COPA-DATA
do it your way

zenon Handbuch

Controls

v.8.00

©2018 Ing. Punzenberger COPA-DATA GmbH

Alle Rechte vorbehalten.

Die Weitergabe und Vervielfältigung dieses Dokuments ist - gleich in welcher Art und Weise - nur mit schriftlicher Genehmigung der Firma COPA-DATA gestattet. Technische Daten dienen nur der Produktbeschreibung und sind keine zugesicherten Eigenschaften im Rechtssinn. Änderungen - auch in technischer Hinsicht - vorbehalten.

Inhaltsverzeichnis

1. Willkommen bei der COPA-DATA Hilfe 5

2. Controls 5

3. Allgemein 6

3.1 Auf zenon API zugreifen 6

3.2 Methoden 8

3.2.1 CanUseVariables 8

3.2.2 MaxVariables 9

3.2.3 VariableTypes 9

3.2.4 zenonExit 9

3.2.5 zenonExitEd 10

3.2.6 zenonInit 10

3.2.7 zenonInitEd 10

4. ActiveX 10

4.1 ActiveX Elemente entwickeln 11

4.1.1 Methoden 11

4.2 Beispiel LatchedSwitch (C++) 13

4.2.1 Schnittstelle 14

4.2.2 Control 15

4.2.3 Methoden 17

4.2.4 Bedienung und Darstellung 20

4.2.5 zenon Interface 22

4.3 Beispiel CD_SliderCtrl (C++) 23

4.3.1 Schnittstelle 23

4.3.2 Control 24

4.3.3 Methoden 26

4.3.4 Bedienung und Darstellung 29

4.3.5 zenon Interface 30

4.4 Beispiel .NET Control als ActiveX (C#) 30

4.4.1 Windows Form Control erzeugen 31

4.4.2 .NET User Control in Dual Control umwandeln 34

4.4.3 Im Editor über VBA mit ActiveX arbeiten ... 39
4.4.4 zenon Variablen mit dem .NET User Control verbinden ... 40

5. .NET User Controls ... 44

5.1 Unterschiede Nutzung .NET Control in Control Container oder ActiveX ... 44
5.2 Beispiel .NET Control Container ... 45
5.2.1 Allgemeines ... 45
5.2.2 .Net User Control erstellen ... 47
5.2.3 CD_DotNetControlContainer und .NET User Control einfügen ... 56
5.2.4 Zugriff auf das User Control über VSTA oder VBA ... 61
5.3 Beispiel .NET Control als ActiveX (C#) ... 64
5.3.1 Windows Form Control erzeugen ... 64
5.3.2 .NET User Control in Dual Control umwandeln ... 67
5.3.3 Im Editor über VBA mit ActiveX arbeiten ... 72
5.3.4 zenon Variablen mit dem .NET User Control verbinden ... 73

6. WPF ... 77

1. Willkommen bei der COPA-DATA Hilfe

ALLGEMEINE HILFE

Falls Sie in diesem Hilfekapitel Informationen vermissen oder Wünsche für Ergänzungen haben, wenden Sie sich per E-Mail an documentation@copadata.com.

PROJEKTUNTERSTÜTZUNG

Unterstützung bei Fragen zu konkreten eigenen Projekten erhalten Sie vom Support-Team, das Sie per E-Mail an support@copadata.com erreichen.

LIZENZEN UND MODULE

Sollten Sie feststellen, dass Sie weitere Module oder Lizenzen benötigen, sind unsere Mitarbeiter unter sales@copadata.com gerne für Sie da.

2. Controls

In zenon können eigene Controls eingebunden werden. Dafür stehen zur Verfügung:

- .NET User Controls (auf Seite 44) (Implementierung in zenon siehe auch .NET Controls im Handbuch Bilder.)
- ActiveX (auf Seite 10) (Implementierung in zenon siehe auch ActiveX im Handbuch Bilder.)
- WPF

> **Info**
>
> *Informationen zur Nutzung der zenon Programmierschnittstellen (PCE, VBA, VSTA) finden Sie im Handbuch Programmierschnittstellen.*

Achtung

Fehler in Anwendungen wie ActiveX, PCE, VBA, VSTA, WPF und externen Anwendungen, die über die API auf zenon zugreifen, können auch die Stabilität der Runtime beeinflussen.

3. Allgemein

Controls für zenon können über ActiveX, .NET und WPF realisiert werden. Über VBA/VSTA kann auf die zenon API zugegriffen werden.

3.1 Auf zenon API zugreifen

Unter zenon kann ein ActiveX Control um spezielle Funktionen erweitert werden, um darin auf die zenon API zuzugreifen.

AUF DIE ZENON API ZUGREIFEN

- wählen Sie in den **Projekt References** mit **Add References...** die **zenon Runtime Objektbibliothek** aus
- fügen Sie die erweiterten Funktionen in den Klassencode des Controls ein

ERWEITERTE ZENON ACTIVEX FUNKTIONEN

```
// Wird beim Initialisieren des Controls in der zenon Runtime aufgerufen.
public bool zenonInit(zenon.Element dispElement)…
// Wird beim Zerstören des Controls in der zenon Runtime aufgerufen.
public bool zenonExit()…
// Unterstützt das Control Variablenverknüpfungen.
public short CanUseVariables()…
// Com Control unterstützte Datentypen.
public short VariableTypes()…
// Maximale Anzahl der Variablen, die mit dem Control verknüpft werden können.
```

public short MaxVariables()…

BEISPIEL

Das COM Object einer zenon Variable wird in einem Member zwischengespeichert, um später im Paint Event des Controls darauf zugreifen zu können:

```
zenon.Variable m_cVal = null;
    public bool zenonInit(zenon.Element dispElement)
    {
      if (dispElement.CountVariable > 0)  {
        try {
          m_cVal = dispElement.ItemVariable(0);
          if (m_cVal != null) {
            object obRead = m_cVal.get_Value((object)-1);
            UserText = obRead.ToString();
          }
        }catch { }
      }
      return true;
    }
    public bool zenonExit()
    {
      try  {
        if (m_cVal != null) {
          System.Runtime.InteropServices.Marshal.ReleaseComObject(m_cVal);
          m_cVal = null;
        }
      }
      catch { }
      return true;
    }

    public short CanUseVariables()
    {
      return 1; // die Variablen werden unterstützt
    }

    public short VariableTypes()
    {
      return short.MaxValue; // alle Daten Typen werden unterstützt
    }
```

```
public short MaxVariables()
{
  return 1; // maximal eine Variable soll mit dem Control Verknüpft werden
}

private void SamplesControl_Paint(object sender, PaintEventArgs e)
{
  // zenon Variables has changed
  try {
    if (m_cVal != null) {
      object obRead = m_cVal.get_Value((object)-1);
      UserText = obRead.ToString();
    }
  } catch { }
}
```

3.2 Methoden

ActiveX- und .NET-Controls, die zenon Variablen verwenden, benötigen bestimmte Methoden.

3.2.1 CanUseVariables

Prototyp: **`short CanUseVariables();`**

Diese Methode gibt `1` oder `0` zurück

Wert	Beschreibung
1:	Das Control kann zenon Variablen verwenden. Es können für das Dynamische Element (über den Button **Variable**) nur zenon Variablen mit den über die Methode **`VariableTypes`** (auf Seite 9) angegebenen Typen in der von der Methode **`MaxVariables`** (auf Seite 9) angegebenen Anzahl angegeben werden.
0:	Das Control kann zenon Variablen nicht verwenden oder besitzt diese Methode nicht. Es können Variablen mit allen Typen ohne Beschränkung der Anzahl angegeben werden. Diese können in der Runtime aber nur über VBA verwendet werden.

3.2.2 MaxVariables

Prototyp: `short MaxVariables();`

Hier wird die Anzahl der Variablen festgelegt, die aus der Variablenliste ausgewählt werden können.

Wird `1` zurückgegeben, so wird in der Variablenliste die Mehrfachauswahl deaktiviert. Es erfolgt eine Warnung, wenn trotzdem mehrere Variablen ausgewählt werden.

3.2.3 VariableTypes

Prototyp: `short VariableTypes();`

Der von dieser Methode zurückgegebene Wert wird als Maske für die verwendbaren Variablentypen in der Variablenliste verwendet. Der Wert ist eine **UND**-Verknüpfung aus folgenden Werten (definiert in `zenon32/dy_type.h`):

Wert 1	Wert 2	Entsprechung
WORT	0x0001	Stelle 0
BYTE	0x0002	Stelle 1
BIT	0x0004	Stelle 2
DWORT	0x0008	Stelle 3
FLOAT	0x0010	Stelle 4
DFLOAT	0x0020	Stelle 5
STRING	0x0040	Stelle 6
EIN_AUSGABE	0x8000	Stelle 15

3.2.4 zenonExit

Prototyp: `boolean zenonExit();`

Diese Methode wird von der zenon Runtime beim Schließen des ActiveX Controls aufgerufen.

Hier sollten alle Dispatch Pointer auf Variablen freigegeben werden.

3.2.5 zenonExitEd

Entspricht zenonExit (auf Seite 9) und wird beim Schließen des ActiveX im Editor ausgeführt.

Damit kann auch im Editor auf Änderungen im ActiveX, z. B. Wertänderungen, reagiert werden.

Info: Steht aktuell nur für ActiveX zur Verfügung.

3.2.6 zenonInit

Prototyp: `boolean zenonInit(IDispatch*dispElement);`

Mit Hilfe dieser Methode wird in der Runtime dem ActiveX Control ein Zeiger auf das Dispatch Interface des Dynamischen Elements übergeben. Über diesen Zeiger können dann die mit dem Dynamischen Element verknüpften zenon Variablen angesprochen werden.

Die Reihenfolge der übergebenen Variablen definieren Sie in der Konfiguration des ActiveX Elements mit den Schaltflächen **Nach unten** oder **Nach oben**.
Der **Element-Eingabe** Dialog öffnet sich nachdem Sie das ActiveX Element doppelklicken oder in den Element Eigenschaften im Knoten **Darstellung** die Eigenschaft **ActiveX Einstellungen** wählen.

3.2.7 zenonInitEd

Entspricht zenonInit (auf Seite 10) und wird beim Öffnen des ActiveX (Doppelklick auf das ActiveX) im Editor ausgeführt.

Info: Steht aktuell nur für ActiveX zur Verfügung.

4. ActiveX

Mit ActiveX kann die Funktionalität der zenon Runtime und des Editors selbstständig erweitert werden.

In diesem Handbuch lesen Sie:

- ActiveX Elemente entwickeln (auf Seite 11)
- Beispiel LatchedSwitch (C++) (auf Seite 13)
- Beispiel CD_SliderCtrl (C++) (auf Seite 23)
- Beispiel .NET Control als ActiveX (C#) (auf Seite 30)

Informationen zum Dynamischen Element ActiveX finden Sie im Handbuch Bilder im Kapitel ActiveX.

ACTIVEX UNTER WINDOWS CE

Wenn Eine ActiveX Control unter Windows CE laufen soll, muss das **apartment** Model auf `Threading` gestellt werden. Wird es auf `Free` gestellt, wird das Control in der zenon Runtime nicht laufen.

4.1 ActiveX Elemente entwickeln

Das Dynamische Element ActiveX in zenon kann Variablen an das ActiveX Control weitergeben, ohne dass der Umweg über VBA verwendet werden muss, um das Control zu bedienen.

Das Control kann selbst definieren, wie viele zenon Variablen es verwenden kann und von welchem Typ sie sein dürfen. Die Properties des Controls können über das Dynamische Element festgelegt werden.

Dazu muss die Schnittstelle (Dispatch-Interface) des Controls eine Reihe bestimmter Methoden (auf Seite 11) unterstützen.

4.1.1 Methoden

Jedes ActiveX-Control, das zenon Variablen verwenden kann, muss folgende Methoden enthalten:

- CanUseVariables (auf Seite 8)
- MaxVariables (auf Seite 9)
- VariableTypes (auf Seite 9)
- zenonExit (auf Seite 9)
- zenonExitEd (auf Seite 10)
- zenonInit (auf Seite 10)
- zenonInitEd (auf Seite 10)

Welche Dispatch ID die Methoden in der Schnittstelle haben, ist dabei gleichgültig. Beim Aufruf der Methoden wird in zenon die richtige ID aus dem Interface geholt.

CanUseVariables

Prototyp: `short CanUseVariables();`

Diese Methode gibt `1` oder `0` zurück

Wert	Beschreibung
1:	Das Control kann zenon Variablen verwenden. Es können für das Dynamische Element (über den Button **Variable**) nur zenon Variablen mit den über die Methode `VariableTypes` (auf Seite 9) angegebenen Typen in der von der Methode `MaxVariables` (auf Seite 9) angegebenen Anzahl angegeben werden.
0:	Das Control kann zenon Variablen nicht verwenden oder besitzt diese Methode nicht. Es können Variablen mit allen Typen ohne Beschränkung der Anzahl angegeben werden. Diese können in der Runtime aber nur über VBA verwendet werden.

MaxVariables

Prototyp: `short MaxVariables();`

Hier wird die Anzahl der Variablen festgelegt, die aus der Variablenliste ausgewählt werden können.

Wird `1` zurückgegeben, so wird in der Variablenliste die Mehrfachauswahl deaktiviert. Es erfolgt eine Warnung, wenn trotzdem mehrere Variablen ausgewählt werden.

VariableTypes

Prototyp: `short VariableTypes();`

Der von dieser Methode zurückgegebene Wert wird als Maske für die verwendbaren Variablentypen in der Variablenliste verwendet. Der Wert ist eine **UND**-Verknüpfung aus folgenden Werten (definiert in `zenon32/dy_type.h`):

Wert 1	Wert 2	Entsprechung
WORT	0x0001	Stelle 0
BYTE	0x0002	Stelle 1
BIT	0x0004	Stelle 2
DWORT	0x0008	Stelle 3
FLOAT	0x0010	Stelle 4
DFLOAT	0x0020	Stelle 5
STRING	0x0040	Stelle 6
EIN_AUSGABE	0x8000	Stelle 15

zenonExit

Prototyp: `boolean zenonExit();`

Diese Methode wird von der zenon Runtime beim Schließen des ActiveX Controls aufgerufen.

Hier sollten alle Dispatch Pointer auf Variablen freigegeben werden.

zenonExitEd

Entspricht zenonExit (auf Seite 9) und wird beim Schließen des ActiveX im Editor ausgeführt.

Damit kann auch im Editor auf Änderungen im ActiveX, z. B. Wertänderungen, reagiert werden.

Info: Steht aktuell nur für ActiveX zur Verfügung.

zenonInit

Prototyp: `boolean zenonInit(IDispatch*dispElement);`

Mit Hilfe dieser Methode wird in der Runtime dem ActiveX Control ein Zeiger auf das Dispatch Interface des Dynamischen Elements übergeben. Über diesen Zeiger können dann die mit dem Dynamischen Element verknüpften zenon Variablen angesprochen werden.

Die Reihenfolge der übergebenen Variablen definieren Sie in der Konfiguration des ActiveX Elements mit den Schaltflächen **Nach unten** oder **Nach oben**.
Der **Element-Eingabe** Dialog öffnet sich nachdem Sie das ActiveX Element doppelklicken oder in den Element Eigenschaften im Knoten **Darstellung** die Eigenschaft **ActiveX Einstellungen** wählen.

zenonInitEd

Entspricht zenonInit (auf Seite 10) und wird beim Öffnen des ActiveX (Doppelklick auf das ActiveX) im Editor ausgeführt.

Info: Steht aktuell nur für ActiveX zur Verfügung.

4.2 Beispiel LatchedSwitch (C++)

Im Folgenden wird als Beispiel ein ActiveX Control beschrieben, das über zwei Bit-Variablen einen verriegelbaren Schalter realisiert. Die erste Variable repräsentiert den Schalter, die zweite Variable den Riegel. Die Schaltervariable kann über das ActiveX Control nur dann geändert werden, wenn die Riegelvariable den Wert 0 hat.

Der Zustand des Elements wird über vier Bitmaps angezeigt, die im zenon Editor über den Properties-Dialog des Controls ausgewählt werden können.

4.2.1 Schnittstelle

Das Control LatchedSwitch besitzt folgendes Dispatch Interface:

```
[ uuid(EB207159-D7C9-11D3-B019-080009FBEAA2),
helpstring(Dispatch interface for LatchedSwitch Control), hidden ]
dispinterface _DLatchedSwitch
{
        properties:
        // NOTE - ClassWizard will maintain method information here.
        // Use extreme caution when editing this section.
        //{{AFX_ODL_PROP(CLatchedSwitchCtrl)
        [id(1)] boolean SollwertDirekt;
        [id(2)] IPictureDisp* SwitchOn; // container for the bitmaps
        [id(3)] IPictureDisp* SwitchOff;
        [id(4)] IPictureDisp* LatchedOn;
        [id(5)] IPictureDisp* LatchedOff;
        //}}AFX_ODL_PROP

methods:
// NOTE - ClassWizard will maintain method information here.
// Use extreme caution when editing this section.
//{{AFX_ODL_METHOD(CLatchedSwitchCtrl)
//}}AFX_ODL_METHOD
[id(6)] short CanUseVariables();
[id(7)] short VariableTypes();
[id(8)] short MaxVariables();
[id(9)] boolean zenonInit(IDispatch* dispElement);
[id(10)] boolean zenonExit();
[id(DISPID_ABOUTBOX)] void AboutBox();
};
```

Die Properties **SwitchOn** bis **LatchedOff** nehmen die Bitmaps zur Anzeige der vier Zustände des Controls auf. Die Bitmaps selbst werden in Objekten der Klasse CPictureHolder abgelegt. Über die Eigenschaft **SollwertDirekt** wird festgelegt, ob die Eingabe von Sollwerten über einen Dialog oder direkt durch Klicken auf das Control möglich sein soll.

4.2.2 Control

Die Implementation des Controls erfolgt über die Klasse **CLatchedSwitchCtrl**. Diese Klasse besitzt als Members die **CPictureHolder**-Objekte zur Aufnahme der Bitmaps. Ausserdem werden drei DispatchDriver für das Dynamische Element und die Variablen mitgeführt:

```
class CLatchedSwitchCtrl : public COleControl
{

DECLARE_DYNCREATE(CLatchedSwitchCtrl)

// Constructor
public:

CLatchedSwitchCtrl();

// Overrides

// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CLatchedSwitchCtrl)
public:
virtual void OnDraw(CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid);
virtual void DoPropExchange(CPropExchange* pPX);
virtual void OnResetState();
virtual DWORD GetControlFlags();
//}}AFX_VIRTUAL

// Implementation
protected:

~CLatchedSwitchCtrl();

DECLARE_OLECREATE_EX(CLatchedSwitchCtrl) // Class factory and guid
DECLARE_OLETYPELIB(CLatchedSwitchCtrl) // GetTypeInfo
DECLARE_PROPPAGEIDS(CLatchedSwitchCtrl) // Property page IDs
DECLARE_OLECTLTYPE(CLatchedSwitchCtrl) // Type name and misc status

// Message maps

//{{AFX_MSG(CLatchedSwitchCtrl)
```

```
afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
//}}AFX_MSG
DECLARE_MESSAGE_MAP()

// Dispatch maps

//{{AFX_DISPATCH(CLatchedSwitchCtrl)
BOOL m_sollwertDirekt;
afx_msg void OnSollwertDirektChanged();
afx_msg LPPICTUREDISP GetSwitchOn();
afx_msg void SetSwitchOn(LPPICTUREDISP newValue);
afx_msg LPPICTUREDISP GetSwitchOff();
afx_msg void SetSwitchOff(LPPICTUREDISP newValue);
afx_msg LPPICTUREDISP GetLatchedOn();
afx_msg void SetLatchedOn(LPPICTUREDISP newValue);
afx_msg LPPICTUREDISP GetLatchedOff();
afx_msg void SetLatchedOff(LPPICTUREDISP newValue);
afx_msg short CanUseVariables();
afx_msg short VariableTypes();
afx_msg short MaxVariables();
afx_msg BOOL zenonInit(LPDISPATCH dispElement);
afx_msg BOOL zenonExit();
//}}AFX_DISPATCH
CPictureHolder m_SwitchOn;
CPictureHolder m_SwitchOff;
CPictureHolder m_LatchedOn;
CPictureHolder m_LatchedOff;


DECLARE_DISPATCH_MAP()

afx_msg void AboutBox();

// Event maps

//{{AFX_EVENT(CLatchedSwitchCtrl)
//}}AFX_EVENT
DECLARE_EVENT_MAP()
```

```
  double VariantToDouble(const VARIANT FAR *v);
  void VariantToCString(CString *c,const VARIANT FAR *v);
  BOOL IsVariantString(const VARIANT FAR *v);
  BOOL IsVariantValue(const VARIANT FAR *v);

// Dispatch and event IDs
public:

CString szVariable[2];
IElement m_dElement;
IVariable m_dLatchVar, m_dSwitchVar;


enum {
//{{AFX_DISP_ID(CLatchedSwitchCtrl)
dispidSollwertDirekt = 1L,
dispidSwitchOn = 2L,
dispidSwitchOff = 3L,
dispidLatchedOn = 4L,
dispidLatchedOff = 5L,
dispidCanUseVariables = 6L,
dispidVariableTypes = 7L,
dispidMaxVariables = 8L,
dispidZenOnInit = 9L,
dispidZenOnExit = 10L,
//}}AFX_DISP_ID
};
};
```

4.2.3 Methoden

Folgende Methoden werden verwendet:

- CanUseVariables (auf Seite 18)
- VariableTypes (auf Seite 18)
- MaxVariables (auf Seite 18)
- zenonInit (auf Seite 18)
- zenonExit (auf Seite 19)

CanUseVariables

Diese Methode gibt 1 zurück, es können also zenon Variablen verwendet werden.

```
short CLatchedSwitchCtrl::CanUseVariables()
{

return 1;
}
```

VariableTypes

Das Control kann nur mit Bit-Variablen arbeiten, daher wird `0x0004` zurückgegeben.

```
short CLatchedSwitchCtrl::VariableTypes()
{

return 0x0004; // Nur Bit-Variablen
}
```

MaxVariables

Es können zwei Variablen verwendet werden, daher wird `2` zurückgegeben.

```
short CLatchedSwitchCtrl::MaxVariables()
{

return 2; // 2 Variablen
}
```

zenonInit

In dieser Methode werden über den `Dispatchpointer` des Dynamischen Elements die `Dispatchdriver` der Variablen geholt. Über diese `Pointer` werden beim Klicken und beim Zeichnen des Controls die Variablenwerte geholt und gesetzt.

```
BOOL CLatchedSwitchCtrl::zenonInit(LPDISPATCH dispElement)
{
m_dElement = IElement(dispElement);
```

```
Element.m_lpDispatch->AddRef();
if (m_dElement.GetCountVariable() >= 2)
{
short iIndex = 0;
m_dSwitchVar = IVariable(m_dElement.ItemVariable(COleVariant(iIndex)));
m_dLatchVar = IVariable(m_dElement.ItemVariable(COleVariant(++iIndex)));
}
return TRUE;
}
```

Info

Element.m_lpDispatch->AddRef();

Nicht verwendete Objekte werden nicht automatisch aus dem Speicher gelöscht. Dies muss durch die Programmierung erfolgen. Der Programmierer bestimmt ob ein Objekt - basierend auf einen Verweiszähler - entfernt werden kann.

`COM` verwendet die `IUnknow` Methoden `AddRef` und `Release` um die Verweisanzahl von Interfaces auf einem Objekt zu verwalten.

Die allgemeine Regel für den Aufruf dieser Methoden sind:

- Immer, wenn der Client einen Interface Pointer empfängt, muss `AddRef` auf dem Interface aufgerufen werden.
- Immer, wenn der Client die Verwendung des Interface Pointers beendet, muss ein `Release` aufgerufen werden.

Bei einer einfachen Implementierung wird durch einen `AddRef` Aufruf eine Zählervariable im Objekt erhöht. Jeder Aufruf eines `Release` vermindert diesen Zähler im Objekt. Wenn dieser Zähler wieder auf NULL ist, kann das Interface aus dem Speicher entfernt werden.

Ein Verweiszähler kann auch so implementiert werden, daß jede Referenz auf das Objekt (und nicht zu einem individuellen Interface) gezählt wird.

In diesem Fall rufen jedes `AddRef` und jedes `Release` Stellvertreter einer zentralen Implementierung auf das Objekt auf. Ein `Release` gibt dann das gesamte Objekt frei, wenn der Verweiszähler Null erreicht hat.

zenonExit

Diese Methode gibt die Dispatchdriver wieder frei.

```
BOOL CLatchedSwitchCtrl::zenonExit()
{

m_dElement.ReleaseDispatch();
m_dSwitchVar.ReleaseDispatch();
m_dLatchVar.ReleaseDispatch();
return TRUE;
}
```

4.2.4 Bedienung und Darstellung

Sollwert setzen

Ein Sollwert wird über einen Klick mit der linken Maustaste auf das Control gesetzt.

Ist **m_iSollwertDirekt** `0`, wird ein Dialog zur Auswahl des Sollwerts angezeigt, sonst wird der aktuelle Wert der Schaltervariable invertiert.

Hat die Riegelvariable den Wert `1`, wird nur ein `MessageBeep` ausgeführt und es kann kein Sollwert über das Control gesetzt werden.

```
void CLatchedSwitchCtrl::OnLButtonDown(UINT nFlags, CPoint point)
{

CRect rcBounds;
GetWindowRect(&rcBounds);

COleVariant coleValue((BYTE)TRUE);
BOOL bLatch = (BOOL)VariantToDouble((LPVARIANT)&m_dLatchVar.GetValue());
BOOL bSwitch = (BOOL)VariantToDouble((LPVARIANT)&m_dSwitchVar.GetValue());

if (bLatch) // Verriegelt!!!

MessageBeep(MB_ICONEXCLAMATION);
else
{

if (m_sollwertDirekt)
{
```

```
bSwitch = !bSwitch;
}
else
{

CSollwertDlg dlg;
dlg.m_iSollwert = bSwitch ? 1 : 0;
if (dlg.DoModal() == IDOK)
{

if (dlg.m_iSollwert == 2) // Toggle

bSwitch = !bSwitch;
else

bSwitch = (BOOL)dlg.m_iSollwert;
}
}
coleValue = (double)bSwitch;
m_dSwitchVar.SetValue(coleValue);
}
COleControl::OnLButtonDown(nFlags, point);
}
```

Zeichnen

Beim Zeichnen des Controls werden über die Dispatchdriver der Variablen deren Werte geholt und entsprechend eine der vier vorgegebenen Grafiken gezeichnet. Wenn sich der Wert einer Variable ändert, wird das Control über die **OnDraw**-Routine aktualisiert.

```
void CLatchedSwitchCtrl::OnDraw(CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{

CRect rcBitmap = rcBounds;
rcBitmap.NormalizeRect();

if (!m_dElement)
{

m_SwitchOn.Render(pdc, &rcBounds, &rcBounds);
```

```
return;
}

BOOL bVal1 = 0, bVal2 = 0;
VARIANT vRes;
if (m_dSwitchVar) // Variable vorhanden?
{

vRes = m_dSwitchVar.GetValue();
bVal1 = (BOOL)VariantToDouble(&vRes);
}
if (m_dLatchVar) // Variable vorhanden?
{

vRes = m_dLatchVar.GetValue();
bVal2 = (BOOL)VariantToDouble(&vRes);
}


if (bVal1 && bVal2)

m_SwitchOn.Render(pdc, rcBitmap, rcBitmap);
else if (!bVal1 && bVal2)

m_SwitchOff.Render(pdc, rcBitmap, rcBitmap);
else if (bVal1 && !bVal2)

m_LatchedOn.Render(pdc, rcBitmap, rcBitmap);
else

m_LatchedOff.Render(pdc, rcBitmap, rcBitmap);
}
```

4.2.5 zenon Interface

Damit das Dispatch Interface von zenon zum Sollwertsetzen verwendet werden kann, müssen für die Variablen und das Element Klassen angelegt werden, die von COleDispatchDriver abgeleitet sind. Diese

Klassen werden am einfachsten über den Class Wizard der Entwicklungsumgebung angelegt (Button **Add Class**, Auswahl von **From a type library**, `zenrt32.tlb` auswählen).

Bei unserem Control sind das die Klassen **IElement** und **IVariable**. Sie sind in `zenrt32.h` und `zenrt32.cpp` definiert.

4.3 Beispiel CD_SliderCtrl (C++)

Da folgende Beispiel beschreibt ein ActiveX Control, das dem Windows **SliderCtrl** gleicht. Diese Komponente kann mit einer zenon Variable verknüpft werden. Der Anwender kann über den Schieberegler den Sollwert einer Variable ändern. Ändert sich die Variable über ein anderes Dynamisches Element, wird der Slider aktualisiert.

4.3.1 Schnittstelle

Das Control **CD_SliderCtrl** besitzt folgendes Dispatch Interface:

```
[ uuid(5CD1B01D-015E-11D4-A1DF-080009FD837F),
  helpstring(Dispatch interface for CD_SliderCtrl Control), hidden
]
dispinterface _DCD_SliderCtrl
{
properties: //*** Eigenschaften des Controls

[id(1)] short TickRaster;
[id(2)] boolean ShowVertical;
[id(3)] short LineSize;

methods: //*** Methoden des Controls (für zenon-ActiveX)

[id(4)] boolean zenonInit(IDispatch* pElementInterface);
[id(5)] boolean zenonExit();
[id(6)] short VariableTypes();
[id(7)] short CanUseVariables();
[id(8)] short MaxVariables();

[id(DISPID_ABOUTBOX)] void AboutBox();
};
```

4.3.2 Control

Die Implementierung des Controls erfolgt über die Klasse CD_SliderCtrlCtrl. Diese Klasse besitzt als Member ein Windows-übliches **CSliderCtrl**, mit dem das Control als Subclass ediniert wird. Die Schnittstellen **IVariable** und **IElement** enthalten zenon Schnittstellen, die integriert werden mußten. Diese sind abgeleitet von **COleDispatchDriver**.

```
class CCD_SliderCtrlCtrl : public COleControl
{

DECLARE_DYNCREATE(CCD_SliderCtrlCtrl)
private: //*** Member-Variablen

BOOL m_bInitialized;
BOOL m_bShowVertical;
BOOL m_bTicksBoth;
long m_nRangeStart;
long m_nRangeEnd;
long m_nTickOrientation;
IVariable m_interfaceVariable;
IElement m_interfaceElement;
CSliderCtrl m_wndSliderCtrl;

public:

CCD_SliderCtrlCtrl();

//{{AFX_VIRTUAL(CCD_SliderCtrlCtrl)
public:
virtual void OnDraw (CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid);
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
virtual void DoPropExchange (CPropExchange* pPX);
virtual void OnResetState ();
//}}AFX_VIRTUAL

protected:

~CCD_SliderCtrlCtrl();
//*** Methoden zur Konvertierung von Variant
double VariantToDouble(const VARIANT FAR *vValue);
```

```
DECLARE_OLECREATE_EX(CCD_SliderCtrlCtrl) // Class factory and guid

DECLARE_OLETYPELIB (CCD_SliderCtrlCtrl) // GetTypeInfo
DECLARE_PROPPAGEIDS (CCD_SliderCtrlCtrl) // Property page IDs
DECLARE_OLECTLTYPE (CCD_SliderCtrlCtrl) // Type name and misc status


//*** Methoden für die Funktionalität des SliderCtrls
BOOL IsSubclassedControl ();
LRESULT OnOcmCommand (WPARAM wParam, LPARAM lParam);


//{{AFX_MSG(CCD_SliderCtrlCtrl)
afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
afx_msg void HScroll(UINT nSBCode, UINT nPos);
afx_msg void VScroll(UINT nSBCode, UINT nPos);
afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
//}}AFX_MSG
DECLARE_MESSAGE_MAP()


//{{AFX_DISPATCH(CCD_SliderCtrlCtrl)
afx_msg BOOL GetTickOnBothSides();
afx_msg void SetTickOnBothSides (short nNewValue);
afx_msg BOOL GetShowVertical();
afx_msg void SetShowVertical(BOOL bNewValue);
afx_msg short GetTickOrientation();
afx_msg void SetTickOrientation (short nNewValue);
afx_msg BOOL zenonInit(LPDISPATCH pElementInterface);
afx_msg BOOL zenonExit();
afx_msg short VariableTypes();
afx_msg short CanUseVariables();
afx_msg short MaxVariables();
//}}AFX_DISPATCH
DECLARE_DISPATCH_MAP()

afx_msg void AboutBox();
```

```
//{{AFX_EVENT(CCD_SliderCtrlCtrl)
//}}AFX_EVENT
DECLARE_EVENT_MAP()

public:

enum {
//{{AFX_DISP_ID(CCD_SliderCtrlCtrl)
dispidShowVertical = 1L,
dispidTicksOnBothSides = 2L,
dispidTickOrientation = 3L,
dispidZenOnInit = 4L,
dispidZenOnExit = 5L,
dispidVariableTypes = 6L,
dispidCanUseVariables = 7L,
dispidMaxVariables = 8L,
//}}AFX_DISP_ID
};
};
```

4.3.3 Methoden

Folgende Methoden werden verwendet:

- CanUseVariables (auf Seite 26)
- VariableTypes (auf Seite 27)
- MaxVariables (auf Seite 27)
- zenonInit (auf Seite 27)
- zenonExit (auf Seite 28)

CanUseVariables

Diese Methode gibt `1` zurück, es können also zenon Variablen verwendet werden.

```
short CCD_SliderCtrlCtrl::CanUseVariables()
{
```

```
return 1;
}
```

VariableTypes

Das Control kann mit Wort, Byte, Doppelwort und Float-Variablen arbeiten. Eine Auflistung der möglichen Datentypen finden Sie in der allgemeinen Beschreibung (auf Seite 9) über diese Methode.

```
short CCD_SliderCtrlCtrl::VariableTypes()
{

return 0x0001 | // Wort

0x0002 | // Byte
0x0008 | // D-Wort
0x0010 | // Float
0x0020; // D-Float
}
```

MaxVariables

Mit diesem Control kann nur eine Variable verknüpft werden.

```
short CCD_SliderCtrlCtrl::MaxVariables()
{

return 1; // 1 Variablen
}
```

zenonInit

Der Parameter **dispElement** enthält das Inteface für das Dynamische Element. Über dieses Element wird die verknüpfte zenon Variable ermittelt. Ist diese gültig, wird der Bereich des **SliderCtrls** gesetzt. Weiters werden die Einstellungen für die Darstellung (Anzahl der Ticks, ...) gesetzt. Wenn keine Variable verknüpft ist, wird ein Laufbereich von 0 bis 0 festgelegt. Somit kann das SliderCtrl nicht verändert werden. Über die Variable **m_bInitialized** wird festgelegt, dass Sollwerte ab jetzt abgesetzt werden können.

```
BOOL CCD_SliderCtrlCtrl::zenonInit(LPDISPATCH dispElement)
{
//*** Ermittle die Variable über das zenon Element
```

```
m_interfaceElement = IElement(pElementInterface);
if (m_interfaceElement.GetCountVariable() > 0) {

short nIndex = 0;
m_interfaceVariable = IVariable
(m_interfaceElement.ItemVariable(COleVariant(nIndex)));
}

//*** Initialisiere den Bereich des Slider-Ctrls
if (m_interfaceVariable) {

//*** Laufbereich festlegen
m_nRangeStart = (long) VariantToDouble(&m_interfaceVariable.GetRangeMin());
m_nRangeEnd = (long) VariantToDouble(&m_interfaceVariable.GetRangeMax());
m_wndSliderCtrl.SetRange(m_nRangeStart,m_nRangeEnd,TRUE);
//*** Nebenticks festlegen
m_wndSliderCtrl.SetTicFreq(m_nTickCount);
m_wndSliderCtrl.SetPageSize(m_nTickCount);
m_wndSliderCtrl.SetLineSize(m_nLineSize);
} else {

m_wndSliderCtrl.SetRange(0,0,TRUE);
return FALSE;
}

m_bInitialized = TRUE;
return TRUE;
}
```

zenonExit

In dieser Methode werden die zenon Schnittstellen wieder freigegeben.

```
BOOL CCD_SliderCtrlCtrl::zenonExit()
{

m_interfaceElement.ReleaseDispatch();
m_interfaceVariable.ReleaseDispatch();
return TRUE;
}
```

4.3.4 Bedienung und Darstellung

Zeichnen

Über **DoSuperclassPaint** wird das SliderCtrl gezeichnet (da es sich um gesubclassed Control handelt). Wenn im Moment des Zeichnens der Regler verschoben wird, nimmt die Variable **m_bInitialized** den Wert `FALSE` an. Damit wird sichergestellt, dass der Wert verändert werden kann. Im Normalfall wird der Wert der Variable ermittelt und über die Methode **SetPos** des SliderCtrls dargestellt.

```
void CCD_SliderCtrlCtrl::OnDraw(CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{

//*** Aktualisiere die Ansicht
DoSuperclassPaint(pdc, rcBounds);
if (m_interfaceVariable && m_bInitialized) {

COleVariant cValue(m_interfaceVariable.GetValue());
int nValue = (int) VariantToDouble(&cValue.Detach());
m_wndSliderCtrl.SetPos(nValue);
}
}
```

Sollwert setzen

In der **LButtonDown**-Methode wird die Variable **m_bInitialized** auf `FALSE` gesetzt und im **LButtonUp**-Event wieder auf `TRUE`. Damit wird sichergestellt, dass der Wert verändert werden kann. Sonst würde die **OnDraw**-Routine aufgerufen und der ursprüngliche Wert dargestellt werden.

```
void CCD_SliderCtrlCtrl::OnLButtonDown(UINT nFlags, CPoint point)
{
	m_bInitialized = FALSE;
	COleControl::OnLButtonDown(nFlags, point);
}

void CCD_SliderCtrlCtrl::OnLButtonUp(UINT nFlags, CPoint point)
{
	m_bInitialized = TRUE;
	COleControl::OnLButtonUp(nFlags, point);
}
```

Ein Sollwert wird über eine Änderung des Reglers herbeigeführt. In den Methoden **HScroll** oder **VScroll** wird der Sollwert abgesetzt (abhängig, davon, ob es sich um einen horizontalen oder einen vertikalen Schieberegler handelt).

```
void CCD_SliderCtrlCtrl::HScroll(UINT nSBCode, UINT nPos)
{

switch (nSBCode) {

case TB_LINEUP:
case TB_PAGEUP:
case TB_LINEDOWN:
case TB_PAGEDOWN:
case TB_THUMBTRACK:
case TB_THUMBPOSITION:{

//*** Sollwert ohne Dialog absetzen ?
int nValue = m_wndSliderCtrl.GetPos();
COleVariant cValue((short) nValue,VT_I2);
m_interfaceVariable.SetValue(cValue);
}
}
}
```

4.3.5 zenon Interface

Damit das Dispatch Interface von zenon zum Sollwertsetzen verwendet werden kann, müssen für die Variablen und das Element Klassen angelegt werden, die von COleDispatchDriver abgeleitet sind. Diese Klassen werden am einfachsten über den Class Wizard der Entwicklungsumgebung angelegt (Button **Add Class**, Auswahl von **From a type library**, `zenrt32.tlb` auswählen).

Bei unserem Control sind das die Klassen **IElement** und **IVariable**. Sie sind in `zenrt32.h` und `zenrt32.cpp` definiert.

4.4 Beispiel .NET Control als ActiveX (C#)

Das folgende Beispiel beschreibt ein .NET Control, das als ActiveX Control in zenon ausgeführt wird.

Die Erstellung und Integration passiert in vier Schritten:

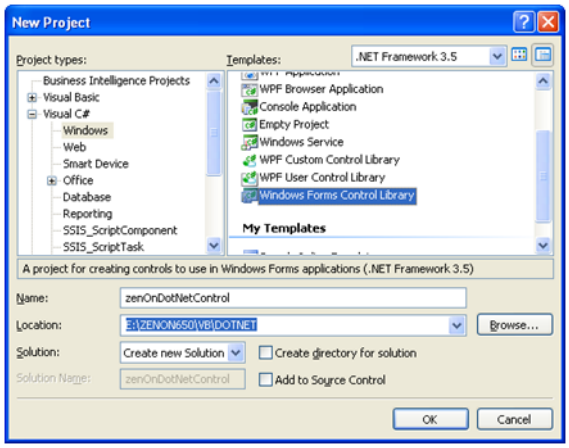
1. Windows Form Control erzeugen (auf Seite 31)
2. .NET User Control in Dual Control umwandeln (auf Seite 34)
3. Im Editor über VBA mit ActiveX arbeiten (auf Seite 39)
4. zenon Variablen mit dem .NET User Control verbinden (auf Seite 40)

> **Achtung**
>
> *Bei der Verwendung von zenon COM-Objekten in selbst erstellten User Controls oder externen Anwendungen müssen diese über die Methode* ***Marshal.ReleaseComObject*** *freigegeben werden. Die Freigabe über die Methode* ***Marshal.FinalReleaseComObject*** *darf nicht verwendet werden, da diese zu einem Ausfall von zenon Add-Ins führt.*
>
> **Info**
>
> *Die Screenshots für dieses Thema sind nur in englischer Sprache verfügbar.*

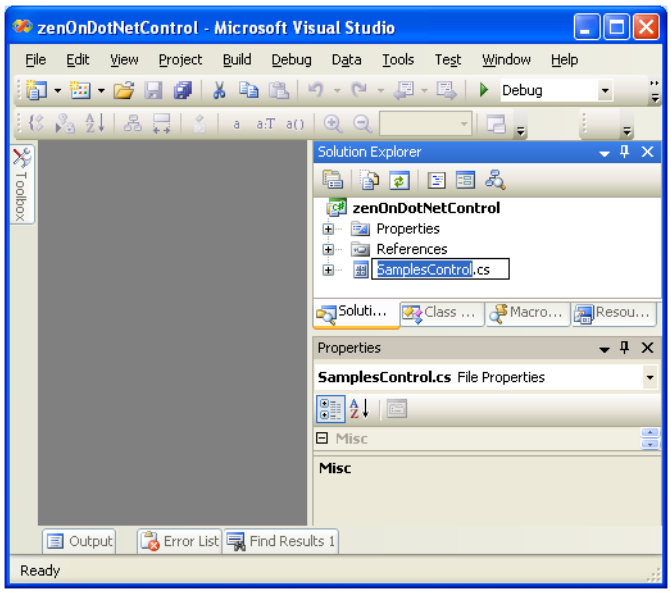
4.4.1 Windows Form Control erzeugen

Um ein Windows Form Control zu erzeugen:

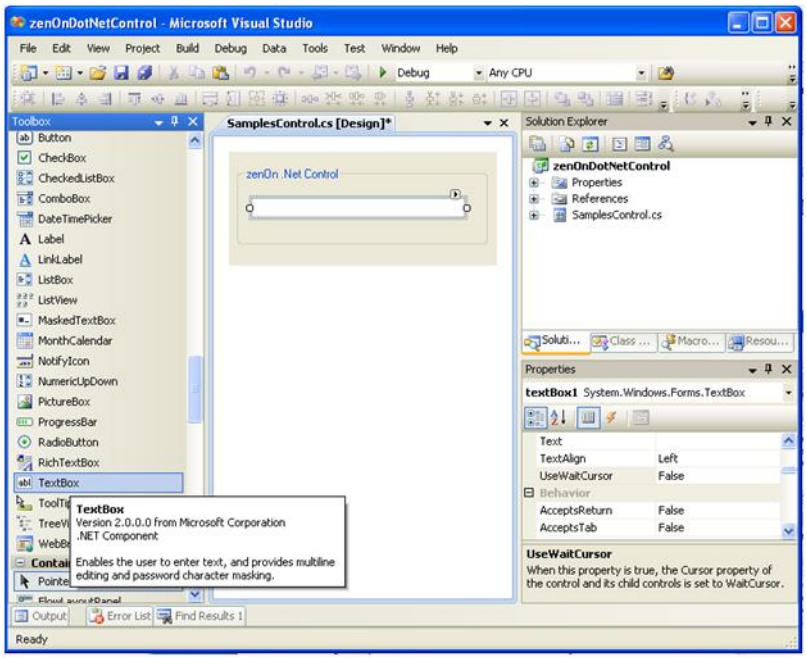
1. Starten Sie Visual Studio 2008 und erzeugen Sie ein neues Windows **Form Control Library** Projekt:



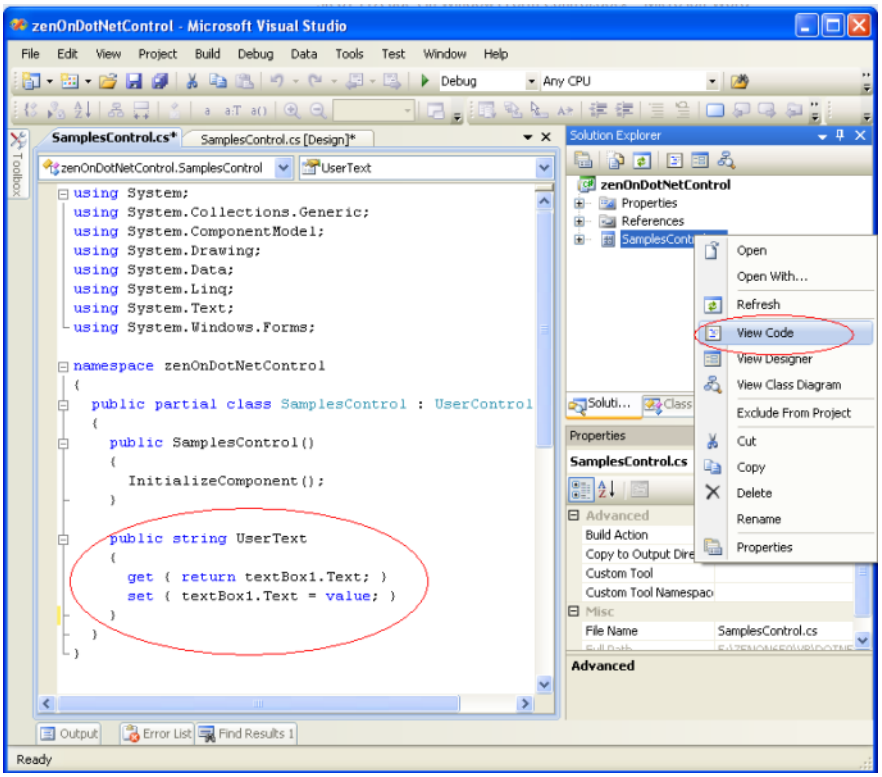
2. Benennen Sie das Default Control auf den gewünschten Controlnamen um.
In unserem Beispiel: **SampesControl.cs**.



3. Öffnen Sie den Control Designer und fügen Sie die gewünschten Controls ein, in unserem Fall eine Textbox:



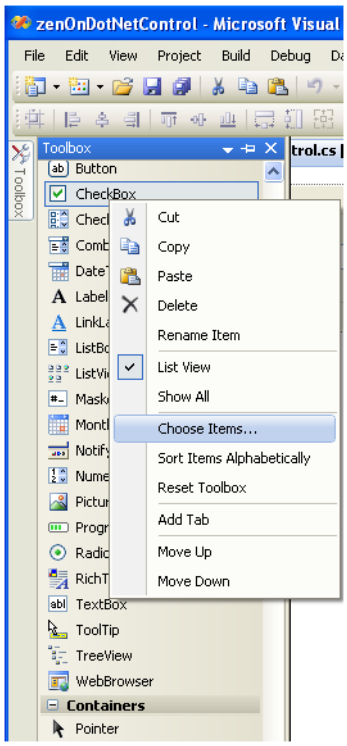
4. Controls verfügen normalerweise über Eigenschaften. Öffnen Sie den Code Designer über **View Code** und fügen Sie die gewünschten extern verfügbaren Properties hinzu.
   In unserem Beispiel: Extern sichtbares Property „**UserText**" mit **get** und **set** Zugriff, das den Text der Textbox enthält:



5. Kompilieren Sie das Projekt.

   Das Windows Forms Control kann jetzt in anderen Windows Forms Projekten verwendet werden.

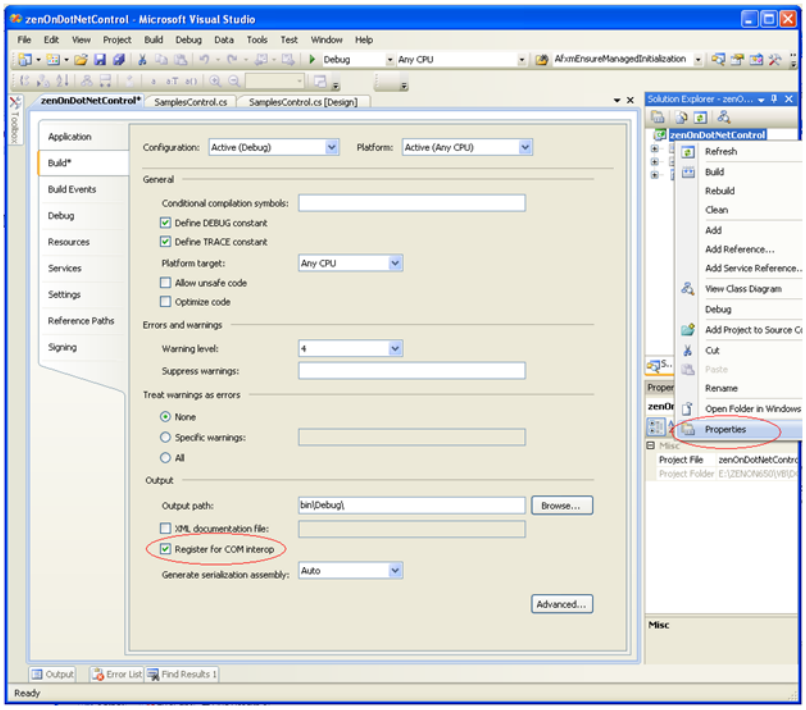
Wichtig: Das Control muss manuell über **Choose Items** in die Control Toolbox eingefügt werden.



4.4.2 .NET User Control in Dual Control umwandeln

Um das .NET in ein Dual Control umzuwandeln, muss zuerst die COM-Schnittstelle für ActiveX aktiviert werden.

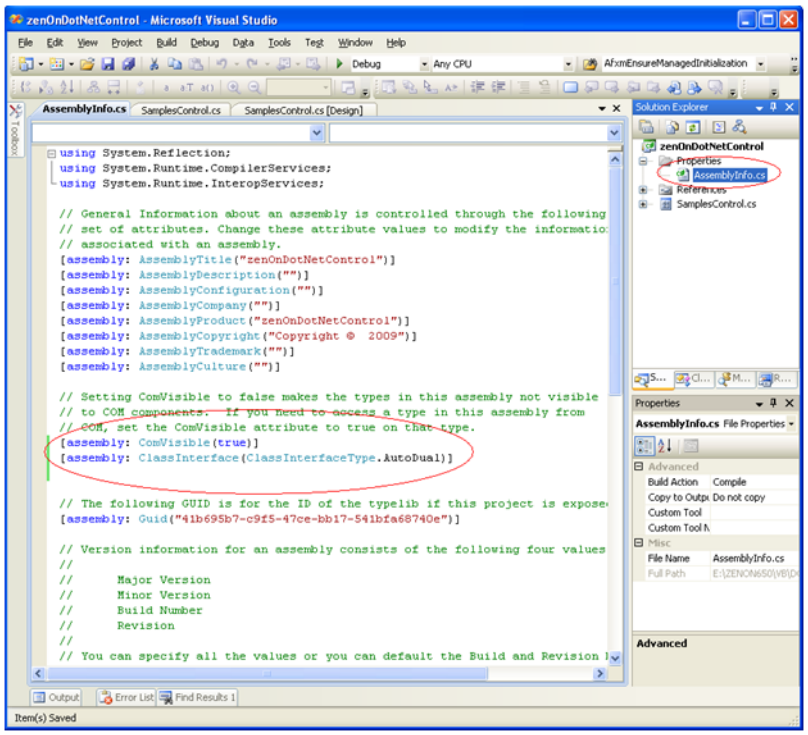
1. Öffnen Sie das Projekt und aktivieren Sie unter den **Build-**Einstellungen die Eigenschaft **Register for COM interop**:



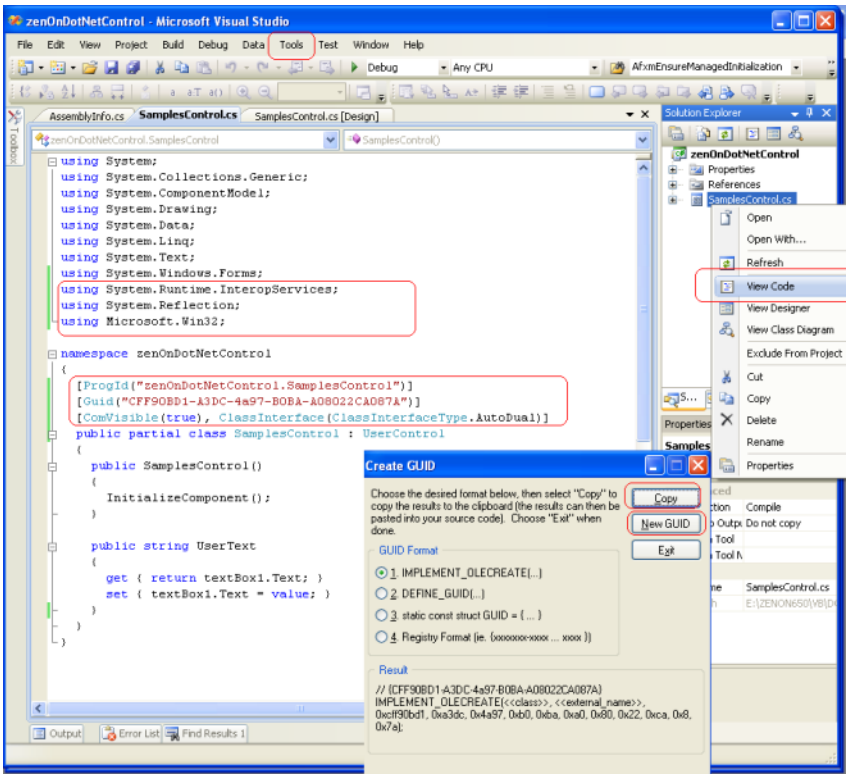
2. Öffnen Sie die Datei **AssemblyInfo.cs** und
   - setzen Sie das Attribut **ComVisible** auf `true`
   - fügen Sie das Attribut **ClassInterface** hinzu

   [`assembly`: ComVisible(`true`)]

[assembly: ClassInterface(ClassInterfaceType.AutoDual)]



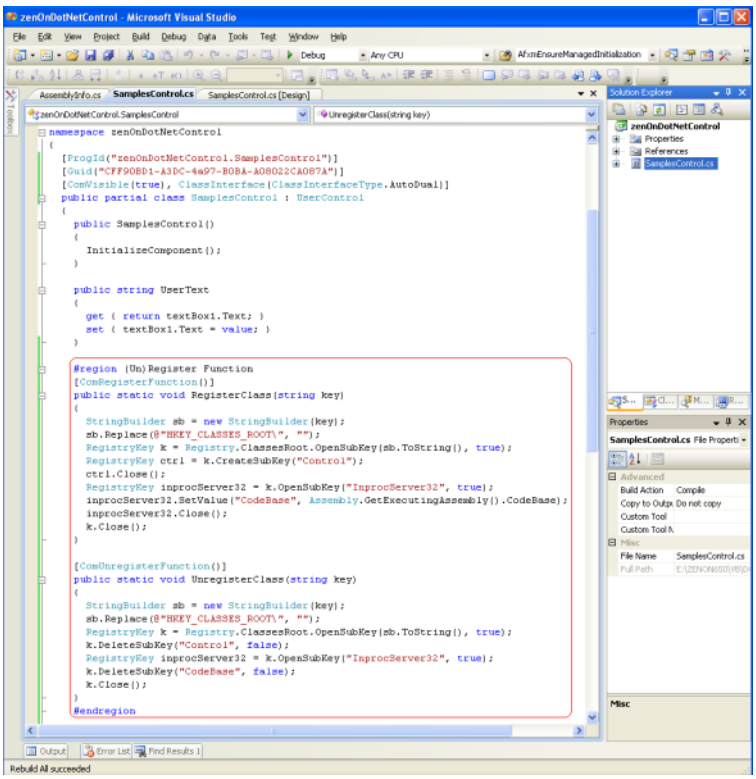
3. Öffnen Sie den Codedesigner über **View Code** und fügen Sie die notwendigen ActiveX Attribute und **using** Einträge hinzu. Erzeugen Sie über das Menü **Tools/Create GUID** eine neue GUID für das GUID-Attribut:



4. Damit das Control auch als ActiveX Oberflächen Control auswählbar ist, müssen in die Control-Klasse noch folgende Funktionen eingefügt werden:

   - RegisterClass

- UnregisterClass



Danach kann das Control in der Registry registriert werden.

5. Kompilieren Sie das Projekt erneut.

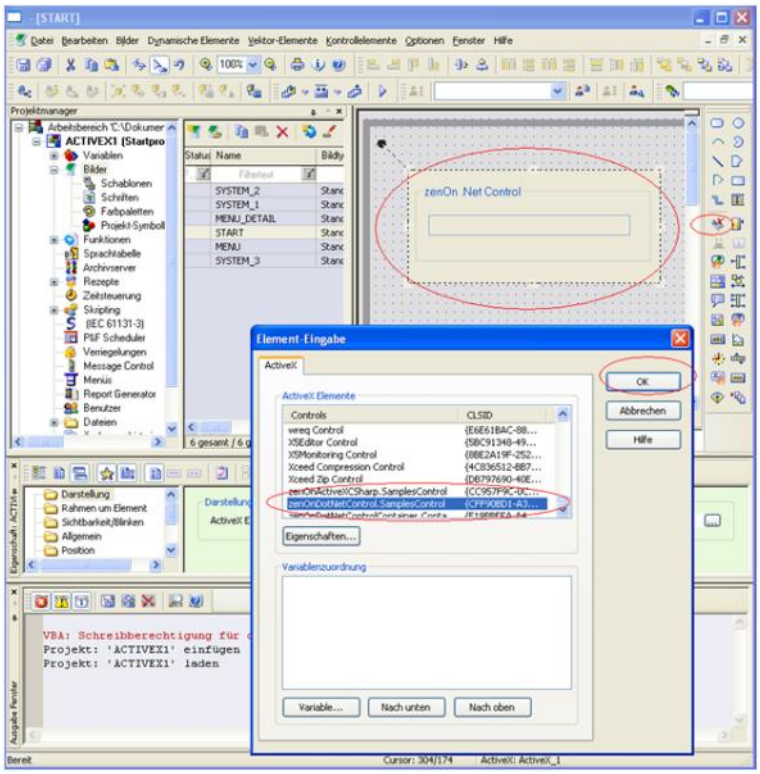
   Das Windows Form Control ist jetzt ActiveX fähig und wurde beim Rebuild automatisch registriert. Eine zusätzliche Typelib-Datei **zenOnDotNetControl.tlb** wurde im Outputverzeichnis erstellt.

6. Um das Control auf einem anderen Rechner einzusetzen:

   a) kopieren Sie die DLL-Datei und die TLB-Datei auf den Zielrechner

   b) registrieren Sie die Dateien über die Kommandozeile mit:
   %windir%\Microsoft.NET\Framework\v2.0.50727\regasm.exe zenOnDotNetControl.dll /tlb:zenOnDotNetControl.tlb

7. Fügen Sie das erweiterte Windows Form Control als ActiveX Control in den zenon Editor ein:



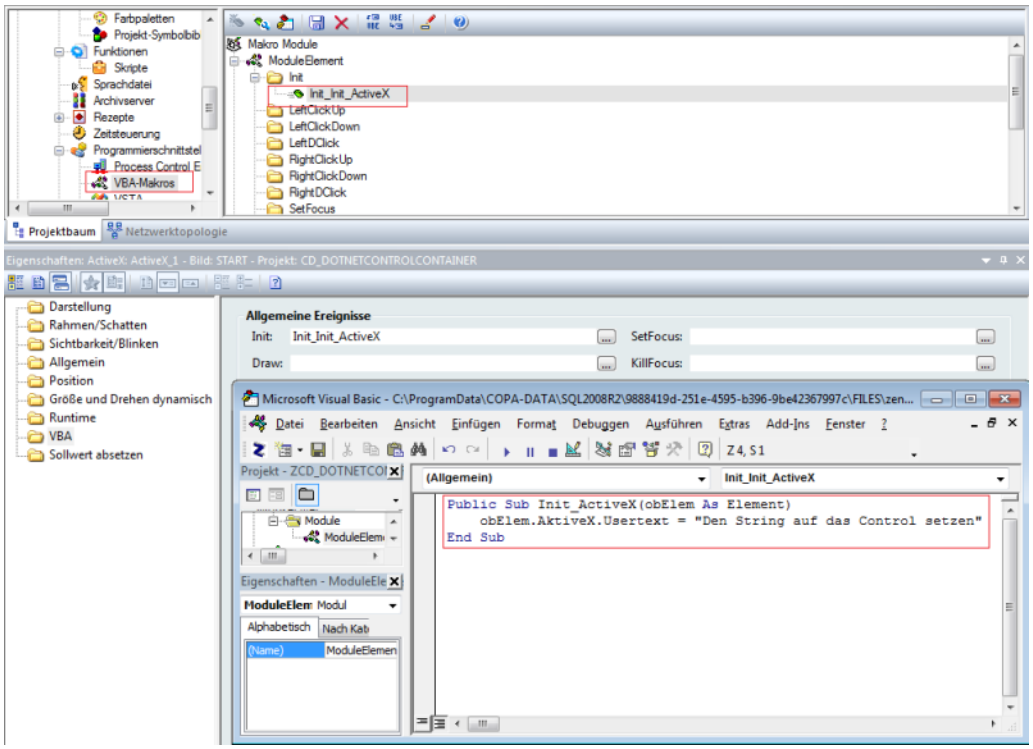
4.4.3 Im Editor über VBA mit ActiveX arbeiten

Um im zenon Editor auf die Eigenschaften des Controls zugreifen zu können:

1. Erzeugen Sie im zenon Editor im Knoten **Programmierschnittstellen/VBA-Makros** ein neues **Init** Makro mit der Bezeichnung **Init_ActiveX**.

   Mit diesem Makro kann man über **obElem.AktiveX** auf alle externen Properties zugreifen.

2. Weisen Sie dieses Makro über die Eigenschaften **VBA Makros/Init** des ActiveX Elements dem ActiveX Control zu.



BEISPIEL INIT MAKRO

```
Public Sub Init_ActiveX(obElem As Element)
    obElem.AktiveX.Usertext = "Den String auf das Control setzen"
End Sub
```

4.4.4 zenon Variablen mit dem .NET User Control verbinden

In zenon haben Sie die Möglichkeit, ein ActiveX Control um spezielle Funktionen zu erweiterten und darin auf die zenon API zuzugreifen.

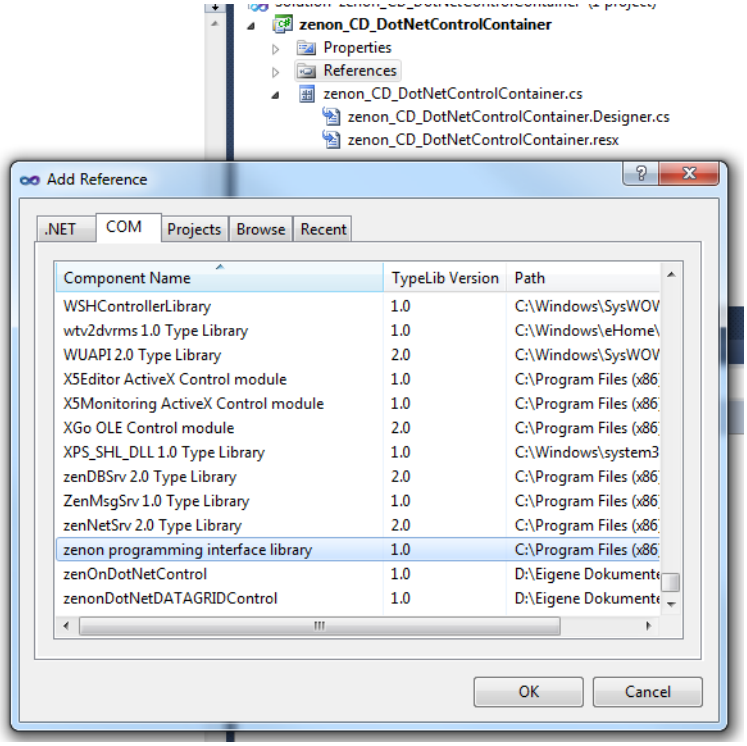
BENÖTIGTE METHODEN

- public bool zenOnInit (auf Seite 42) (Wird beim Control Initialisieren in der zenon Runtime aufgerufen.)
- public bool zenOnInitED (auf Seite 42) (Wird im Editor verwendet.)
- public bool zenOnExit() (auf Seite 43) (Wird beim Control-Zerstören in der zenon Runtime aufgerufen.)

- public bool zenOnExitED() (auf Seite 43) (Wird im Editor verwendet.)
- public short CanUseVariables() (auf Seite 43) (Unterstützt das Verknüpfungen von Variablen.)
- public short VariableTypes() (auf Seite 43) (Vom Control unterstützte Datentypen)
- public MaxVariables() (auf Seite 44) (Maximale Anzahl der Variablen, die mit dem Control verknüpft werden können.)

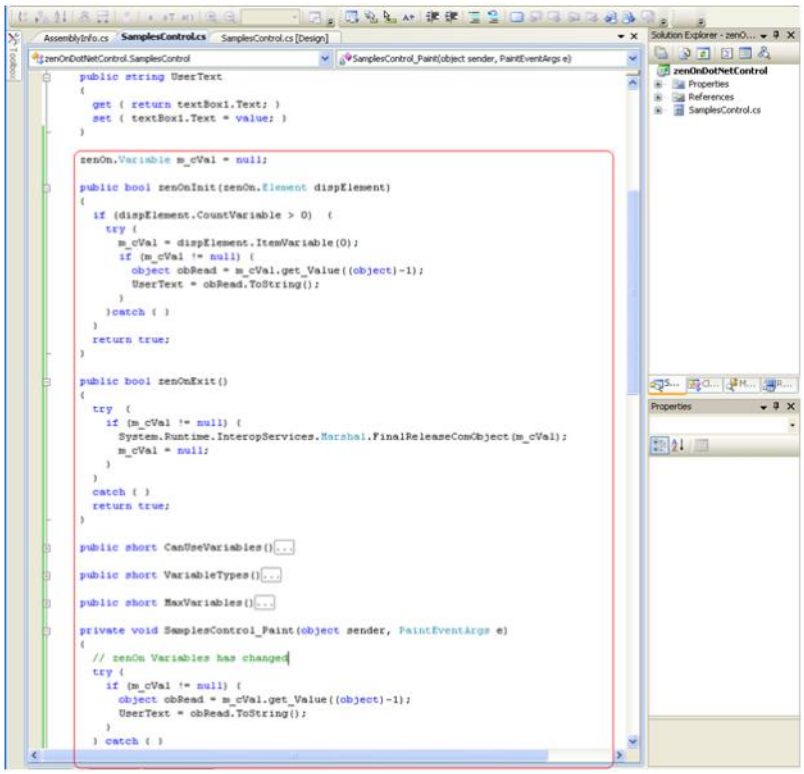
REFERENZ HINZUFÜGEN

1. Wählen Sie in Microsoft Visual Studio unter **Add References** die zenon Runtime-Objektbibliothek aus, damit man im Control auf die zenon API zugreifen kann.



2. Fügen Sie die erweiterten Funktionen in den Klassencode des Controls ein, um auf die gesamte zenon API zuzugreifen.

In unserem Beispiel wird das COM-Objekt einer zenon Variable in einem **Member** zwischengespeichert, um später im **Paint** Event des Controls darauf zugreifen zu können.



```
public string UserText
{
  get { return textBox1.Text; }
  set { textBox1.Text = value; }
}

zenOn.Variable m_cVal = null;

public bool zenOnInit(zenOn.Element dispElement)
{
  if (dispElement.CountVariable > 0)  {
    try {
      m_cVal = dispElement.ItemVariable(0);
      if (m_cVal != null) {
        object obRead = m_cVal.get_Value((object)-1);
        UserText = obRead.ToString();
      }
    }catch { }
  }
  return true;
}

public bool zenOnExit()
{
  try  {
    if (m_cVal != null) {
      System.Runtime.InteropServices.Marshal.FinalReleaseComObject(m_cVal);
      m_cVal = null;
    }
  }
  catch { }
  return true;
}

public short CanUseVariables()...

public short VariableTypes()...

public short MaxVariables()...

private void SamplesControl_Paint(object sender, PaintEventArgs e)
{
  // zenOn Variables has changed
  try {
    if (m_cVal != null) {
      object obRead = m_cVal.get_Value((object)-1);
      UserText = obRead.ToString();
    }
  } catch { }
```

public bool zenOnInit(zenOn.Element dispElement)

Mit Hilfe dieser Methode wird in der Runtime dem ActiveX Control ein Zeiger auf das Dispatch Interface des Dynamischen Elements übergeben. Über diesen Zeiger können dann die mit dem Dynamischen Element verknüpften zenon Variablen angesprochen werden.

Die Reihenfolge der übergebenen Variablen stellen Sie im Element-Eingabe Dialog mit den Schaltflächen **Nach unten** oder **Nach oben** ein. Der Element-Eingabe Dialog öffnet sich, wenn Sie:

- das ActiveX Element doppelklicken oder
- im Kontextmenü **Eigenschaften** wählen oder
- im Eigenschaftenfenster im Knoten **Darstellung** die Eigenschaft **ActiveX Einstellungen** wählen

public bool zenOnInitED(zenOn.Element dispElement)

Entspricht public bool zenOnInit (auf Seite 42)und wird beim Öffnen des ActiveX im Editor (Doppelklick auf ActiveX) ausgeführt.

public bool zenOnExit()

Diese Methode wird von der zenon Runtime beim Schließen des ActiveX Controls aufgerufen. Hier sollten alle Dispatch Pointer auf Variablen freigegeben werden.

public bool zenOnExitED()

Entspricht public bool zenOnExit() (auf Seite 43) und wird beim Schließen des ActiveX im Editor ausgeführt. Damit kann man auch im Editor auf Änderungen, etwas Wertänderungen, reagieren.

public short CanUseVariables()

Diese Methode gibt `1` zurück, wenn das Control zenon Variablen verwenden kann, und `0` wenn nicht.

- `1`: Für das Dynamische Element können(über die Schaltfläche **Variablen**) nur zenon Variablen mit den über die Methode **`VariableTypes`** angegebenen Typen in der von der Methode **`MaxVariables`** angegebenen Anzahl angegeben werden.
- `0`: Gibt **`CanUseVariables`** `0` zurück oder besitzt das Control diese Methode nicht, können Variablen mit allen Typen ohne Beschränkung der Anzahl angegeben werden. Diese können in der Runtime aber nur über VBA verwendet werden.

public short VariableTypes()

Der von dieser Methode zurückgegebene Wert wird als Maske für die verwendbaren Variablentypen in der Variablenliste verwendet. Der Wert ist eine **UND**-Verknüpfung aus folgenden Werten (definiert in **zenon32/dy_type.h**):

Parameter	Wert	Beschreibung
WORD	0x0001	entspricht Stelle 0
BYTE	0x0002	entspricht Stelle 1
BIT	0x0004	entspricht Stelle 2
DWORD	0x0008	entspricht Stelle 3
FLOAT	0x0010	entspricht Stelle 4
DFLOAT	0x0020	entspricht Stelle 5
STRING	0x0040	entspricht Stelle 6
IN_OUTPUT	0x8000	entspricht Stelle 15

public MaxVariables()

Hier wird die Anzahl der Variablen festgelegt, die aus der Variablenliste ausgewählt werden können:

`1`: In der Variablenliste wird die Mehrfachauswahl deaktiviert. Es erfolgt eine Warnung, wenn trotzdem mehrere Variable ausgewählt werden.

5. .NET User Controls

Mit .NET Controls kann die Funktionalität der zenon Runtime und des Editors selbstständig erweitert werden.

In diesem Handbuch lesen Sie:

- Unterschiede zwischen Control Container und ActiveX (auf Seite 44)
- Beispiel .NET Control Container (auf Seite 45)
- Beispiel .NET Control als ActiveX (C#) (auf Seite 30)

Informationen zur Verwendung von .NET Controls in ActiveX finden Sie im auch Handbuch Bilder im Kapitel .NET Controls.

> **Achtung**
>
> *Bei der Verwendung von zenon COM-Objekten in selbst erstellten User Controls oder externen Anwendungen müssen diese über die Methode* ***Marshal.ReleaseComObject*** *freigegeben werden. Die Freigabe über die Methode* ***Marshal.FinalReleaseComObject*** *darf nicht verwendet werden, da diese zu einem Ausfall von zenon Add-Ins führt.*

5.1 Unterschiede Nutzung .NET Control in Control Container oder ActiveX

Ein .NET User Control kann:

- über CD_DotNetControlContainer Control direkt in das zenon ActiveX-Element eingebunden werden
- als ActiveX Control verwendet und direkt in das zenon ActiveX-Element eingebunden werden

Unterschiede zwischen Container Control und ActiveX Control sind vor allem:

CD_DotNetControlContainer Control	ActiveX Control
Muss nicht am Rechner registriert werden.	Muss als ActiveX am Rechner registriert werden (regsvr32).
Bei Änderungen am Control muss nur die DLL ausgetauscht werden.	Bei Änderungen am Control muss die TLB neu registriert werden.
Zugriff über VBA und VSTA nur über CD_DotNetControlContainer Methoden möglich.	Zugriff über VBA und VSTA sehr einfach.

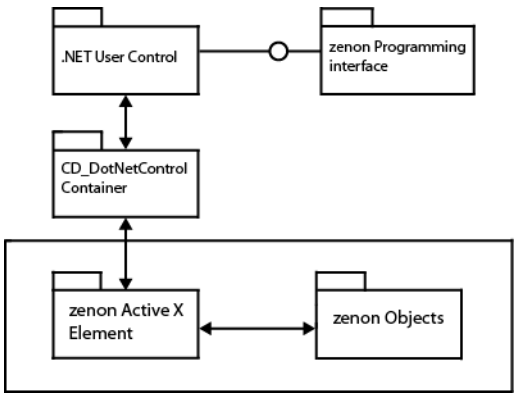
5.2 Beispiel .NET Control Container

In diesem Tutorial wird beschrieben, wie man in Visual Studio 2010 ein einfaches .NET User Control (Programmiersprache **C#**) erstellt und dieses anschließend mit Hilfe des zenon **CD_DotNetControlContainer** Controls als ActiveX in ein zenon ActiveX Element einbindet.

5.2.1 Allgemeines

Der CD_DotNetControlContainer dient als Wrapper zwischen einem User Control und dem zenon AktiveX Element. Alle im folgenden Beispiel verwendeten Methoden sowie alle Public Methoden und Properties werden über den CD_DotNetControlContainer vom User Control zum ActiveX weitergegeben und können von zenon verwendet werden, auch in VBA und VSTA.

Wird im User Control auf das zenon Programming Interface referenziert, kann direkt auf zenon Objekte zugegriffen werden.



Im folgenden Beispiel werden wir:

- ein .NET User Control erstellen (auf Seite 47)

- CD_DotNetControlContainer und .NET User Control einfügen (auf Seite 56)
- den Zugriff auf das User Control über VSTA (VBA) ermöglichen (auf Seite 61)

PFAD FÜR DLL IN EDITOR UND RUNTIME

Der Pfad zur **.Net DLL**, der im Editor gewählt wird, wird auch zur Runtime verwendet. Er wird absolut gesetzt und darf nicht verändert werden.

Stellen Sie sicher, dass auf allen Rechnern im zenon Netzwerk für Editor und Runtime der gleiche Pfad verwendet wird.
Tipp: Wählen Sie einen absoluten Pfad, zum Beispiel: `C:\Controls`. Tragen Sie den Pfad fix im **Remote-Transport** und im **.NET Control Container** ein. Gleichen Sie diesen Pfad über den **Remote-Transport** mit allen Rechnern ab.

public bool zenOnInit(zenOn.Element dispElement)

Mit Hilfe dieser Methode wird in der Runtime dem ActiveX Control ein Zeiger auf das Dispatch Interface des Dynamischen Elements übergeben. Über diesen Zeiger können dann die mit dem Dynamischen Element verknüpften zenon Variablen angesprochen werden.

Die Reihenfolge der übergebenen Variablen stellen Sie im Element-Eingabe Dialog mit den Schaltflächen **Nach unten** oder **Nach oben** ein. Der Element-Eingabe Dialog öffnet sich, wenn Sie:

- das ActiveX Element doppelklicken oder
- im Kontextmenü **Eigenschaften** wählen oder
- im Eigenschaftenfenster im Knoten **Darstellung** die Eigenschaft **ActiveX Einstellungen** wählen

public bool zenOnExit()

Diese Methode wird von der zenon Runtime beim Schließen des ActiveX Controls aufgerufen. Hier sollten alle Dispatch Pointer auf Variablen freigegeben werden.

public short CanUseVariables()

Diese Methode gibt `1` zurück, wenn das Control zenon Variablen verwenden kann, und `0` wenn nicht.

- `1`: Für das Dynamische Element können(über die Schaltfläche **Variablen**) nur zenon Variablen mit den über die Methode **`VariableTypes`** angegebenen Typen in der von der Methode **`MaxVariables`** angegebenen Anzahl angegeben werden.

- 0: Gibt `CanUseVariables 0` zurück oder besitzt das Control diese Methode nicht, können Variablen mit allen Typen ohne Beschränkung der Anzahl angegeben werden. Diese können in der Runtime aber nur über VBA verwendet werden.

public short VariableTypes()

Der von dieser Methode zurückgegebene Wert wird als Maske für die verwendbaren Variablentypen in der Variablenliste verwendet. Der Wert ist eine **UND**-Verknüpfung aus folgenden Werten (definiert in **zenon32/dy_type.h**):

Parameter	Wert	Beschreibung
WORD	0x0001	entspricht Stelle 0
BYTE	0x0002	entspricht Stelle 1
BIT	0x0004	entspricht Stelle 2
DWORD	0x0008	entspricht Stelle 3
FLOAT	0x0010	entspricht Stelle 4
DFLOAT	0x0020	entspricht Stelle 5
STRING	0x0040	entspricht Stelle 6
IN_OUTPUT	0x8000	entspricht Stelle 15

public MaxVariables()

Hier wird die Anzahl der Variablen festgelegt, die aus der Variablenliste ausgewählt werden können:

1: In der Variablenliste wird die Mehrfachauswahl deaktiviert. Es erfolgt eine Warnung, wenn trotzdem mehrere Variable ausgewählt werden.

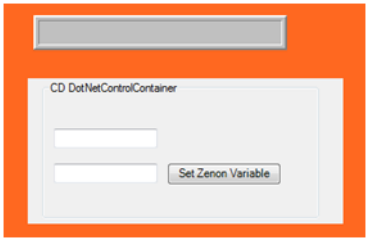
5.2.2 .Net User Control erstellen

Bei dem User Control handelt es sich um ein einfaches Control, das es ermöglicht, über ein Eingabefeld (Textbox) einen neuen Wert zu setzen. Nach Drücken des Buttons wird der Wert auf die gewünschte zenon Variable geschrieben.

Eine weitere Funktion soll automatisch eine Wertänderung der Variable in zenon erkennen und den neuen Wert automatisch im Control anzeigen.

Info

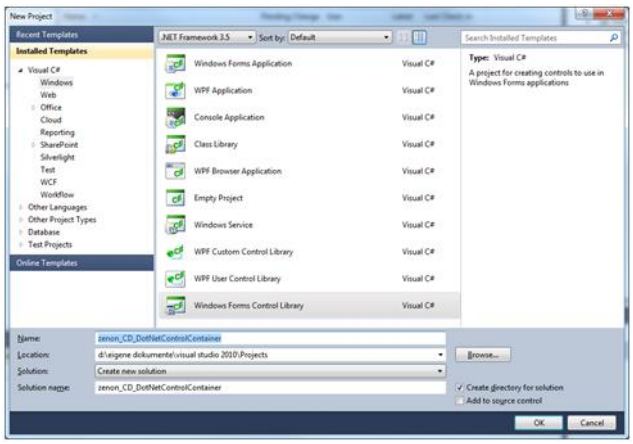
Die Screenshots für dieses Thema sind nur in englischer Sprache verfügbar.



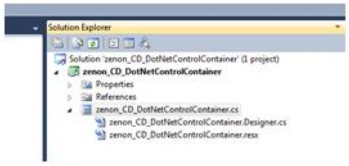
ARBEITSSCHRITTE

1. Zunächst erstellen Sie ein neues Projekt in VS und verwenden dazu den Projekttypen „**Windows Forms Control Library**"

   Wichtig: Framework auf 3.5 einstellen!



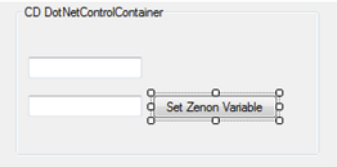
2. Danach benennen Sie die CS-Datei „**UserControl**" in „**zenon_CD_DotNetControlContainer.cs**" um. Die Dateien **Designer.cs** und **.resx** werden automatisch mit unbenannt.



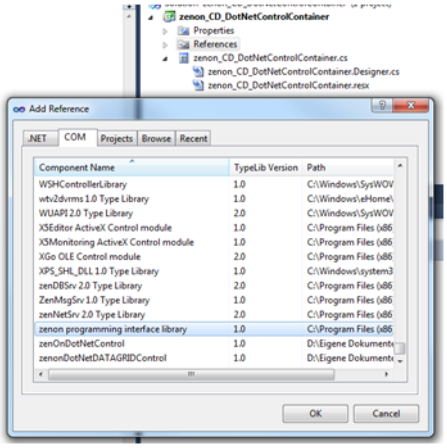
3. Im nächsten Schritt erstellen Sie das User Control. Dazu benötigen Sie zwei Textboxen für die Ein-und Ausgabe sowie einen Button zum Absetzen eines neuen Wertes auf die zenon Variable. Benennen Sie:

   - die erste Textbox „**txtGetZenonVariable**"
   - die zweite Textbox „**txtSetZenonVariable**"

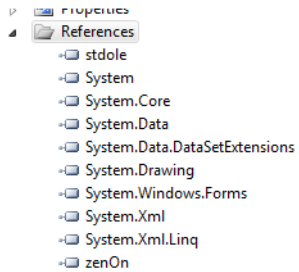
- den Button „**btnSetZenonVariable**"



4. Um auf zenon Objekte zugreifen zu können, benötigen Sie eine Referenz auf das zenon Programming Interface. Dazu:
   - klicken Sie auf den Knoten „**References**" im Solution Explorer
   - öffnen das Kontextmenü
   - wählen **Add References...**
   - wechseln zur Registerkarte **COM**
   - wählen die **zenon programming interface library** aus



Danach sollte die „**zenOn**" Referenz in der Liste der Referenzen sichtbar sein.



5. Im nächsten Schritt legen Sie im Programmcode der **zenon_CD_DotNetControlContainer.cs** eine globale Variable vom Typ `zenon.Variable` an:

```
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 zenOn;

namespace zenon_CD_DotNetControlContainer
{
    public partial class zenon_CD_DotNetControlContainer : UserControl
    {
        //This will be needed to get the zenon Variable Container
        zenOn.Variable m_cVal = null;

        public zenon_CD_DotNetControlContainer()
        {
            InitializeComponent();
        }


    }
}

```

6. Diese Variable wird über die public Methode **zenOnInit** initialisiert:

```
        /// <summary>
        /// This public Method will be called by the initialization of the control during
        /// the zenon Runtime.
        /// </summary>
        /// <param name="dispElement"></param>
        /// <returns></returns>
        public bool zenOnInit(zenOn.Element dispElement)
        {
            //Check if zenon Variables are added to the
            //Control
            if (dispElement.CountVariable > 0)
            {
                try
                {
                    //Take the first zenon Variable and added
                    //to the global Variable
                    m_cVal = dispElement.ItemVariable(0);
                    //Set Value to the TextBox
                    txtGetZenonVariable.Text = m_cVal.get_Value(0).ToString();
                }
                catch { }
            }
            return true;
        }
```

und über die public Methode **zenOnExit** wieder freigegeben:

```
        /// <summary>
        /// This public Method will be calles by the release of the control during
        /// the zenon Runtime.
        /// </summary>
        /// <returns></returns>
        public bool zenOnExit()
        {
            try
            {
                if (m_cVal != null)
                {
                    //Release the zenon Variable (Com-Object)
                    System.Runtime.InteropServices.Marshal.FinalReleaseComObject(m_cVal);
                    m_cVal = null;
                }
            }
            catch { }
            return true;
        }
```

In den folgenden Methoden wird festgelegt ob zenon Variable und Datentypen verwendet werden und wie viele Variable übergeben werden dürfen:

```
        /// <summary>
        /// This public Method is needed to link zenon Variables
        /// to the control.
        /// </summary>
        /// <returns></returns>
        public short CanUseVariables()
        {
            return 1; // Only tis Variable is supported
        }

        /// <summary>
        /// This public Method returns the Type of
        /// supported zenon Variables
        /// </summary>
        /// <returns></returns>
        public short VariableTypes()
        {
            return short.MaxValue; // all Data Types supported
        }

        /// <summary>
        /// This public Method returns the number of
        /// supported zenon Variables
        /// </summary>
        /// <returns></returns>
        public short MaxVariables()
        {
            return 1; // Only 1 Variable should linked to the Control
        }
```

7. Im nächsten Schritt legen Sie im **Click-Event** des Buttons **btnSetZenonVariable** fest, dass beim Klicken des Buttons der Wert in der Textbox **txtSetZenonVariable** auf den Wert der zenon Variable geschrieben und danach der Inhalt der Textbox gelöscht wird.

```
98          /// <summary>
99          /// This will be triggert by clicking the Button. The new Value will
100         /// be set to the zenon Variable
101         /// </summary>
102         /// <param name="sender"></param>
103         /// <param name="e"></param>
104         private void btnSetZenonVariable_Click(object sender, EventArgs e)
105         {
106             //Set Value from TextBox to the zenon Variable
107             m_cVal.set_Value(0,txtSetZenonVariable.Text.ToString());
108             this.txtSetZenonVariable.Text = string.Empty;
109         }
```

8. Um auf eine Wertänderung der Variable reagieren können, benötigen Sie das **Paint-Event** des Controls. Das **Paint-Event** wird auch getriggert, wenn sich der Wert der initialisierten zenon Variable ändert und kann somit für die Aktualisierung von Werten verwendet werden. Da Variablen die im zenon ActiveX-Element referenziert sind automatisch `advised` sind, kann generell auf den **zenon.OnlineVariable** Container im Control verzichtet werden.

```
111         /// <summary>
112         /// This will be triggert by painting the User Control or the Value of the Variable changed.
113         /// After the value of the Variable changed the Control will be new painted and the new Value
114         /// will be set to the Textbox.
115         /// </summary>
116         /// <param name="sender"></param>
117         /// <param name="e"></param>
118         private void zenon_CD_DotNetControlContainer_Paint(object sender, PaintEventArgs e)
119         {
120             if (m_cVal != null)
121             {
122                 this.txtGetZenonVariable.Text = m_cVal.get_Value(0).ToString();
123                 return;
124             }
125             else
126             {
127                 this.txtGetZenonVariable.Text = "Variable Value";
128                 return;
129             }
130
131         }
```

DER CODE IM ÜBERBLICK

Hier nochmals der gesamte Code im Überblick:

```
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 zenOn;

namespace zenon_CD_DotNetControlContainer

{
```

```
public partial class zenon_CD_DotNetControlContainer : UserControl
{
    //This will be needed to get the zenon Variable Container
    zenOn.Variable m_cVal = null;
    public zenon_CD_DotNetControlContainer()
    {
        InitializeComponent();
    }
/// <summary>
/// This public Method will be called by the initialization of the control during
/// the zenon Runtime.
/// </summary>
/// <param name="dispElement"></param>
/// <returns></returns>
    public bool zenOnInit(zenOn.Element dispElement)
    {
      //Check if zenon Variables are added to the
 //Control
     if (dispElement.CountVariable > 0)
        {
            try
            {
         //Take the first zenon Variable and added
         //to the global Variable
                m_cVal = dispElement.ItemVariable(0);
            }
            catch { }
        }
        return true;
    }
```

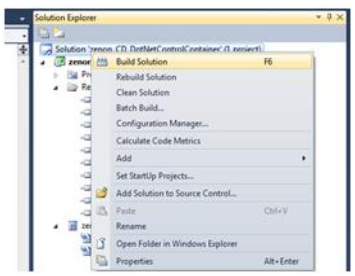
```
// <summary>
/// This public Method will be called by the release of the control during
/// the zenon Runtime.
/// </summary>
/// <returns></returns>
    public bool zenOnExit()
    {
        try
        {
            if (m_cVal != null)
            {
                //Release the zenon Variable (Com-Object)
                System.Runtime.InteropServices.Marshal.ReleaseComObject(m_cVal);
                m_cVal = null;
            }
        }
        catch {}
        return true;
    }
/// <summary>
/// This public Method is needed to link zenon Variables
/// to the control.
/// </summary>
/// <returns></returns>
    public short CanUseVariables()
    {
        return 1; // Only this Variable is supported
    }
    /// <summary>
    /// This public Method returns the Type of
```

```
        /// supported zenon Variables
        /// </summary>
        /// <returns></returns>
        public short VariableTypes()
        {
            return short.MaxValue; // all Data Types supported
        }
        /// <summary>
        /// This public Method returns the number of
        /// supported zenon Variables
        /// </summary>
        /// <returns></returns>
        public short MaxVariables()
        {
            return 1; // Only 1 Variable should linked to the Control
        }
        /// <summary>
        /// This will be triggert by clicking the Button. The new Value will
        /// be set to the zenon Variable
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btnSetZenonVariable_Click(object sender, EventArgs e)
        {
            //Set Value from TextBox to the zenon Variable
            m_cVal.set_Value(0,txtSetZenonVariable.Text.ToString());
            this.txtSetZenonVariable.Text = string.Empty;
        }
        /// <summary>
        /// This will be triggert by painting the User Control or the Value of the Variable
changed.
```

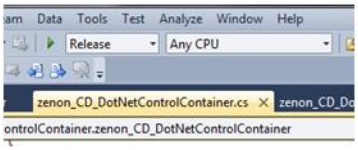
```
        /// After the value of the Variable changed the Control will be new painted and
the new Value
        /// will be set to the Textbox.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void zenon_CD_DotNetControlContainer_Paint(object sender, PaintEventArgs e)
        {
            if (m_cVal != null)
            {
                this.txtGetZenonVariable.Text = m_cVal.get_Value(0).ToString();
                return;
            }
            else
            {
                this.txtGetZenonVariable.Text = "Variable Value";
                return;
            }
        }
    }
}
```

RELEASE ERSTELLEN

Zum Schluss erstellen Sie ein Release, um die fertige DLL in zenon und/oder in den **CD_DotNetControlContainer** einbinden zu können.



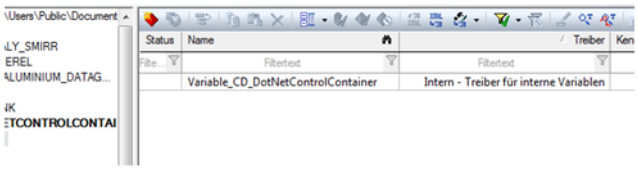
Dafür muss unbedingt in den Settings von **Debug** auf **Release** umgestellt werden.



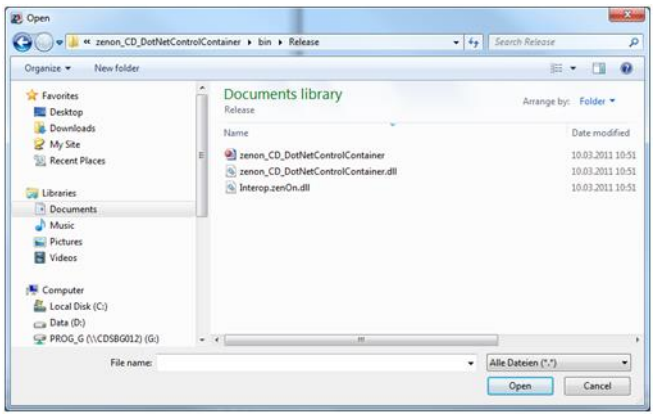
5.2.3 CD_DotNetControlContainer und .NET User Control einfügen

Um das zenon Projekt vorzubereiten und den **CD_DotNetControlContainer** und das **.NET User Control** einzufügen, führen Sie folgende Schritte aus:

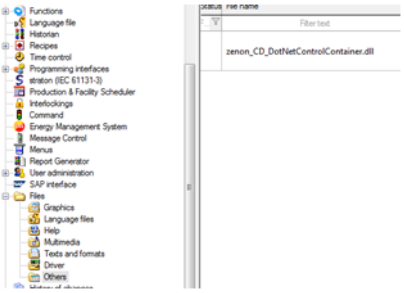
1. Legen Sie eine interne Variable vom Typ `String` an und setzen Sie die Stringlänge auf `30`.



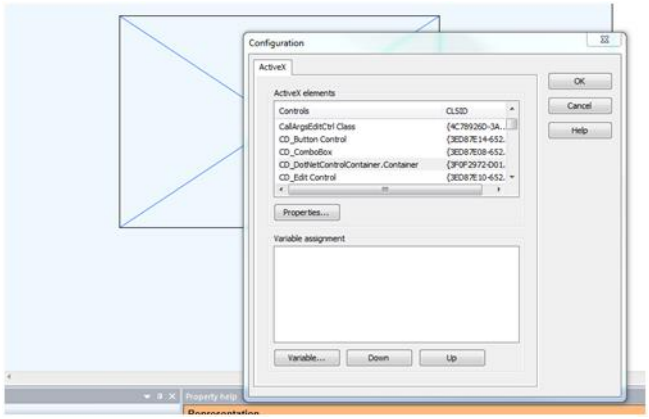
2. Fügen Sie im zenon Projektknoten `Projekt/Dateien/Sonstige` die DLL des erstellten .NET User Controls ein. Hierbei wird die Datei in den Ordner **Additional** auf Dateisystemebene kopiert.



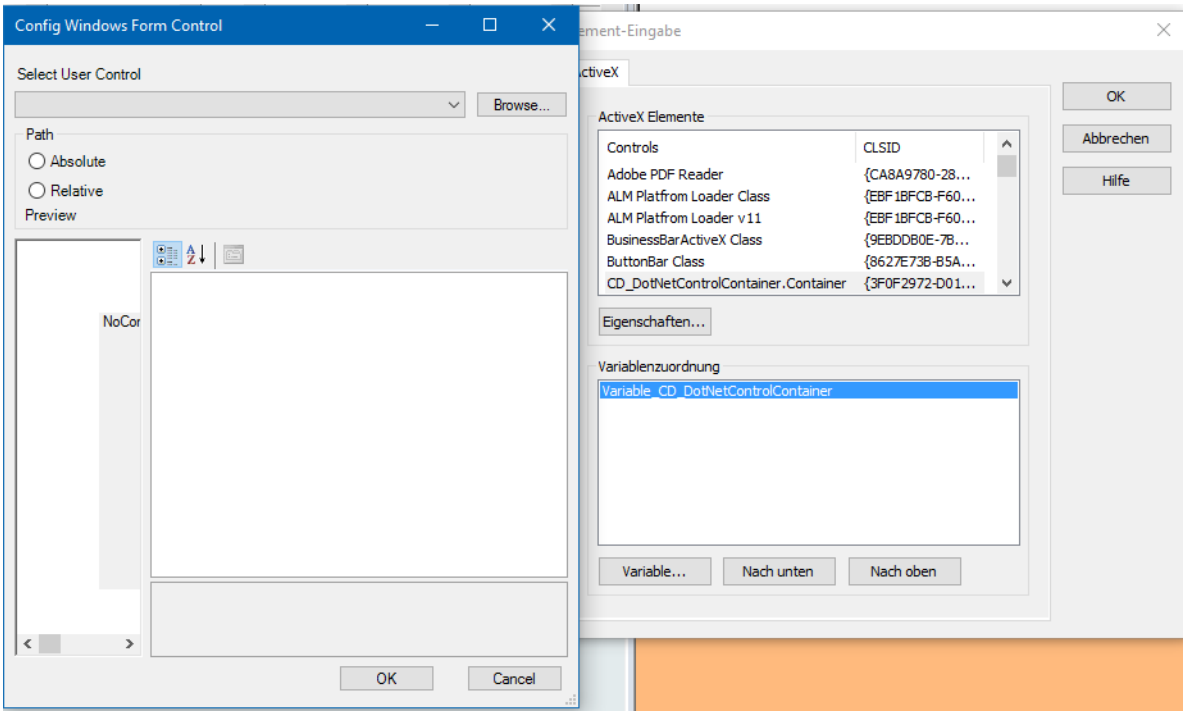
3. Die DLL befindet sich im Output-Ordner des Visual Studio Projekts unter `...\bin\Release\zenon_CD_DotNetControlContainer.dll`.



4. Im Projekt wählen Sie das ActiveX-Element und ziehen es in ein zenon Bild.
   - Der Dialog **Konfiguration** wird geöffnet.
   - Wählen Sie das **CD_DotNetControlContainer.Container** Control aus.

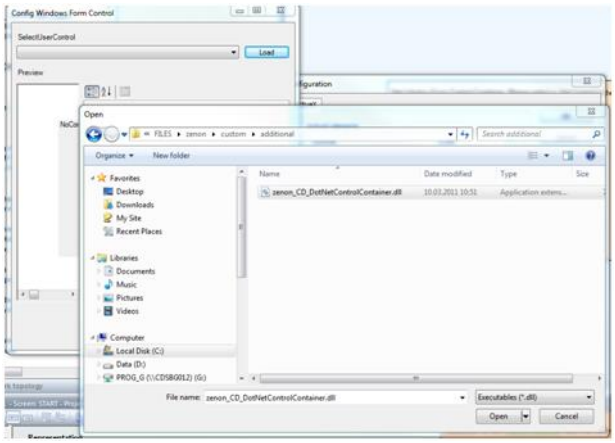


5. Um das .NET User Control in das **CD_DotNetControlContainer** Control einzubetten:
   - Klicken Sie auf die Schaltfläche **Eigenschaften.**
   - Ein neuer Dialog wird geöffnet.



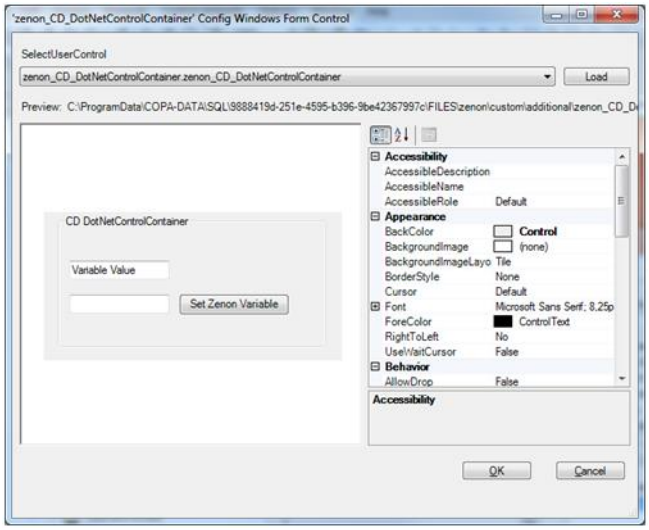
   - Mit Klick auf die Schaltfläche **Load** wählen Sie den Pfad des Projektordners, zum Beispiel: `C:\ProgramData\COPA-DATA\SQL\9888419d-251e-4595-b396-9be42367997c\FILES\zenon\custom\additional\zenon_CD_DotNetControlContainer.dll`

Hinweis: Controls sollten immer im Ordner **Additional** des Projekts abgelegt werden. Somit werden sie bei Sicherungen und Übertragungen berücksichtigt. Der Pfad ist dabei relativ zum Speicherort zu definieren. Alternativ können Sie bei **Path** die Option `Absolute` verwenden, dabei muss aber sichergestellt sein dass am Zielsystem die selbe Verzeichnisstruktur existiert.

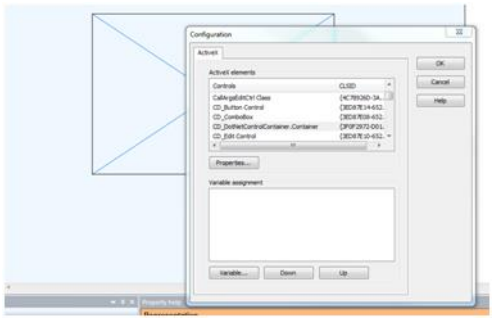


Nun sollte das .NET User Control angezeigt werden.

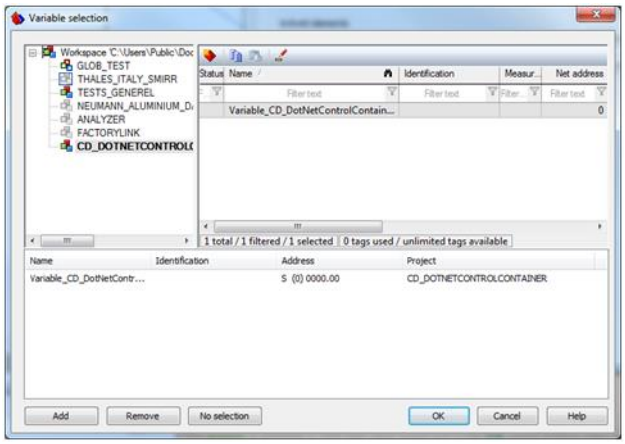
Bestätigen Sie den Dialog mit Klick auf **OK**.



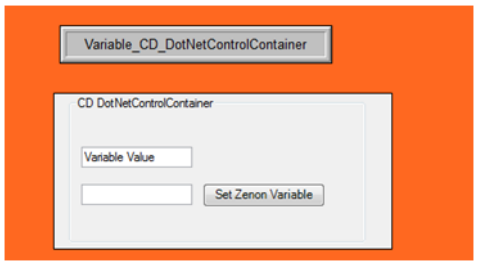
6. Im letzten Schritt verknüpfen Sie über die Schaltfläche **Variablen** eine Variable mit dem Control.



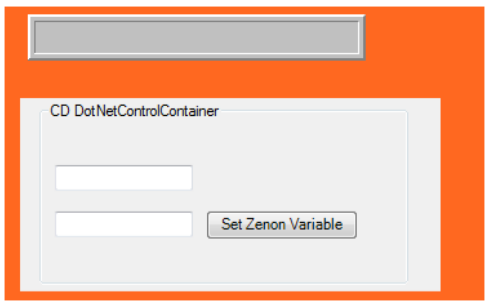
Die erste ausgewählte Variable wird automatisch über die **public** Methode `zenonInit` mit der global definierten Variable (.NET UserControl) verknüpft. Die Verlinkung mit dem Control erfolgt aber erst nach dem Start der Runtime.



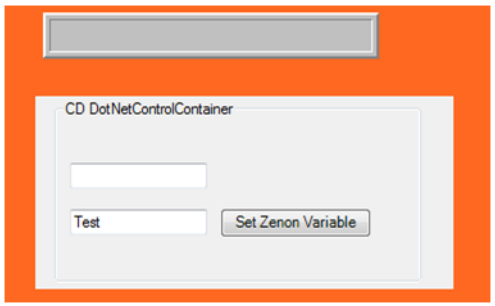
Dann verlinken Sie noch die interne Variable mit einem Textelement.



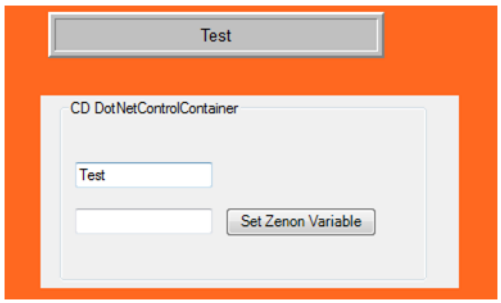
7. Nach dem Runtime Start ist das Control zunächst leer.



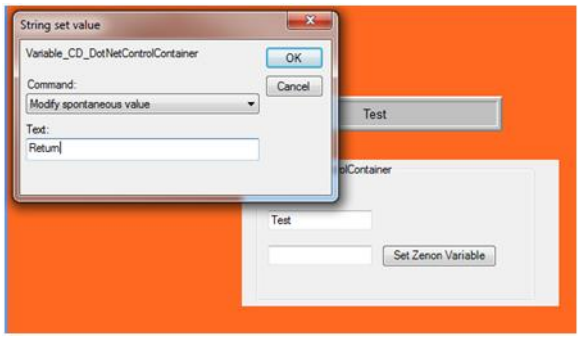
Wird ein Wert in die zweite Textbox geschrieben und danach mit dem Button **Set zenon Variable** bestätigt, dann wird der Wert auf die zenon Variable geschrieben. (Das **btnSetZenonVariable_Click** Event wird ausgeführt.)



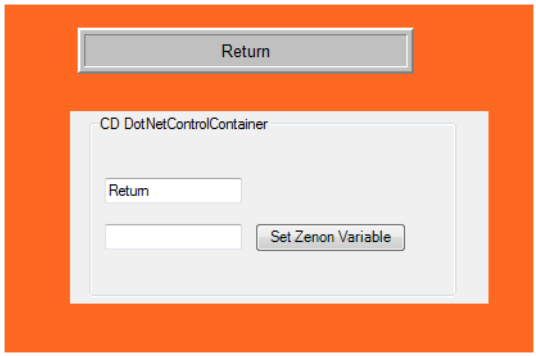
Dieser wird auch im zenon Textelement angezeigt.



Wird der Wert direkt im zenon Text-Element geändert,



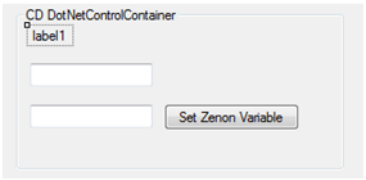
dann wird der Wert direkt über das **Paint** Event des .NET Controls in die erste Textbox geschrieben.



5.2.4 Zugriff auf das User Control über VSTA oder VBA

In diesem Beispiel wird der Zugriff über VSTA gezeigt. Die Vorgehensweise ist bei VBA die gleiche.

1. Erweitern Sie das Control mit einen Label (**label**) und benennen Sie dieses l**blzenonInfo**. In diesem Label soll der Wert einer anderen zenon Variablen angezeigt werden. Der neue Wert soll über ein VSTA Makro gesetzt werden.

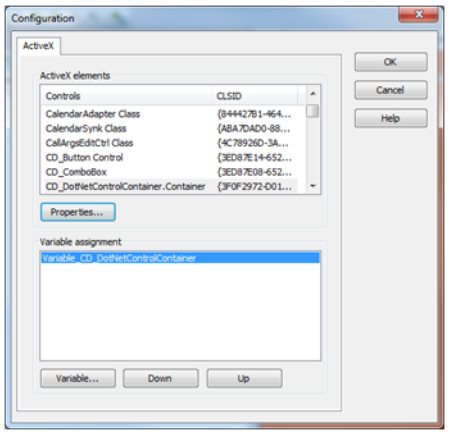


2. Erweitern Sie den Code um ein Property (**Information**) und fügen **get** und **set** Eigenschaften zu dem Property hinzu. Diese erlauben es, den Text des Labels zu lesen und zu schreiben.

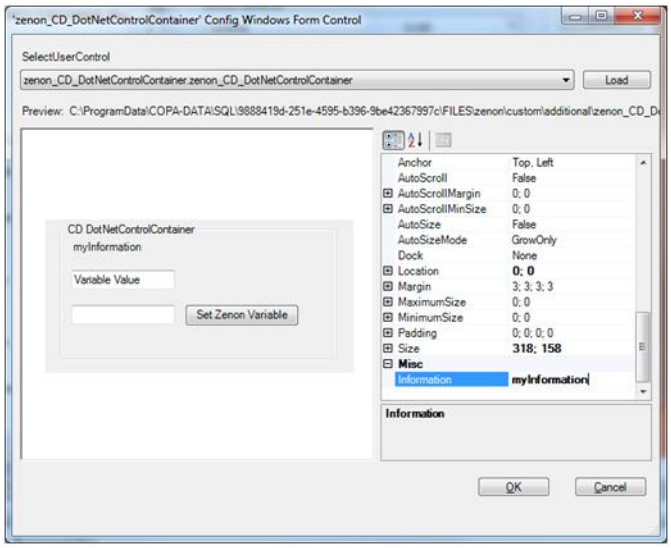
```
13  public partial class zenon_CD_DotNetControlContainer : UserControl
14  {
15      //This will be needed to get the zenon Variable Container
16      zenOn.Variable m_cVal = null;
17
18      public zenon_CD_DotNetControlContainer()
19      {
20          InitializeComponent();
21      }
22
23      public string Information
24      {
25          set{this.lblZenonInfo.Text = value;}
26          get { return this.lblZenonInfo.Text; }
27      }
28
```

3. Erstellen Sie ein neues Release für unser User Control und kopieren Sie dieses wieder in den Ordner `additional` des zenon Projekts.
   Nicht vergessen: zenon Editor vorher schließen!
   Löschen Sie die alte DLL und starten Sie den zenon Editor neu. Sollte sich die DLL immer noch im Ordner befinden, einfach ein zweites Mal löschen. Jetzt kann die geänderte DLL neu importiert werden. Das **CD_DotNetContainerControl** sowie das ActiveX werden automatisch aktualisiert.

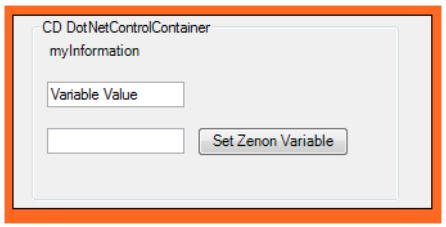
4. Klicken Sie im zenon Editor auf das ActiveX und öffnen Sie das Eigenschaftenfenster.



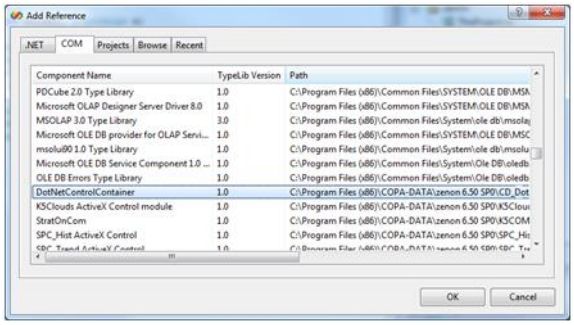
Sie können nun die neue Eigenschaft **Information** im Auswahlfenster des Controls sehen und auch einen Wert setzen.



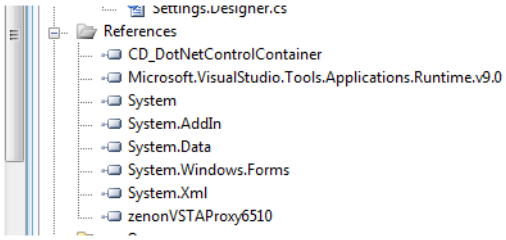
Dieser Wert wird auch im Control gesetzt („myInformation")



5. Um mit dem **CD_DotNetControlContainer** in VSTA oder VBA arbeiten zu können, benötigen Sie zunächst die Referenz auf das Control. Nach dem VSTA für das Projekt geöffnet wurde (**ProjectAddin**) muss die Referenz vom **CD_DotNetControlContainer** hinzugefügt werden.



Zusätzlich muss auch das Assembly **System.Windows.Forms** hinzugefügt werden.



6. Mit folgendem Code kann der Wert für die Control Eigenschaft **Information** neu gesetzt werden:

```
public void Macro_Test()
{
    try
    {
        zenOn.IElements zELements = this.DynPictures().Item("START").Elements();
        zenOn.IElement zElement = zELements.Item("ActiveX_1");

        // Create a Variable of Type CD_DotNetControlContainer.Container and get the zenon AktiveX Element
        // with a cast
        CD_DotNetControlContainer.Container zAktiveX = (CD_DotNetControlContainer.Container)zElement.AktiveX();

        //With using SetExternalUserControlProperty and the name of the Property "Information" we can set
        // a new Value "New Information" to the Property
        if (zAktiveX.GetExternalUserControlProperty("Information").Equals("myInformation"))
        {
            zAktiveX.SetExternalUserControlProperty("Information", "New Information");
        }
        else
        {
            zAktiveX.SetExternalUserControlProperty("Information", "myInformation");
        }

    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.Print("ERROR : " + ex.Message + " " + ex.Source);
    }
}
```

7. Zum Schluss:
   - erstellen Sie eine zenon Funktion **VSTA Makro ausführen**
   - verknüpfen Sie die Funktion mit einem Button

   Zur Runtime wird das Label beim Klick auf den Button von **myInformation** auf **New Information** geändert.

   Und beim nächsten Drücken des Buttons wieder zurück geändert.

5.3 Beispiel .NET Control als ActiveX (C#)

Das folgende Beispiel beschreibt ein .NET Control, das als ActiveX Control in zenon ausgeführt wird.

Die Erstellung und Integration passiert in vier Schritten:

1. Windows Form Control erzeugen (auf Seite 31)
2. .NET User Control in Dual Control umwandeln (auf Seite 34)
3. Im Editor über VBA mit ActiveX arbeiten (auf Seite 39)
4. zenon Variablen mit dem .NET User Control verbinden (auf Seite 40)

Achtung

Bei der Verwendung von zenon COM-Objekten in selbst erstellten User Controls oder externen Anwendungen müssen diese über die Methode ***Marshal.ReleaseComObject*** *freigegeben werden. Die Freigabe über die Methode* ***Marshal.FinalReleaseComObject*** *darf nicht verwendet werden, da diese zu einem Ausfall von zenon Add-Ins führt.*

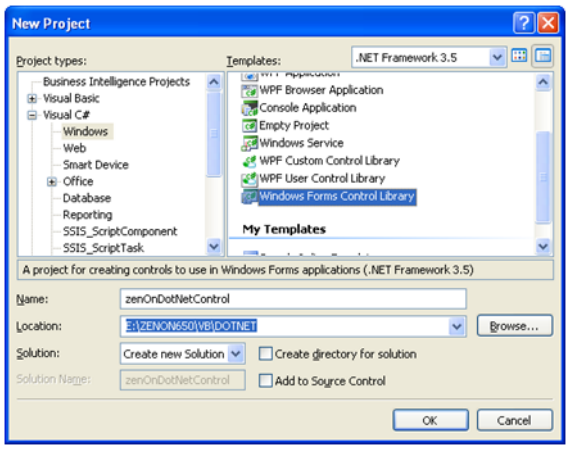
Info

Die Screenshots für dieses Thema sind nur in englischer Sprache verfügbar.

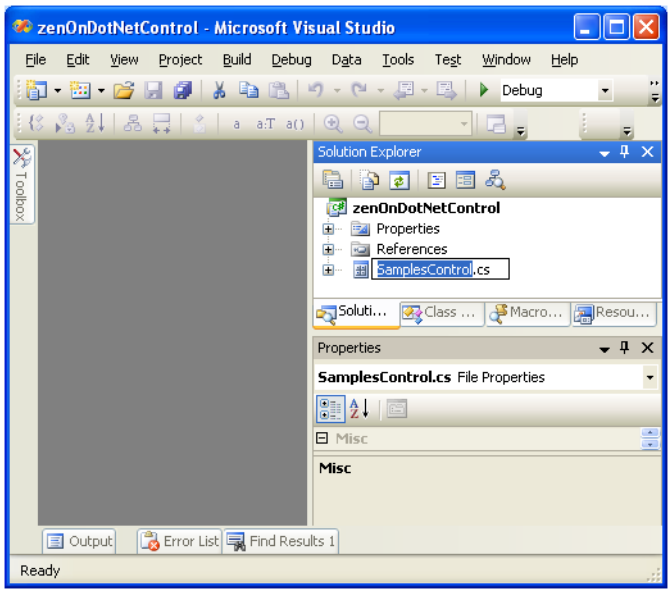
5.3.1 Windows Form Control erzeugen

Um ein Windows Form Control zu erzeugen:

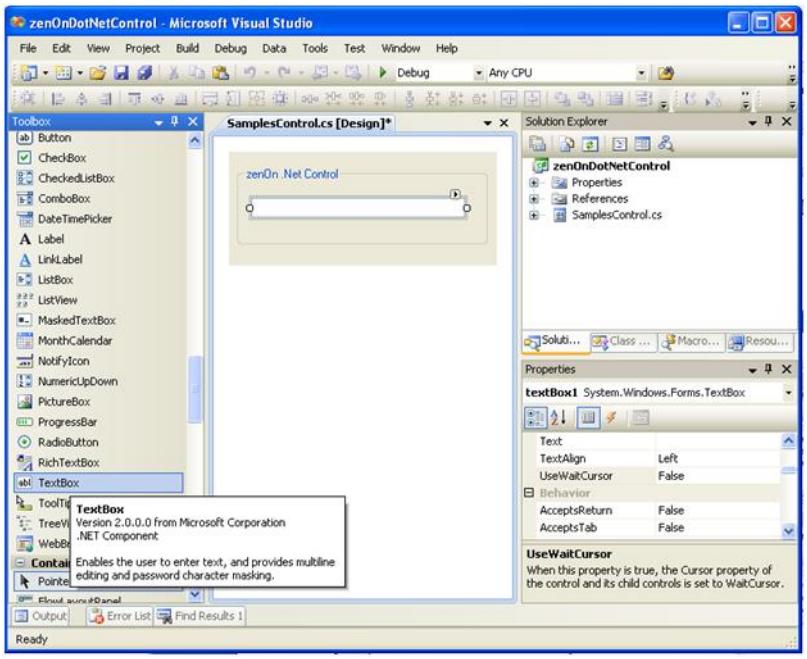
1. Starten Sie Visual Studio 2008 und erzeugen Sie ein neues Windows **Form Control Library** Projekt:



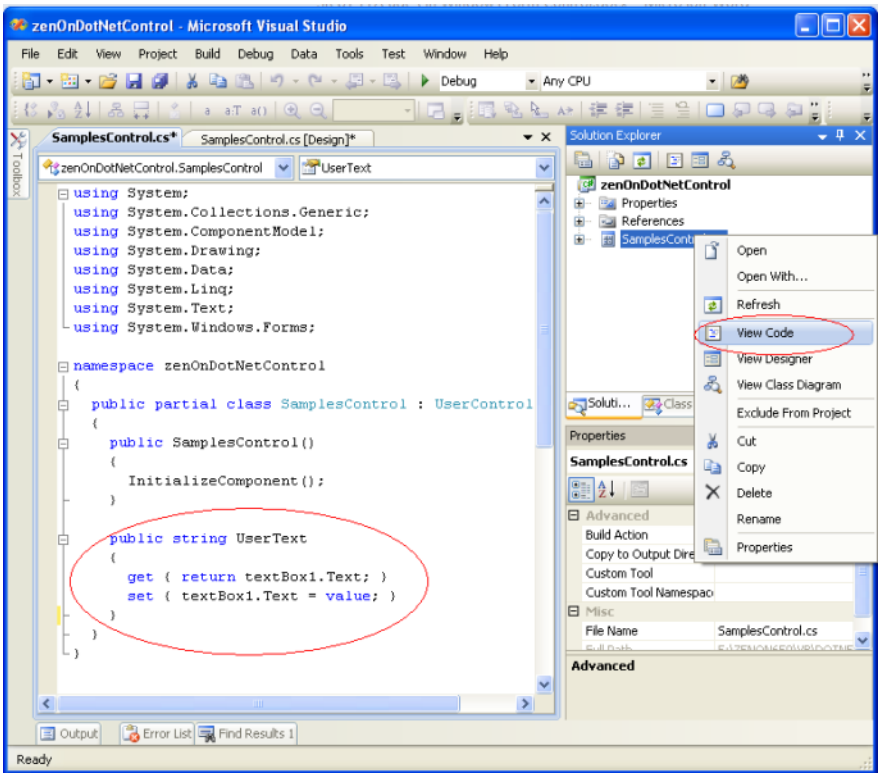
2. Benennen Sie das Default Control auf den gewünschten Controlnamen um.
In unserem Beispiel: **SampesControl.cs**.



3. Öffnen Sie den Control Designer und fügen Sie die gewünschten Controls ein, in unserem Fall eine Textbox:



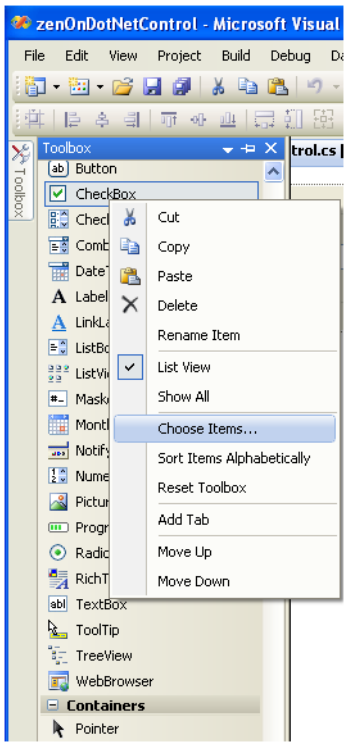
4. Controls verfügen normalerweise über Eigenschaften. Öffnen Sie den Code Designer über **View Code** und fügen Sie die gewünschten extern verfügbaren Properties hinzu.
   In unserem Beispiel: Extern sichtbares Property „**UserText**" mit **get** und **set** Zugriff, das den Text der Textbox enthält:



5. Kompilieren Sie das Projekt.

   Das Windows Forms Control kann jetzt in anderen Windows Forms Projekten verwendet werden.

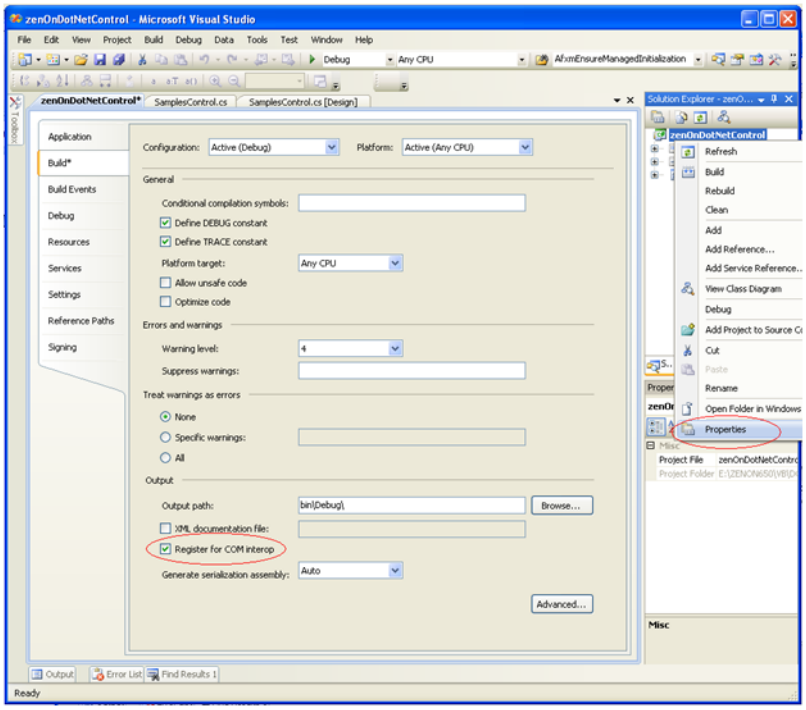
Wichtig: Das Control muss manuell über **Choose Items** in die Control Toolbox eingefügt werden.



5.3.2 .NET User Control in Dual Control umwandeln

Um das .NET in ein Dual Control umzuwandeln, muss zuerst die COM-Schnittstelle für ActiveX aktiviert werden.

1. Öffnen Sie das Projekt und aktivieren Sie unter den **Build-**Einstellungen die Eigenschaft **Register for COM interop**:

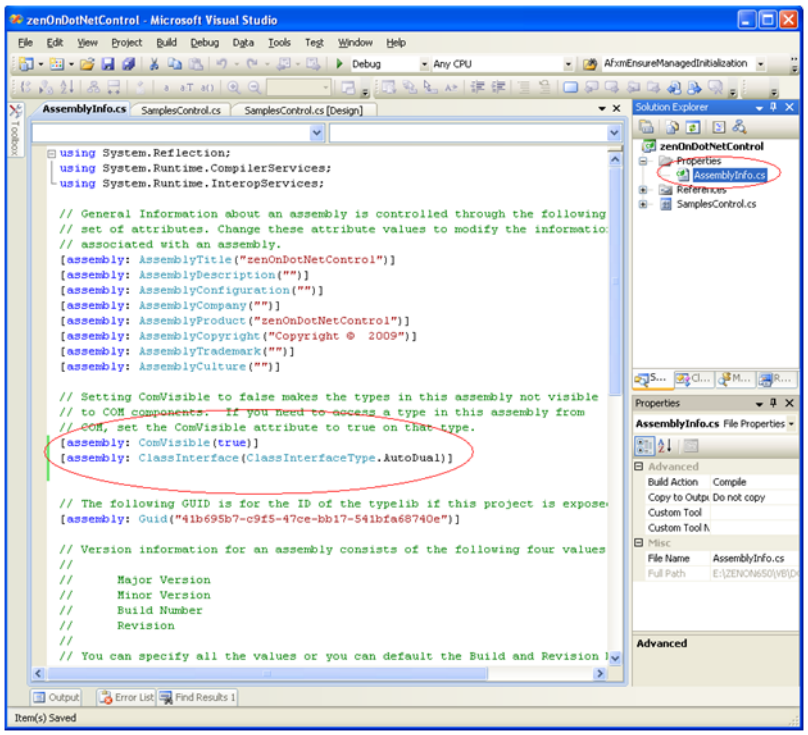


2. Öffnen Sie die Datei **AssemblyInfo.cs** und

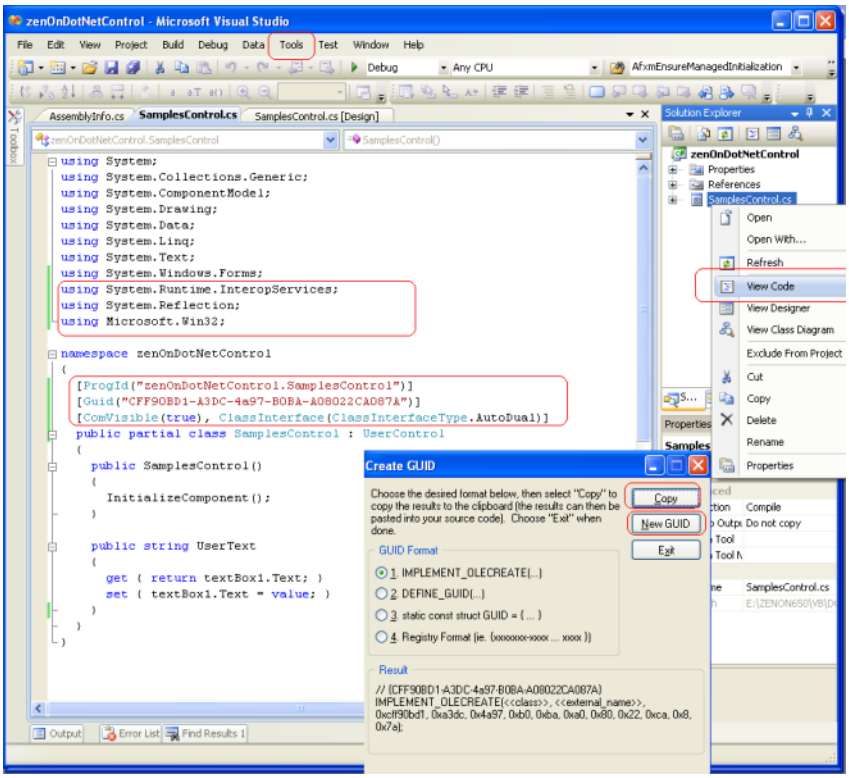
   - setzen Sie das Attribut **ComVisible** auf `true`
   - fügen Sie das Attribut **ClassInterface** hinzu

   [**assembly**: ComVisible(**true**)]

[assembly: ClassInterface(ClassInterfaceType.AutoDual)]



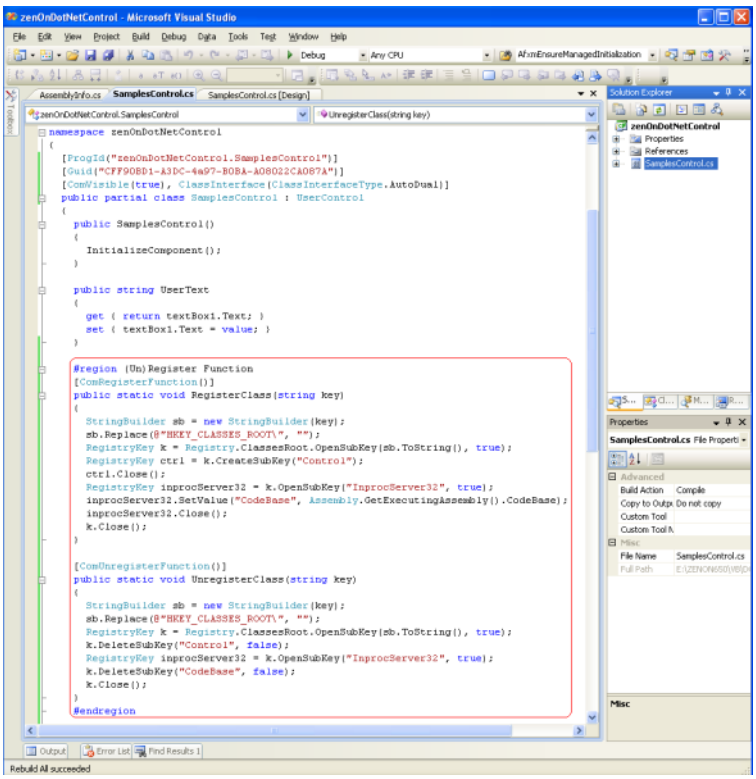
3. Öffnen Sie den Codedesigner über **View Code** und fügen Sie die notwendigen ActiveX Attribute und **using** Einträge hinzu. Erzeugen Sie über das Menü **Tools/Create GUID** eine neue GUID für das GUID-Attribut:



4. Damit das Control auch als ActiveX Oberflächen Control auswählbar ist, müssen in die Control-Klasse noch folgende Funktionen eingefügt werden:

   - RegisterClass

- UnregisterClass



Danach kann das Control in der Registry registriert werden.

5. Kompilieren Sie das Projekt erneut.

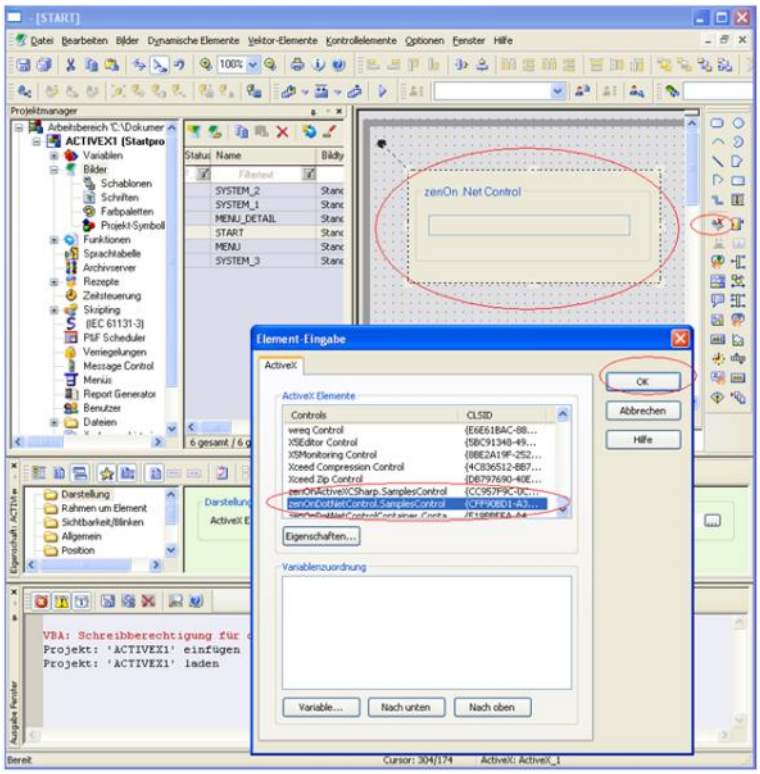
   Das Windows Form Control ist jetzt ActiveX fähig und wurde beim Rebuild automatisch registriert. Eine zusätzliche Typelib-Datei **zenOnDotNetControl.tlb** wurde im Outputverzeichnis erstellt.

6. Um das Control auf einem anderen Rechner einzusetzen:

   a) kopieren Sie die DLL-Datei und die TLB-Datei auf den Zielrechner

   b) registrieren Sie die Dateien über die Kommandozeile mit:
   %windir%\Microsoft.NET\Framework\v2.0.50727\regasm.exe zenOnDotNetControl.dll /tlb:zenOnDotNetControl.tlb

7. Fügen Sie das erweiterte Windows Form Control als ActiveX Control in den zenon Editor ein:



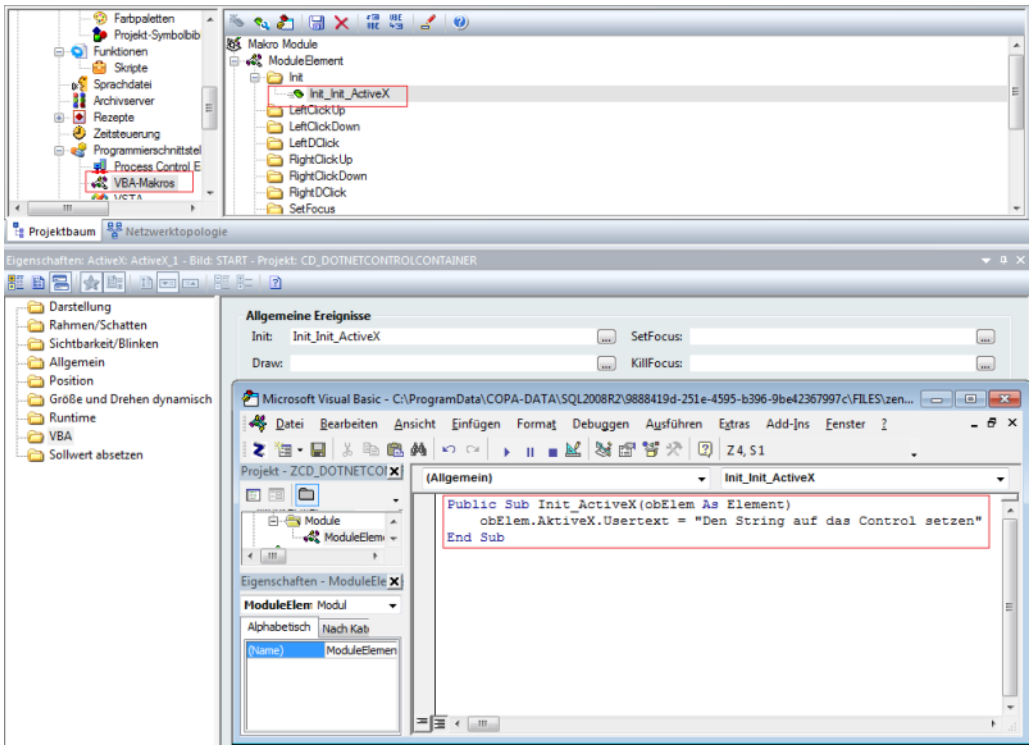
5.3.3 Im Editor über VBA mit ActiveX arbeiten

Um im zenon Editor auf die Eigenschaften des Controls zugreifen zu können:

1. Erzeugen Sie im zenon Editor im Knoten **Programmierschnittstellen/VBA-Makros** ein neues **Init** Makro mit der Bezeichnung **Init_ActiveX**.

   Mit diesem Makro kann man über **obElem.AktiveX** auf alle externen Properties zugreifen.

2. Weisen Sie dieses Makro über die Eigenschaften **VBA Makros/Init** des ActiveX Elements dem ActiveX Control zu.



BEISPIEL INIT MAKRO

```
Public Sub Init_ActiveX(obElem As Element)
    obElem.AktiveX.Usertext = "Den String auf das Control setzen"
End Sub
```

5.3.4 zenon Variablen mit dem .NET User Control verbinden

In zenon haben Sie die Möglichkeit, ein ActiveX Control um spezielle Funktionen zu erweiterten und darin auf die zenon API zuzugreifen.

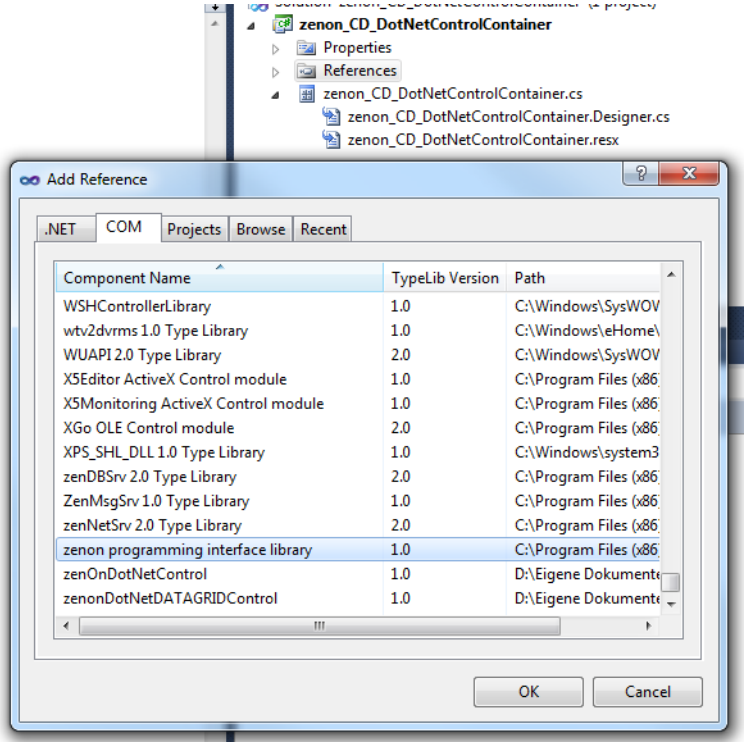
BENÖTIGTE METHODEN

- public bool zenOnInit (auf Seite 42) (Wird beim Control Initialisieren in der zenon Runtime aufgerufen.)
- public bool zenOnInitED (auf Seite 42) (Wird im Editor verwendet.)
- public bool zenOnExit() (auf Seite 43) (Wird beim Control-Zerstören in der zenon Runtime aufgerufen.)

- public bool zenOnExitED() (auf Seite 43) (Wird im Editor verwendet.)
- public short CanUseVariables() (auf Seite 43) (Unterstützt das Verknüpfungen von Variablen.)
- public short VariableTypes() (auf Seite 43) (Vom Control unterstützte Datentypen)
- public MaxVariables() (auf Seite 44) (Maximale Anzahl der Variablen, die mit dem Control verknüpft werden können.)

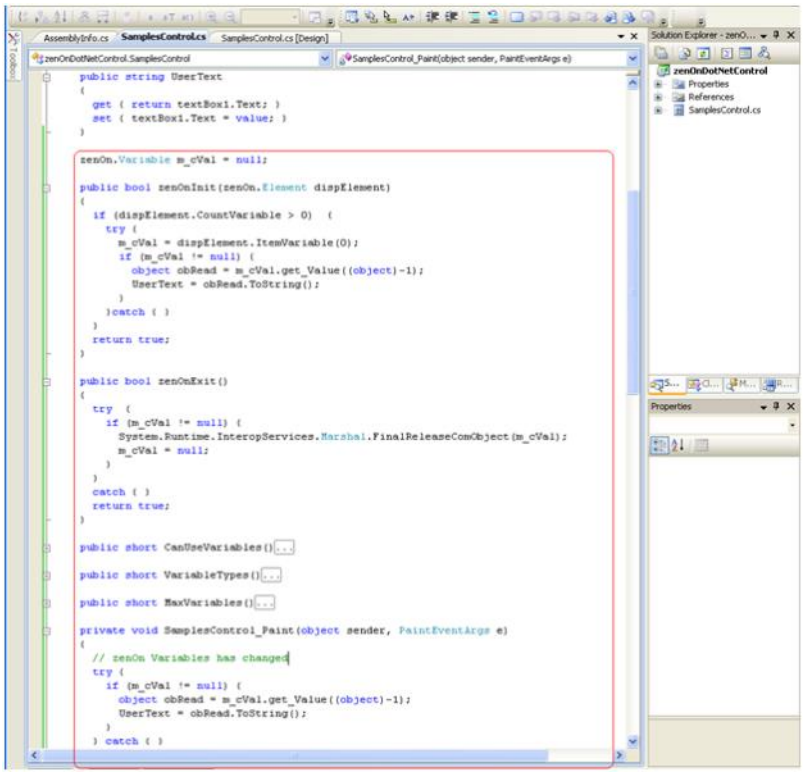
REFERENZ HINZUFÜGEN

1. Wählen Sie in Microsoft Visual Studio unter **Add References** die zenon Runtime-Objektbibliothek aus, damit man im Control auf die zenon API zugreifen kann.



2. Fügen Sie die erweiterten Funktionen in den Klassencode des Controls ein, um auf die gesamte zenon API zuzugreifen.

In unserem Beispiel wird das COM-Objekt einer zenon Variable in einem **Member** zwischengespeichert, um später im **Paint** Event des Controls darauf zugreifen zu können.



```
public string UserText
{
  get { return textBox1.Text; }
  set { textBox1.Text = value; }
}

zenOn.Variable m_cVal = null;

public bool zenOnInit(zenOn.Element dispElement)
{
  if (dispElement.CountVariable > 0)  {
    try {
      m_cVal = dispElement.ItemVariable(0);
      if (m_cVal != null) {
        object obRead = m_cVal.get_Value((object)-1);
        UserText = obRead.ToString();
      }
    }catch { }
  }
  return true;
}

public bool zenOnExit()
{
  try  {
    if (m_cVal != null) {
      System.Runtime.InteropServices.Marshal.FinalReleaseComObject(m_cVal);
      m_cVal = null;
    }
  }
  catch { }
  return true;
}

public short CanUseVariables()...

public short VariableTypes()...

public short MaxVariables()...

private void SamplesControl_Paint(object sender, PaintEventArgs e)
{
  // zenOn Variables has changed
  try {
    if (m_cVal != null) {
      object obRead = m_cVal.get_Value((object)-1);
      UserText = obRead.ToString();
    }
  } catch { }
```

public bool zenOnInit(zenOn.Element dispElement)

Mit Hilfe dieser Methode wird in der Runtime dem ActiveX Control ein Zeiger auf das Dispatch Interface des Dynamischen Elements übergeben. Über diesen Zeiger können dann die mit dem Dynamischen Element verknüpften zenon Variablen angesprochen werden.

Die Reihenfolge der übergebenen Variablen stellen Sie im Element-Eingabe Dialog mit den Schaltflächen **Nach unten** oder **Nach oben** ein. Der Element-Eingabe Dialog öffnet sich, wenn Sie:

- das ActiveX Element doppelklicken oder
- im Kontextmenü **Eigenschaften** wählen oder
- im Eigenschaftenfenster im Knoten **Darstellung** die Eigenschaft **ActiveX Einstellungen** wählen

public bool zenOnInitED(zenOn.Element dispElement)

Entspricht public bool zenOnInit (auf Seite 42)und wird beim Öffnen des ActiveX im Editor (Doppelklick auf ActiveX) ausgeführt.

public bool zenOnExit()

Diese Methode wird von der zenon Runtime beim Schließen des ActiveX Controls aufgerufen. Hier sollten alle Dispatch Pointer auf Variablen freigegeben werden.

public bool zenOnExitED()

Entspricht public bool zenOnExit() (auf Seite 43) und wird beim Schließen des ActiveX im Editor ausgeführt. Damit kann man auch im Editor auf Änderungen, etwas Wertänderungen, reagieren.

public short CanUseVariables()

Diese Methode gibt `1` zurück, wenn das Control zenon Variablen verwenden kann, und `0` wenn nicht.

- `1`: Für das Dynamische Element können(über die Schaltfläche **Variablen**) nur zenon Variablen mit den über die Methode **`VariableTypes`** angegebenen Typen in der von der Methode **`MaxVariables`** angegebenen Anzahl angegeben werden.
- `0`: Gibt **`CanUseVariables`** `0` zurück oder besitzt das Control diese Methode nicht, können Variablen mit allen Typen ohne Beschränkung der Anzahl angegeben werden. Diese können in der Runtime aber nur über VBA verwendet werden.

public short VariableTypes()

Der von dieser Methode zurückgegebene Wert wird als Maske für die verwendbaren Variablentypen in der Variablenliste verwendet. Der Wert ist eine **UND**-Verknüpfung aus folgenden Werten (definiert in **zenon32/dy_type.h**):

Parameter	Wert	Beschreibung
WORD	0x0001	entspricht Stelle 0
BYTE	0x0002	entspricht Stelle 1
BIT	0x0004	entspricht Stelle 2
DWORD	0x0008	entspricht Stelle 3
FLOAT	0x0010	entspricht Stelle 4
DFLOAT	0x0020	entspricht Stelle 5
STRING	0x0040	entspricht Stelle 6
IN_OUTPUT	0x8000	entspricht Stelle 15

public MaxVariables()

Hier wird die Anzahl der Variablen festgelegt, die aus der Variablenliste ausgewählt werden können:

`1`: In der Variablenliste wird die Mehrfachauswahl deaktiviert. Es erfolgt eine Warnung, wenn trotzdem mehrere Variable ausgewählt werden.

6. WPF

Mit dem Dynamischen Element **WPF** können gültige WPF/XAML-Dateien in zenon eingebunden und dargestellt werden.

Eine ausführliche Dokumentation über WPF-Elemente in zenon sowie Tutorials für Designer, Entwickler und Projektanten finden Sie im Handbuch zenon WPF-Element.

Dieses Handbuch behandelt folgende Themen:

- Grundlagen
- Leitfaden für Designer
- Leitfaden für Entwickler
- Projektierung in zenon