ConfigureConnection is "ref")
    string strConnectionName = oConnectionName.ToString();
    UpdateConnectionList(strConnectionName);
  }
  catch (COMException cex)
  {
    string strClassName = this.GetType().ToString();
    string strMethodName = MethodInfo.GetCurrentMethod().Name;
    MyHelper.ShowComException(cex.ErrorCode, strClassName, strMethodName);
  }
  catch (Exception ex)
  {
    string strClassName = this.GetType().ToString();
    string strMethodName = MethodInfo.GetCurrentMethod().Name;
    MyHelper.ShowException(ex, strClassName, strMethodName);
  }
  finally
  {
    if (machine != null)
      Marshal.ReleaseComObject(machine);
  }
}
```

6.4.2 Verbindungsaufbau

Es wird versucht eine Verbindung zur in der ComboBox angewählten Steuerung aufzubauen.

```
private void btnConnect_Click(object sender, EventArgs e)
{
  try
  {
    AppState = ApplicationState.connecting;
    string strSelectedConnection = cbxConnectionList.Text;
    m_Machine = new HeidenhainDNCLib.JHMachine();
```

```
    m_Machine.Connect(strSelectedConnection);
    AppState = ApplicationState.connected;
  }
  catch (COMException cex)
  {
    AppState = ApplicationState.disconnected;
    string strClassName = this.GetType().Name;
    string strMethodName = MethodInfo.GetCurrentMethod().Name;
    MyHelper.ShowComException(cex.ErrorCode, strClassName, strMethodName);
  }
}
```

6.4.3 Verbindungsabbau

Es wird versucht die aktuelle Verbindung wieder abzubauen. Dabei werden alle während der Verbindung angeforderten Ressourcen wieder frei gegeben. Siehe dazu auch Kapitel 6.3.2 (Verbindungsabbau), beziehungsweise Kapitel 7.2.3 (Änderungen im Verbindungsabbau).

```
private void btnDisconnect_Click(object sender, EventArgs e)
{
  Disconnect();
}
```

7 Verwenden der Schnittstellen

Die Schnittstellen der HeidenhainDNC COM Komponente werden wir in diesem Beispiel in ein eigenes UserControl auslagern. Das hat den Vorteil, dass man den von „Windows Forms“ vorgesehenen Mechanismus verwenden kann um beim Schließen des Hauptformulars alle darin enthaltenen Ressourcen automatisch freizugeben.

7.1 Anlegen eines UserControls für die JHAutomatic Schnittstelle

Führen Sie dazu bitte die in folgenden Bildschirmausschnitten dargestellten Anweisungen durch und legen ein UserControl mit dem Namen UCAutomatic an.

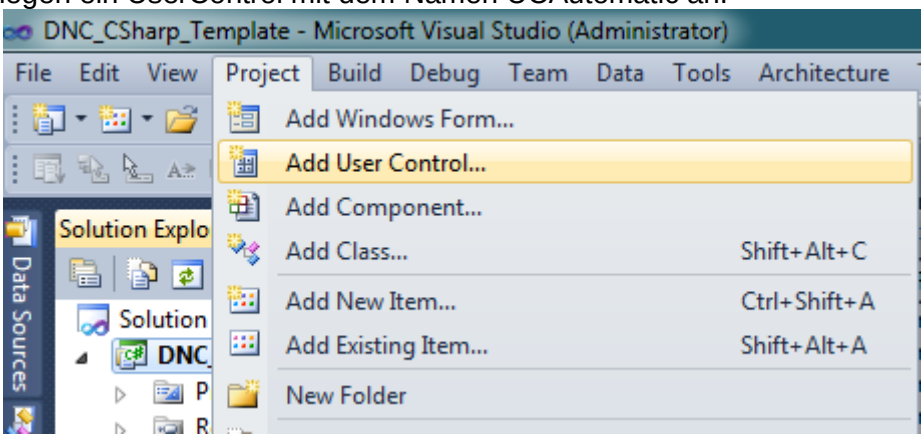

Abbildung 15

Bitte passen Sie die Eigenschaften des UserControls (UCAutomatic) wie folgt an:

Eigenschaft	Wert
(Name)	UCAutomatic
Size.Width	578
Size.Height	296

Nachdem Sie das UserControl angelegt haben, wechseln Sie bitte zum Quelltext von UCAutomatic und übernehmen folgenden Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
using System.Reflection;
using System.Runtime.InteropServices;

namespace DNC_CSharp_Template
{
  public partial class UCAutomatic : UserControl
  {
    #region "fields"
    private HeidenhainDNCLib.IJHAutomatic m_Automatic = null;
    #endregion

    #region "constructor & destructor"
    public UCAutomatic(HeidenhainDNCLib.JHMachine machine)
    {
      InitializeComponent();
      this.Disposed += new EventHandler(UCAutomatic_Disposed);
```

```
      m_Automatic =
machine.GetInterface(HeidenhainDNCLib.DNC_INTERFACE_OBJECT.DNC_INTERFACE_JHAUTOMATIC);
    }

    void UCAutomatic_Disposed(object sender, EventArgs e)
    {
      if (m_Automatic != null)
        Marshal.ReleaseComObject(m_Automatic);
    }
    #endregion
  }
}
```

7.2 Dynamisches hinzufügen des UserControls zum Hauptfenster

Die Schnittstelle IJHAutomatic kann natürlich nur dann verwendet werden, wenn eine Verbindung zur TNC besteht. Deshalb fügen wir unser UserControl (UCAutomatic) erst nach erfolgreichem Verbindungsaufbau hinzu und geben es nach dem Verbindungsabbau wieder frei.

Dafür sind folgende Änderungen im Quelltext des Hauptfensters (FrmMain.cs) durchzuführen:

7.2.1 Änderungen im Konstruktor

Um den in Kapitel 7 beschriebenen Mechanismus zum Aufräumen von zum Steuerelement gehörenden Ressourcen verwenden zu können, müssen sie den Komponenten Container initialisieren. Deklariert wird der „components“ Container immer im Designer der Windows Forms Steuerelementklassen (in diesem Fall ist das die Datei FrmMain.Designer.cs).

```
public FrmMain()
{
  InitializeComponent();
  components = new Container();
}
```

7.2.2 Änderungen im Verbindungsaufbau

```
private void btnConnect_Click(object sender, EventArgs e)
{
  try
  {
    AppState = ApplicationState.connecting;
    string strSelectedConnection = cbxConnectionList.Text;

    m_Machine = new HeidenhainDNCLib.JHMachine();
    m_Machine.Connect(strSelectedConnection);
    AppState = ApplicationState.connected;

    // --- create user control with automatic interface -------------------
    UCAutomatic automatic = new UCAutomatic(m_Machine);
    tlpMain.Controls.Add(automatic, 0, 1);
    components.Add(automatic);
  }
  catch (COMException cex)
  {
    AppState = ApplicationState.disconnected;
    string strClassName = this.GetType().Name;
    string strMethodName = MethodInfo.GetCurrentMethod().Name;
    MyHelper.ShowComException(cex.ErrorCode, strClassName, strMethodName);
  }
}
```

7.2.3 Änderungen im Verbindungsabbau

```
private void Disconnect()
{
```

```
  try
  {
    if (m_Machine != null)
    {
      AppState = ApplicationState.disconnecting;
      // --- 1. release all interfaces -----------------------------------
      components.Dispose();
      // --- 2. disconnect from TNC --------------------------------------
      m_Machine.Disconnect();
      AppState = ApplicationState.disconnected;
      // --- 3. release machine object -----------------------------------
      Marshal.ReleaseComObject(m_Machine);
      m_Machine = null;
    }
  }
  catch (COMException cex)
  {
    AppState = ApplicationState.connected;
    string strClassName = this.GetType().Name;
    string strMethodName = MethodInfo.GetCurrentMethod().Name;
    MyHelper.ShowComException(cex.ErrorCode, strClassName, strMethodName);
  }
}
```

7.3 Erstellen des UserControls

Bauen Sie wie gewohnt die Steuerelemente ein. Die Konfiguration wird in den folgenden Tabellen beschrieben.

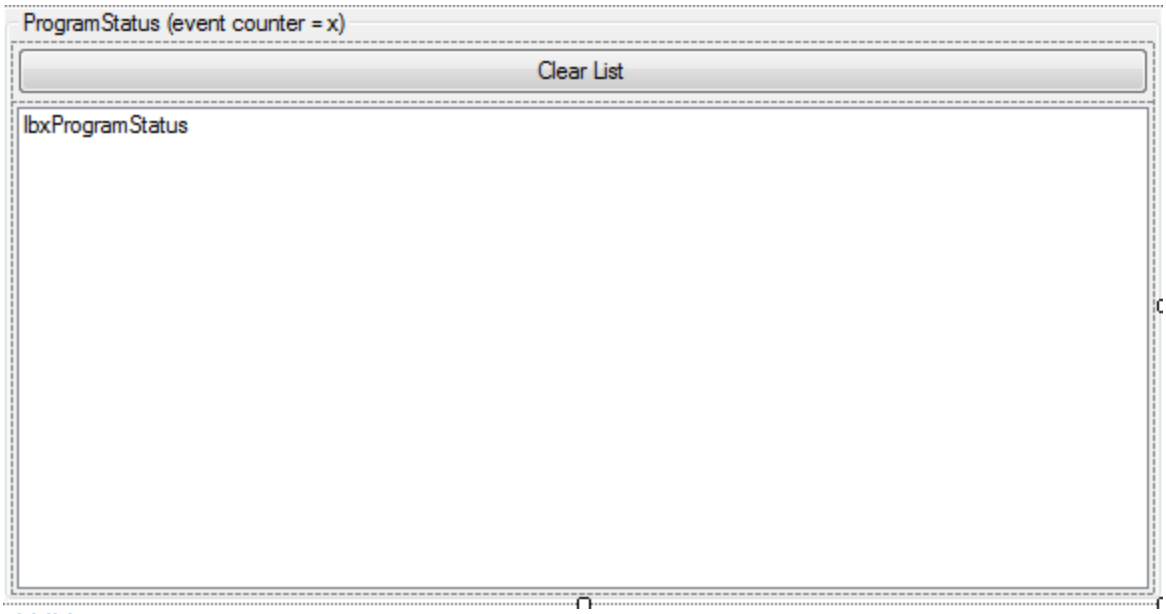

Abbildung 16

Eigenschaft	Wert
(Name)	gbxProgramStatus
Dock	Fill
Text	ProgramStatus (event counter = x)

Eigenschaft	Wert		
(Name)	tlpPogramStatus		
Dock	Fill		
ColumnCount	1		
RowCount	2		
Rows	Row1	Absolute	30
	Row2	Percent	100

Eigenschaft	Wert
(Name)	btnClearList
Dock	Fill
Text	Clear List

Eigenschaft	Wert
(Name)	lblProgramStatus
Dock	Fill
HorizontalScrollbar	True

Events	Wert
Load	UCAutomatic_Load

7.4 Verwenden der IJHAutomatik Schnittstellen

7.4.1 Eingenschaften im UCAutomatic

Um mit der IJHAutomatic Schnittstelle arbeiten zu könne, wird ein JHAutomatic Objekt benötigt. In den restlichen Variablen werden die über die Schnittstelle gelieferten Werte zwischengespeichert. Fügen sie diese „Felder“ der Datei UCAutomatic.cs hinzu.

```
#region "fields"
private HeidenhainDNCLib.JHAutomatic m_Automatic = null;
private int m_OnProgramStatusChangedCounter = 0;
private string m_ProgramStatus = null;
private string m_ProgramStatusEvent = null;
#endregion
```

7.4.2 Instanzieren

Mit dem Anmelden des Events „Disposed“ wird sichergestellt, dass ein explizites Aufräumen der Ressourcen beim Zerstören des UserControls durchgeführt werden kann. Die Implementierung finden Sie unter Kapitel 7.4.4 (Aufräumen).

```
public UCAutomatic(HeidenhainDNCLib.JHMachine machine)
{
  InitializeComponent();
  this.Dock = DockStyle.Fill;

  this.Disposed += new EventHandler(UCAutomatic_Disposed);

  try
  {
    m_Automatic =
machine.GetInterface(HeidenhainDNCLib.DNC_INTERFACE_OBJECT.DNC_INTERFACE_JHAUTOMATIC);
  }
  catch (COMException cex)
  {
    string strClassName = this.GetType().Name;
    string strMethodName = MethodInfo.GetCurrentMethod().Name;
    MyHelper.ShowComException(cex.ErrorCode, strClassName, strMethodName);
  }
}
```

7.4.3 Initialisieren

In der Eventhandler Methode des Load Events wird die Initialisierung des UserControls durchgeführt. Hier werden auch die benötigten Events angemeldet. Alle angemeldeten COM Events müssen nach Gebrauch auch wieder abgemeldet werden (-=). Das wird in den Meisten Fällen im Disposed Eventhandler durchgeführt. Siehe auch Kapitel 7.4.4 (Aufräumen).

```
private void UCAutomatic_Load(object sender, EventArgs e)
{
```

```
  m_Automatic.OnProgramStatusChanged += new
HeidenhainDNCLib._DJHAutomaticEvents_OnProgramStatusChangedEventHandler(m_Automatic_OnProgramS
tatusChanged);

  // --- init program status ------------------------------------------
  HeidenhainDNCLib.DNC_STS_PROGRAM programStatus = m_Automatic.GetProgramStatus();
  m_ProgramStatus = Enum.GetName(typeof(HeidenhainDNCLib.DNC_STS_PROGRAM), programStatus);

  UpdateGui();
}
```

7.4.4 Aufräumen

Hier müssen Sie alle innerhalb der Klasse des UserControls angeforderten COM Ressource wieder freigeben. Das gilt auch für die angemeldeten Events. Diese Eventhandler Methode wird über den „components“ Container auch automatisch aufgerufen, wenn Sie das Hauptformular „FrmMain“ schießen. Siehe dazu auch Kapitel 7.2.3.

```
private void UCAutomatic_Disposed(object sender, EventArgs e)
{
  if (m_Automatic != null)
  {
    m_Automatic.OnProgramStatusChanged -= new
HeidenhainDNCLib._DJHAutomaticEvents_OnProgramStatusChangedEventHandler(m_Automatic_OnProgramS
tatusChanged);
    Marshal.ReleaseComObject(m_Automatic);
  }
}
```

7.4.5 Abfrage des Programm Status

In der Parameterliste der Eventhandler Methode des Events OnProgramStatusChanged wird der **Eventübergang** geliefert. Mit der Methode GetProgramStatus wird der darauf folgende Status abgerufen. Die Werte werden in die String-Felder dieser Klasse gespeichert. Die Oberfläche darf nur vom selben Thread beschrieben werden, der diese auch erzeugt hat. Deshalb wird das aktualisieren der Oberfläche über den Invoke synchron an diesen Thread „delegiert“. Weitere Informationen hierzu finden Sie in Lektüren zum Thema „Windows Forms“.

```
private void m_Automatic_OnProgramStatusChanged(HeidenhainDNCLib.DNC_EVT_PROGRAM programEvent)
{
  m_OnProgramStatusChangedCounter++;
  // program status event
  m_ProgramStatusEvent = Enum.GetName(typeof(HeidenhainDNCLib.DNC_EVT_PROGRAM), programEvent);
  // program status
  HeidenhainDNCLib.DNC_STS_PROGRAM programState = m_Automatic.GetProgramStatus();
  m_ProgramStatus = Enum.GetName(typeof(HeidenhainDNCLib.DNC_STS_PROGRAM), programState);

  Invoke(new OnProgramStatusChangedHandler(UpdateGui), false);
}
```

7.4.6 Aktualisieren der Oberfläche

Über das Delegat OnProgramStatusChangedHandler wird nach Auftreten eines OnProgramStatusChangedHandler Events das aktualisieren der GUI an die private Methode UpdateGui deligiert.

```
#region "private methods"
private void UpdateGui()
{
  try
  {
    // --- set event counter -------------------------------------------------
    gbxProgramStatus.Text = "ProgramStatus (event counter = " +
Convert.ToString(m_OnProgramStatusChangedCounter) + ")";
    // --- set program status and events -------------------------------------
    lbxProgramStatus.BeginUpdate();
```

```
    if (!String.IsNullOrEmpty(m_ProgramStatusEvent))
      lbxProgramStatus.Items.Add(m_ProgramStatusEvent);

    if (!String.IsNullOrEmpty(m_ProgramStatus))
      lbxProgramStatus.Items.Add(m_ProgramStatus);
    lbxProgramStatus.EndUpdate();
  }
  catch (COMException cex)
  {
    string strClassName = this.GetType().Name;
    string strMethodName = MethodInfo.GetCurrentMethod().Name;
    MyHelper.ShowComException(cex.ErrorCode, strClassName, strMethodName);
  }
    }
#endregion
```

8 Inbetriebnahme

8.1 Anlegen einer Verbindung

8.2 Starten eines Programmierplatzes

8.3 Verbindungsaufbau

8.4 Prüfen der Methoden und Events

8.5 Verbindungsabbbau