zenon Handbuch

Controls

v.7.50

Inhaltsverzeichnis

1. Willkommen bei der COPA-DATA Hilfe

ALLGEMEINE HILFE

Falls Sie in diesem Hilfekapitel Informationen vermissen oder Wünsche für Ergänzungen haben, wenden Sie sich bitte per E-Mail an documentation@copadata.com (mailto: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 (mailto:support@copadata.com) erreichen.

LIZENZEN UND MODULE

Sollten Sie feststellen, dass Sie weitere Module oder Lizenzen benötigen, sind unsere Mitarbeiter unter sales@copadata.com (mailto: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 46) (Implementierung in zenon siehe auch .NET Controls im Handbuch Bilder.)
- ActiveX (auf Seite 12) (Implementierung in zenon siehe auch ActiveX im Handbuch Bilder.)
- WPF (auf Seite 79)

Info

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

Lizenzinformation

In Standardlizenz für Editor und Runtime enthalten.

Achtung

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

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-RT 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.FinalReleaseComObject(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 10) angegebenen Typen in der von der Methode `MaxVariables` (auf Seite 10) 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 11) 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 11) 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 12)
- Beispiel LatchedSwitch (C++) (auf Seite 15)
- Beispiel CD_SliderCtrl (C++) (auf Seite 24)
- Beispiel .NET Control als ActiveX (C#) (auf Seite 32)

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 12) unterstützen.

4.1.1 Methoden

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

- CanUseVariables (auf Seite 9)
- MaxVariables (auf Seite 10)
- VariableTypes (auf Seite 10)
- zenonExit (auf Seite 11)
- zenonExitEd (auf Seite 11)
- zenonInit (auf Seite 11)

- zenonInitEd (auf Seite 11)

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 10) angegebenen Typen in der von der Methode `MaxVariables` (auf Seite 10) 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 11) 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 11) 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 19)
- VariableTypes (auf Seite 19)
- MaxVariables (auf Seite 19)
- zenonInit (auf Seite 20)
- zenonExit (auf Seite 21)

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 28)
- VariableTypes (auf Seite 28)
- MaxVariables (auf Seite 29)
- zenonInit (auf Seite 29)
- zenonExit (auf Seite 30)

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 10) 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:

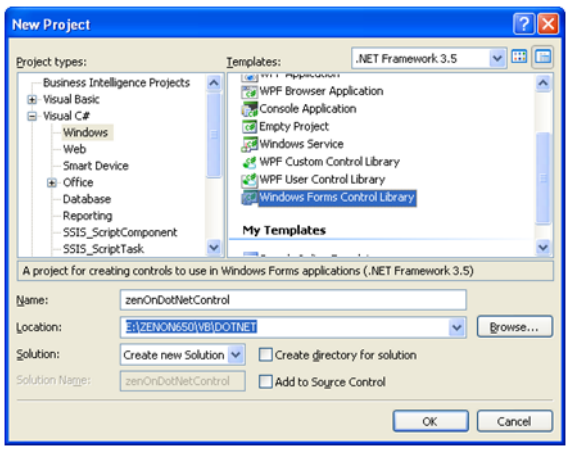
1. Windows Form Control erzeugen (auf Seite 33)
2. .NET User Control in Dual Control umwandeln (auf Seite 36)
3. Im Editor über VBA mit ActiveX arbeiten (auf Seite 41)
4. zenon Variablen mit dem .NET User Control verbinden (auf Seite 42)

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

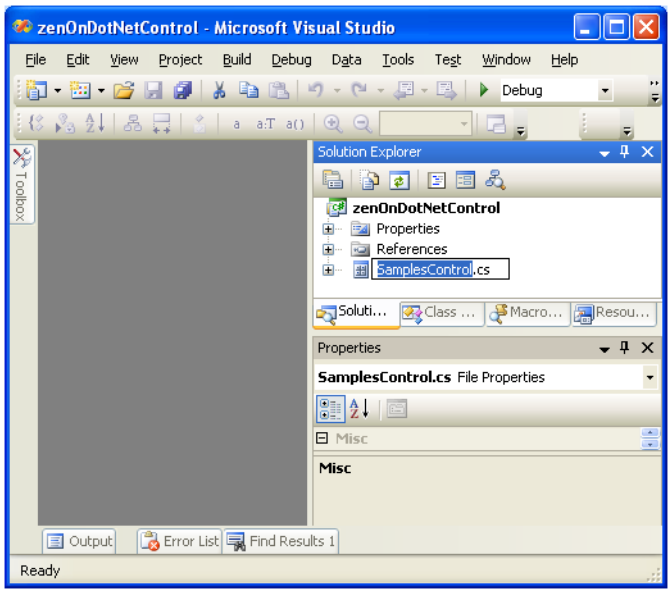
4.4.1 Windows Form Control erzeugen

Um ein Windows Form Control zu erzeugen:

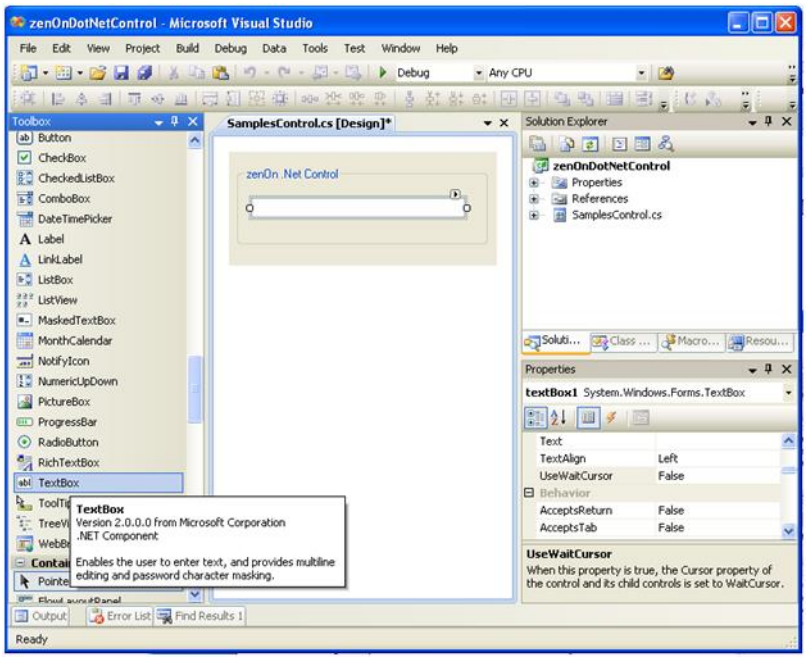
1. Starten Sie Visual Studio 2008 und erzeugen Sie ein neues Windows **Form Control Library** Projekt:



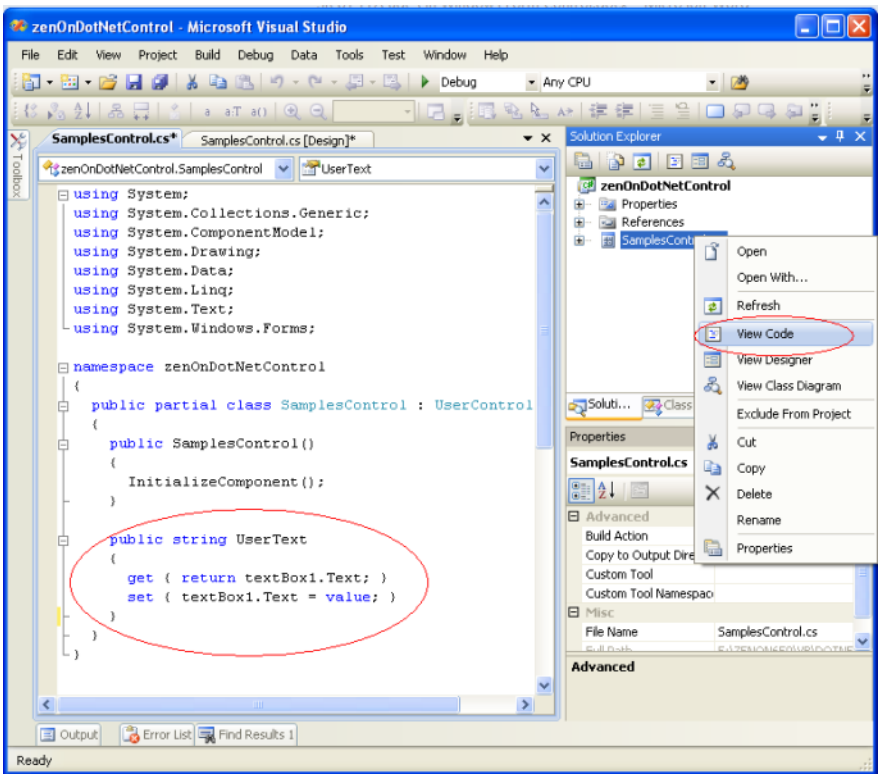
2. Benennen Sie das Default Control auf den gewünschten Controlnamen um.
   In unserem Beispiel: **SampesControl.cs**.



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



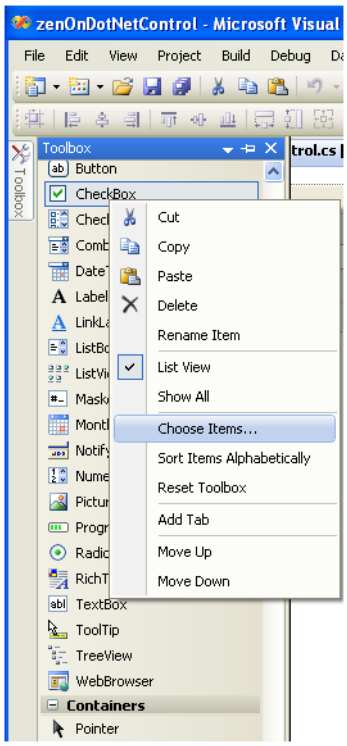
4. Controls verfügen normalerweise über Eigenschaften. Öffnen Sie den Code Designer über **View Code** und fügen Sie die gewünschten extern verfügbaren Properties hinzu.
   In unserem Beispiel: Extern sichtbares Property „**UserText**" mit **get** und **set** Zugriff, das den Text der Textbox enthält:



5. Kompilieren Sie das Projekt.

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

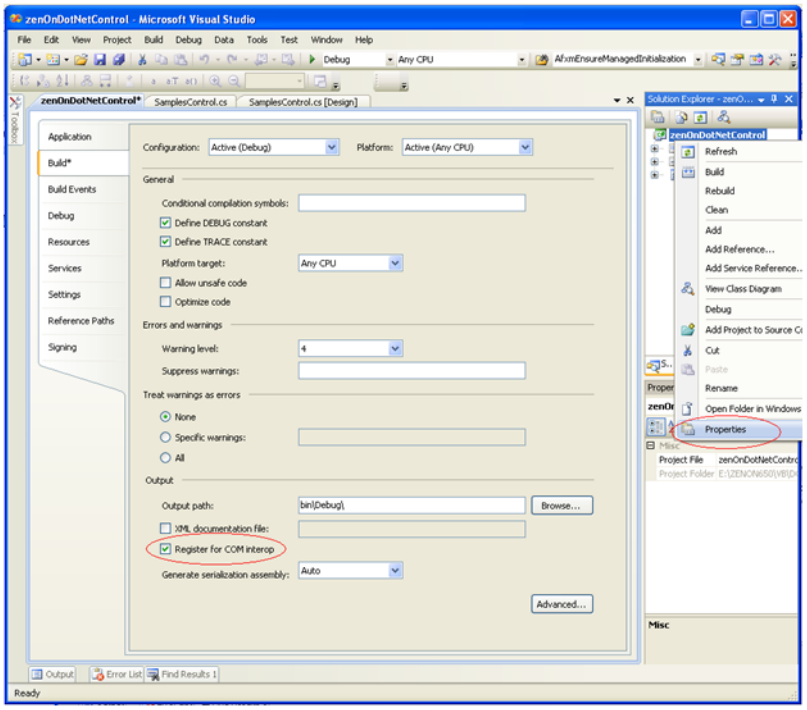
Wichtig: Das Control muss manuell über **Choose Items** in die Control Toolbox eingefügt werden.



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.

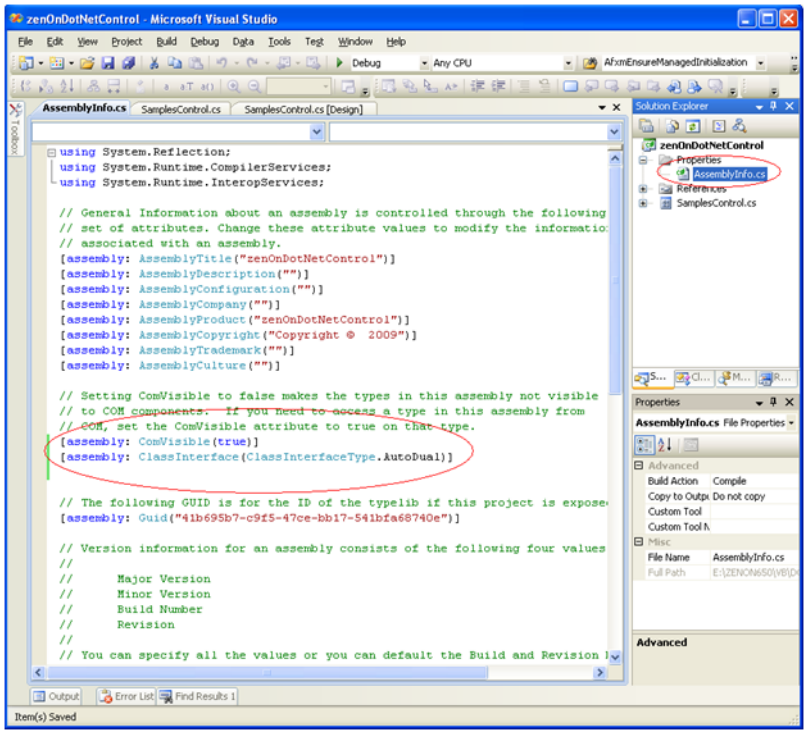
1. Öffnen Sie das Projekt und aktivieren Sie unter den **Build**-Einstellungen die Eigenschaft **Register for COM interop**:



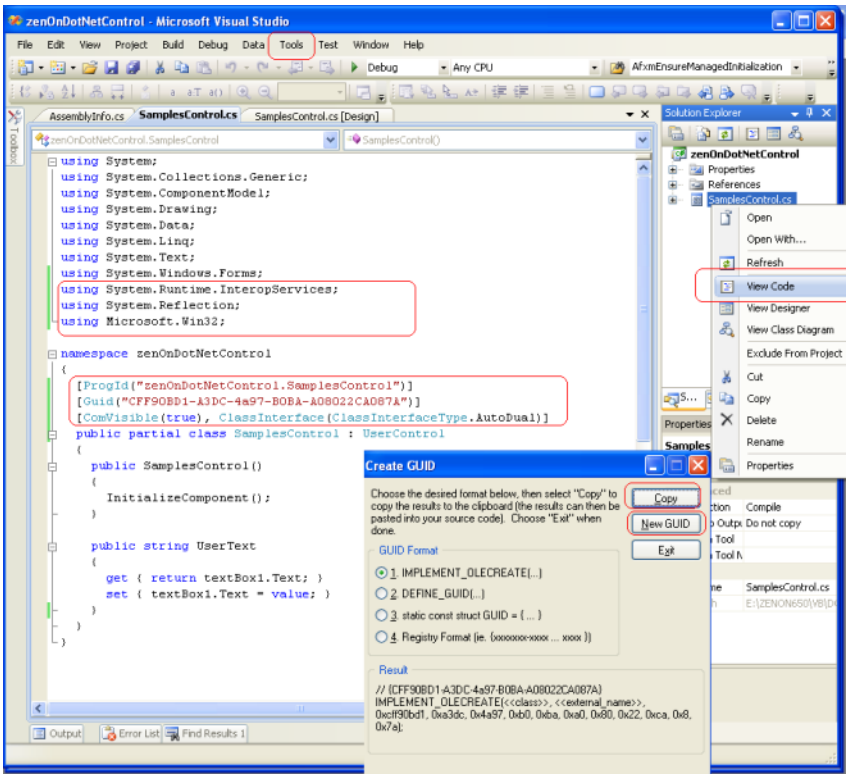
2. Öffnen Sie die Datei **AssemblyInfo.cs** und
   - setzen Sie das Attribut **ComVisible** auf `true`
   - fügen Sie das Attribut **ClassInterface** hinzu

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

[assembly: ClassInterface(ClassInterfaceType.AutoDual)]



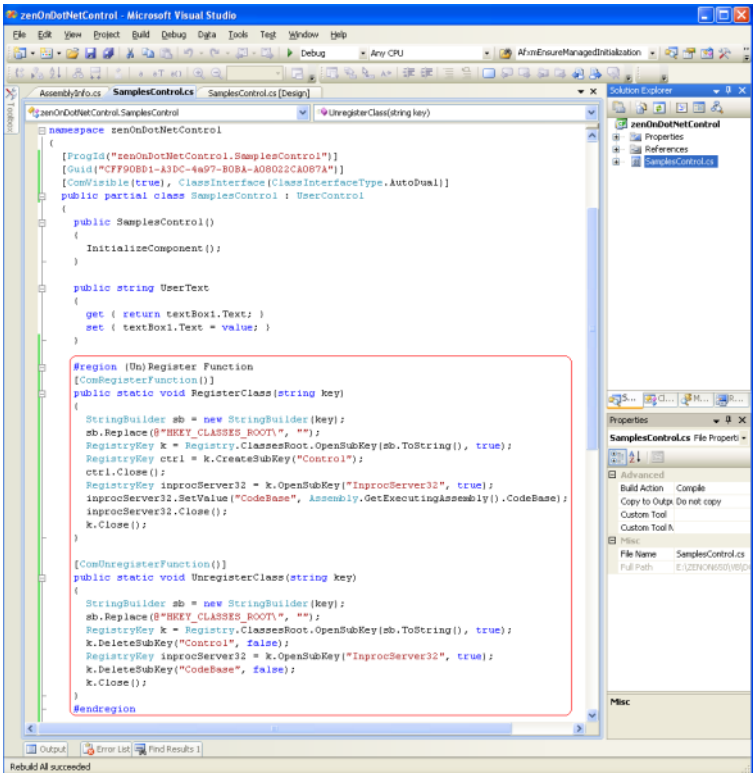
3. Öffnen Sie den Codedesigner über **View Code** und fügen Sie die notwendigen ActiveX Attribute und **using** Einträge hinzu. Erzeugen Sie über das Menü **Tools/Create GUID** eine neue GUID für das GUID-Attribut:



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.

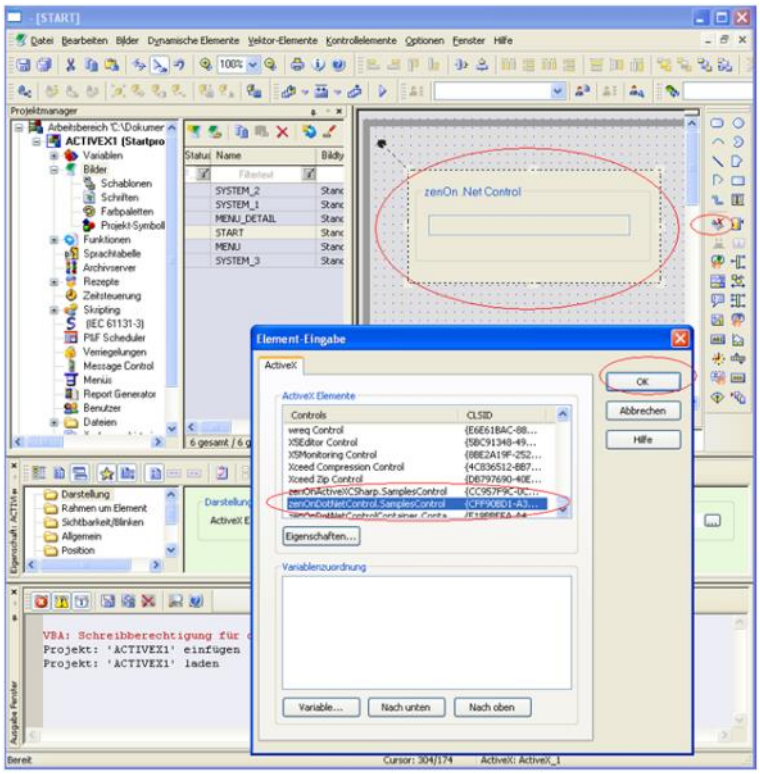
   Das Windows Form Control ist jetzt ActiveX fähig und wurde beim Rebuild automatisch registriert. Eine zusätzliche Typelib-Datei **zenOnDotNetControl.tlb** wurde im Outputverzeichnis erstellt.

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:



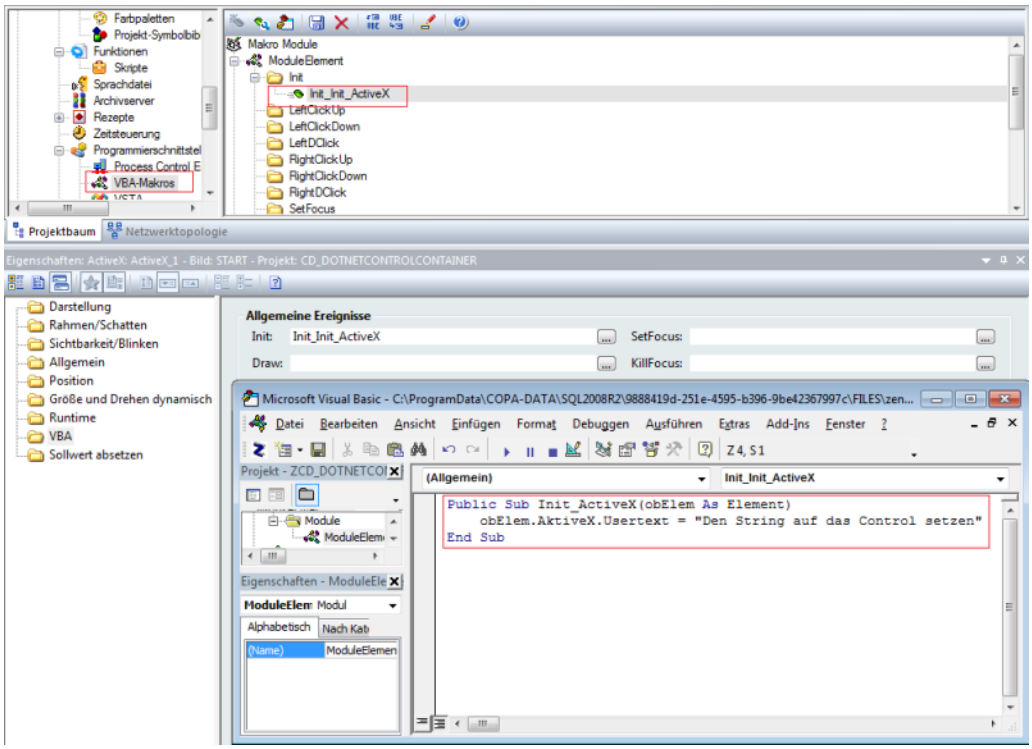
4.4.3 Im Editor über VBA mit ActiveX arbeiten

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.

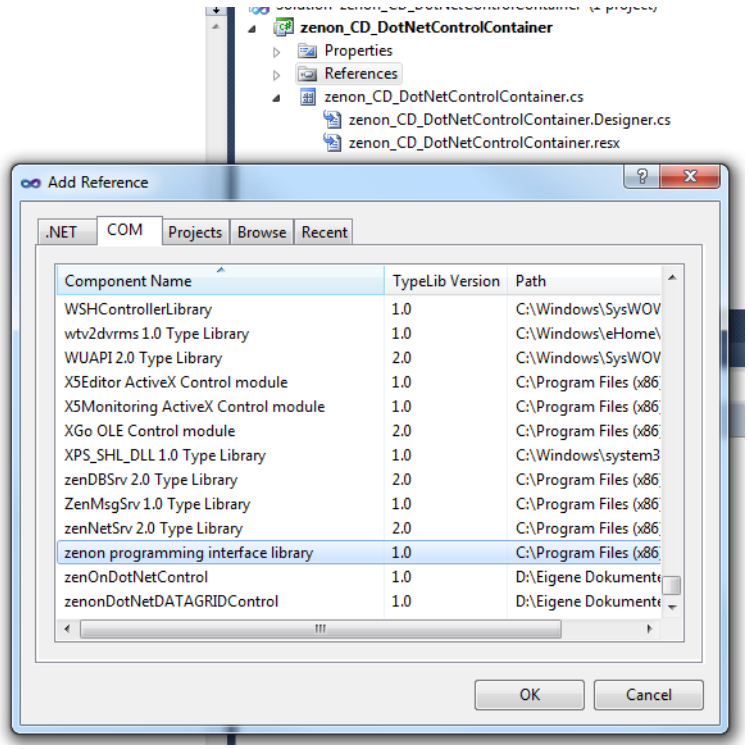
BENÖTIGTE METHODEN

- public bool zenOnInit (auf Seite 44) (Wird beim Control Initialisieren in der zenon Runtime aufgerufen.)
- public bool zenOnInitED (auf Seite 44) (Wird im Editor verwendet.)
- public bool zenOnExit() (auf Seite 45) (Wird beim Control-Zerstören in der zenon Runtime aufgerufen.)

- public bool zenOnExitED() (auf Seite 45) (Wird im Editor verwendet.)
- public short CanUseVariables() (auf Seite 45) (Unterstützt das Verknüpfungen von Variablen.)
- public short VariableTypes() (auf Seite 45) (Vom Control unterstützte Datentypen)
- public MaxVariables() (auf Seite 46) (Maximale Anzahl der Variablen, die mit dem Control verknüpft werden können.)

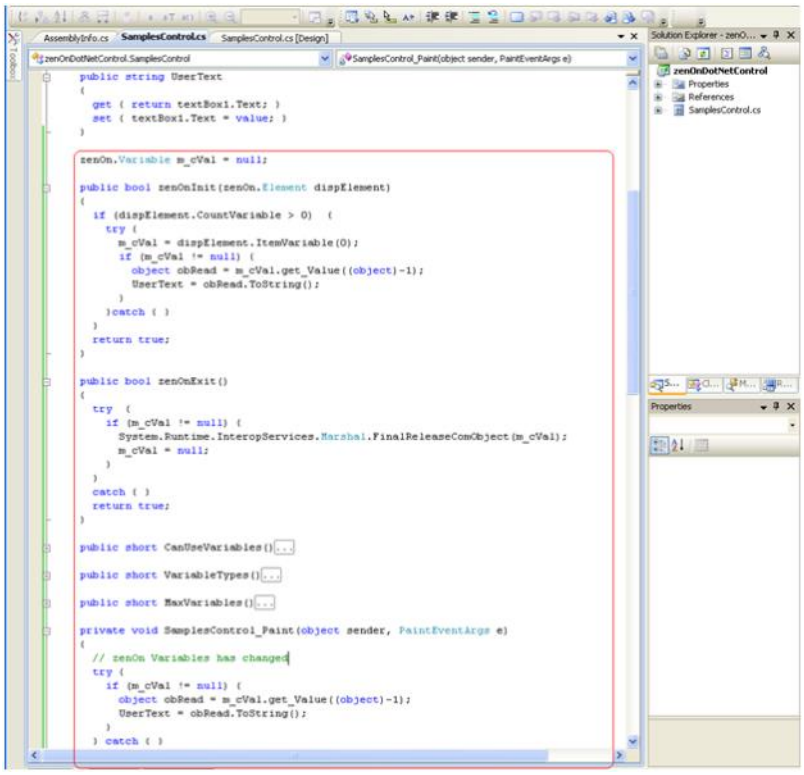
REFERENZ HINZUFÜGEN

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 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 44)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 45) 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 46)
- Beispiel .NET Control Container (auf Seite 47)
- Beispiel .NET Control als ActiveX (C#) (auf Seite 32)

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

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.

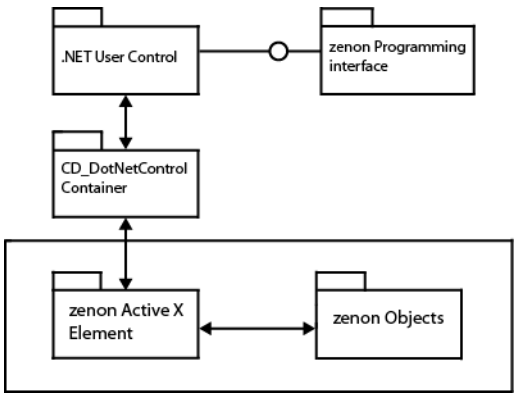
5.2 Beispiel .NET Control Container

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 49)

- CD_DotNetControlContainer und .NET User Control einfügen (auf Seite 58)
- den Zugriff auf das User Control über VSTA (VBA) ermöglichen (auf Seite 63)

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.

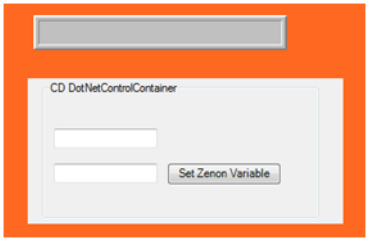
5.2.2 .Net User Control erstellen

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

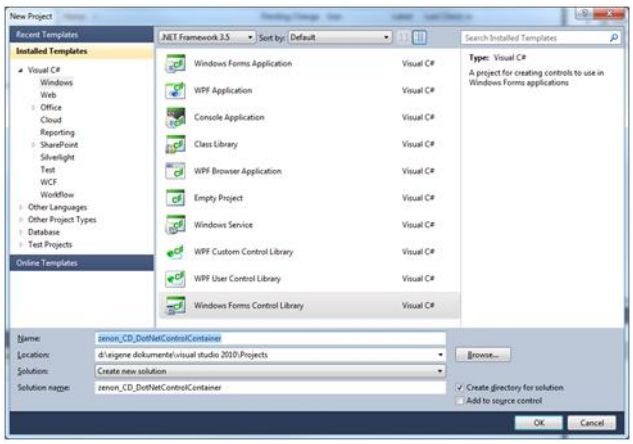
Die Screenshots für dieses Thema sind nur in englischer Sprache verfügbar.



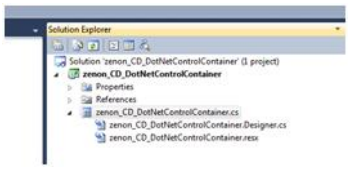
ARBEITSSCHRITTE

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

   Wichtig: Framework auf 3.5 einstellen!



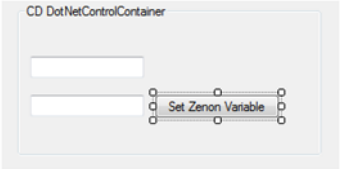
2. Danach benennen Sie die CS-Datei „**UserControl**" in „**zenon_CD_DotNetControlContainer.cs**" um. Die Dateien **Designer.cs** und **.resx** werden automatisch mit unbenannt.



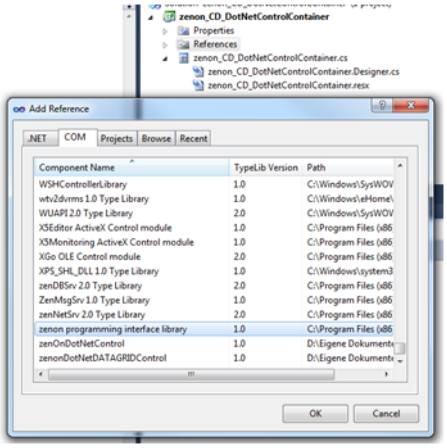
3. Im nächsten Schritt erstellen Sie das User Control. Dazu benötigen Sie zwei Textboxen für die Ein-und Ausgabe sowie einen Button zum Absetzen eines neuen Wertes auf die zenon Variable. Benennen Sie:

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

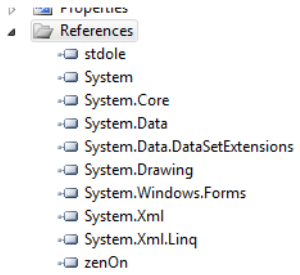
- den Button „**btnSetZenonVariable**"



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:

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Drawing;
5  using System.Data;
6  using System.Linq;
7  using System.Text;
8  using System.Windows.Forms;
9  using zenOn;
10
11 namespace zenon_CD_DotNetControlContainer
12 {
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
24     }
25 }
26
```

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

```
23         /// <summary>
24         /// This public Method will be called by the initialization of the control during
25         /// the zenon Runtime.
26         /// </summary>
27         /// <param name="dispElement"></param>
28         /// <returns></returns>
29         public bool zenOnInit(zenOn.Element dispElement)
30         {
31             //Check if zenon Variables are added to the
32             //Control
33             if (dispElement.CountVariable > 0)
34             {
35                 try
36                 {
37                     //Take the first zenon Variable and added
38                     //to the global Variable
39                     m_cVal = dispElement.ItemVariable(0);
40                     //Set Value to the TextBox
41                     txtGetZenonVariable.Text = m_cVal.get_Value(0).ToString();
42                 }
43                 catch { }
44             }
45             return true;
46         }
```

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

```
47         /// <summary>
48         /// This public Method will be calles by the release of the control during
49         /// the zenon Runtime.
50         /// </summary>
51         /// <returns></returns>
52         public bool zenOnExit()
53         {
54             try
55             {
56                 if (m_cVal != null)
57                 {
58                     //Release the zenon Variable (Com-Object)
59                     System.Runtime.InteropServices.Marshal.FinalReleaseComObject(m_cVal);
60                     m_cVal = null;
61                 }
62             }
63             catch { }
64             return true;
65         }
```

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

```
107        /// <summary>
108        /// This public Method is needed to link zenon Variables
109        /// to the control.
110        /// </summary>
111        /// <returns></returns>
112        public short CanUseVariables()
113        {
114            return 1; // Only tis Variable is supported
115        }
116
117        /// <summary>
118        /// This public Method returns the Type of
119        /// supported zenon Variables
120        /// </summary>
121        /// <returns></returns>
122        public short VariableTypes()
123        {
124            return short.MaxValue; // all Data Types supported
125        }
126
127        /// <summary>
128        /// This public Method returns the number of
129        /// supported zenon Variables
130        /// </summary>
131        /// <returns></returns>
132        public short MaxVariables()
133        {
134            return 1; // Only 1 Variable should linked to the Control
135        }
```

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;
    }
```

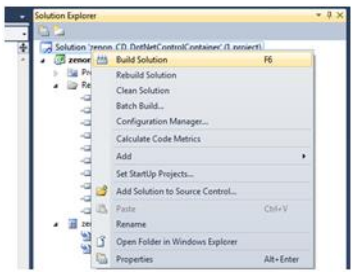
```
// <summary>
/// This public Method will be called by the release of the control during
/// the zenon Runtime.
/// </summary>
/// <returns></returns>
    public bool zenOnExit()
    {
        try
        {
            if (m_cVal != null)
            {
                //Release the zenon Variable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

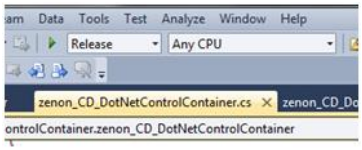
```
        /// After the value of the Variable changed the Control will be new painted and
the new Value
        /// will be set to the Textbox.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void zenon_CD_DotNetControlContainer_Paint(object 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.



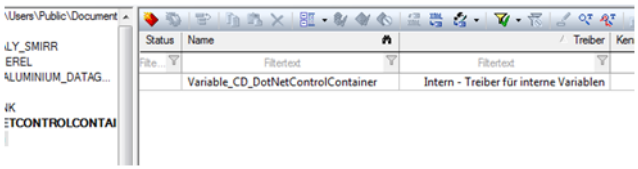
Dafür muss unbedingt in den Settings von **Debug** auf **Release** umgestellt werden.



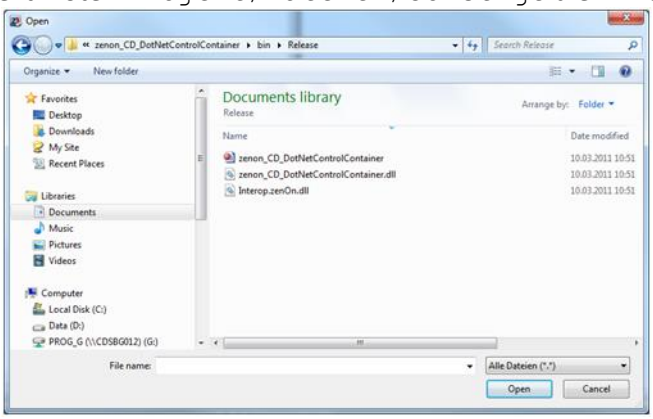
5.2.3 CD_DotNetControlContainer und .NET User Control einfügen

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

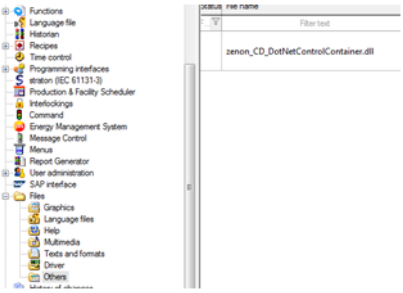
1. Legen Sie eine interne Variable vom Typ `String` an und setzen Sie die Stringlänge auf `30`.



2. Fügen Sie im zenon Projektknoten `Projekt/Dateien/Sonstige` die DLL des erstellten .NET User Controls ein.

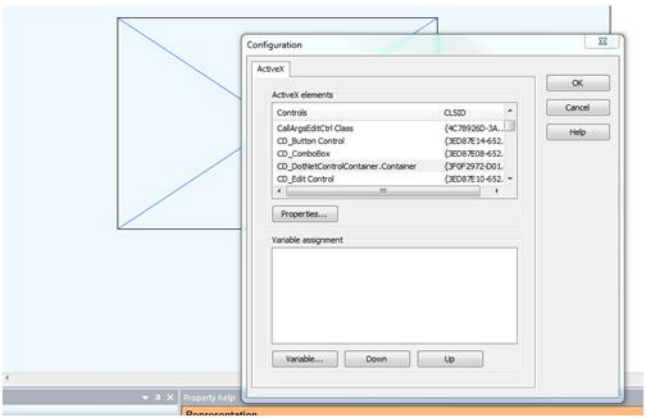


Die DLL befindet sich im Visual Studio Projekt-Ordner unter `bin\Release\zenon_CD_DotNetControlContainer.dll`.

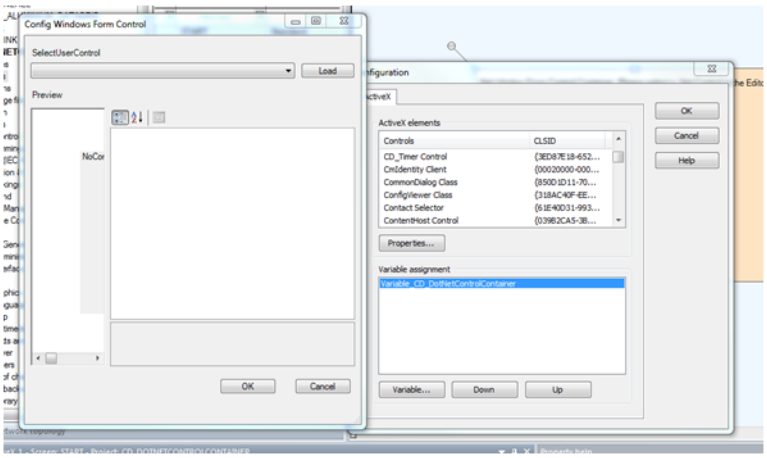


3. Im Projekt wählen Sie das ActiveX-Element und ziehen es in ein zenon Bild.
   - Der Dialog **Konfiguration** wird geöffnet.

- Wählen Sie das **CD_DotNetControlContainer.Container** Control aus.

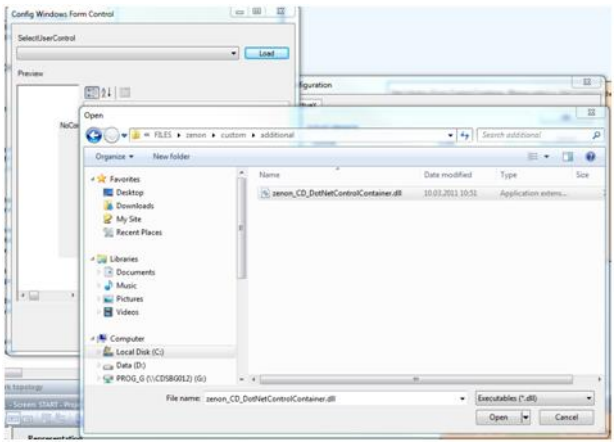


4. Um das .NET User Control in das **CD_DotNetControlContainer** Control einzubetten:
   - Klicken Sie auf die Schaltfläche **Eigenschaften.**
   - Ein neuer Dialog wird geöffnet.



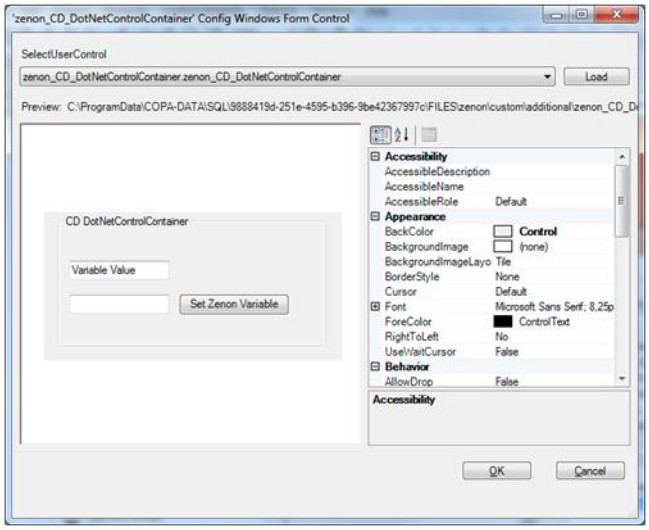
   - Mit Klick auf die Schaltfläche **Load** wählen Sie den Pfad des Projektordners, zum Beispiel: `C:\ProgramData\COPA-DATA\SQL\9888419d-251e-4595-b396-9be42367997c\FILES\zenon\custom\additional\zenon_CD_DotNetControlContainer.dll`

Durch das Einfügen der DLL im Ordner `additional` wird das Control automatisch beim Kopieren oder Laden der Runtime-Dateien auf einen anderen Rechner mit geschickt. Dadurch geht die Verknüpfung nicht verloren.

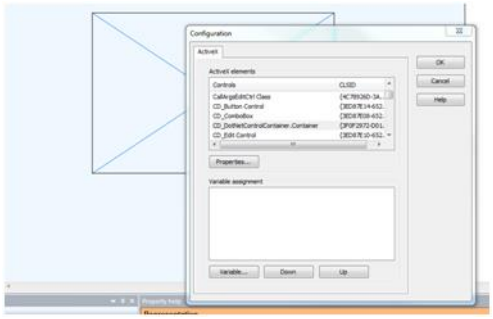


Nun sollte das .NET User Control angezeigt werden.

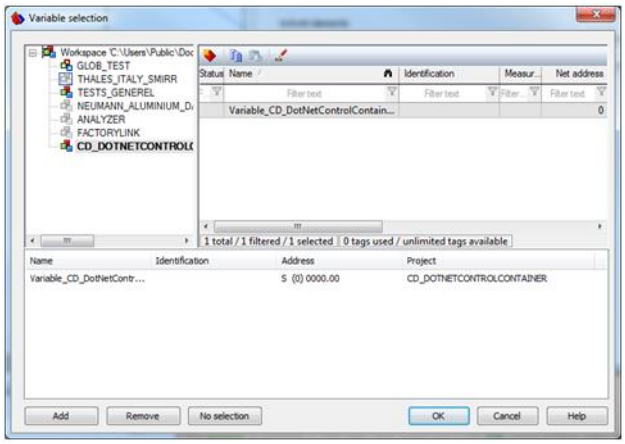
Bestätigen Sie den Dialog mit Klick auf **OK**.



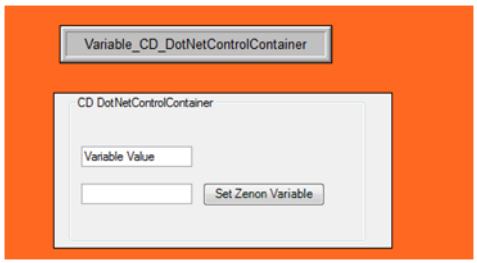
5. Im letzten Schritt verknüpfen Sie über die Schaltfläche **Variablen** eine Variable mit dem Control verknüpfen.



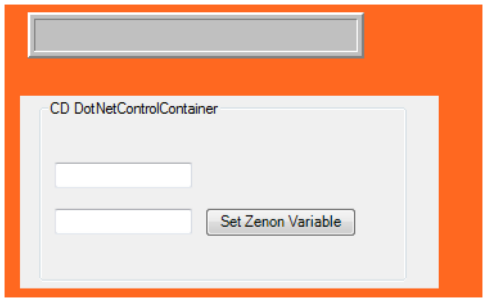
Die erste ausgewählte Variable wird automatisch über die `public` Methode **zenonInit** mit der global definierten Variable (.NET UserControl) verknüpft. Die Verlinkung mit dem Control erfolgt aber erst nach dem Start der Runtime.



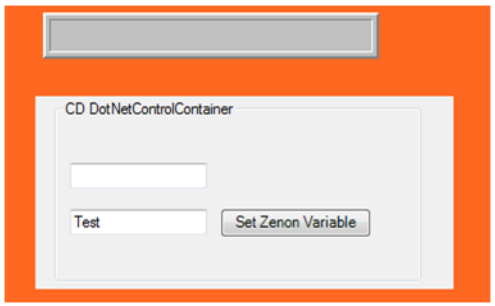
Dann verlinken Sie noch die interne Variable mit einem Textelement.



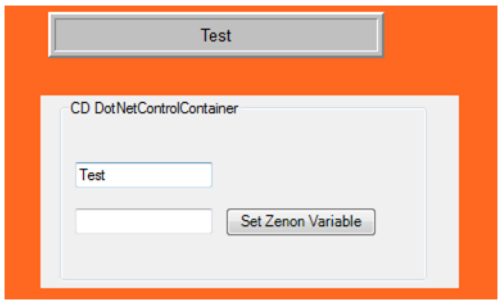
6. Nach dem Runtime Start ist das Control zunächst leer.



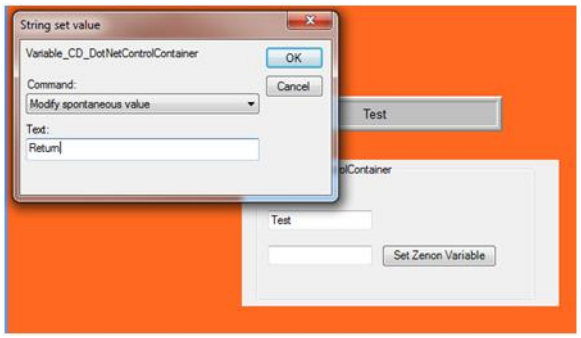
Wird ein Wert in die zweite Textbox geschrieben und danach mit dem Button **Set zenon Variable** bestätigt, dann wird der Wert auf die zenon Variable geschrieben. (Das **btnSetZenonVariable_Click** Event wird ausgeführt.)



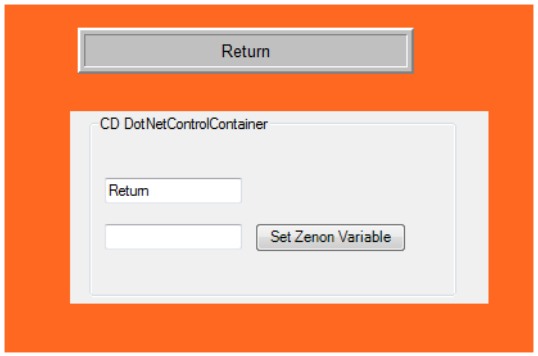
Dieser wird auch im zenon Textelement angezeigt.



Wird der Wert direkt im zenon Text-Element geändert,



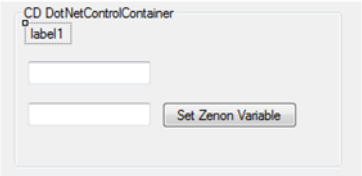
dann wird der Wert direkt über das **Paint** Event des .NET Controls in die erste Textbox geschrieben.



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.

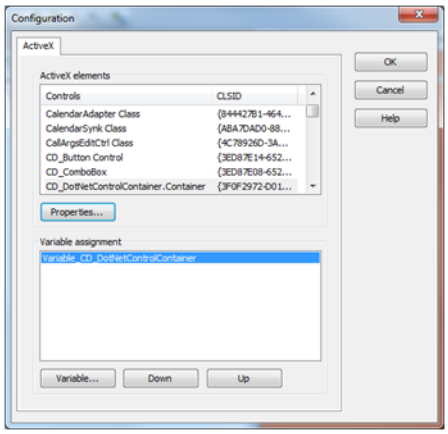


2. Erweitern Sie den Code um ein Property (**Information**) und fügen **get** und **set** Eigenschaften zu dem Property hinzu. Diese erlauben es, den Text des Labels zu lesen und zu schreiben.

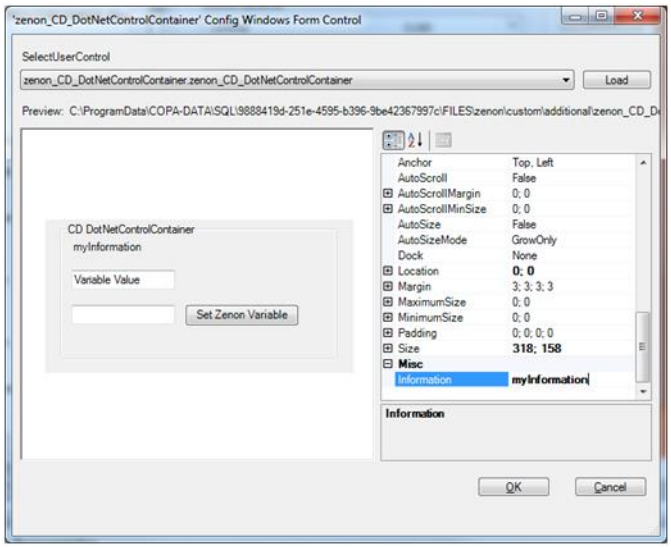
```
13  public partial class zenon_CD_DotNetControlContainer : UserControl
14  {
15      //This will be needed to get the zenon Variable Container
16      zenOn.Variable m_cVal = null;
17
18      public zenon_CD_DotNetControlContainer()
19      {
20          InitializeComponent();
21      }
22
23      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.

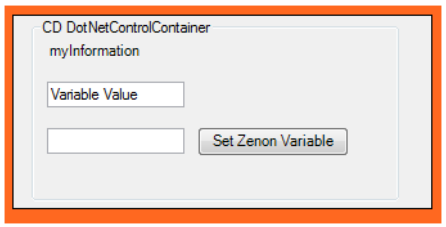
4. Klicken Sie im zenon Editor auf das ActiveX und öffnen Sie das Eigenschaftenfenster.



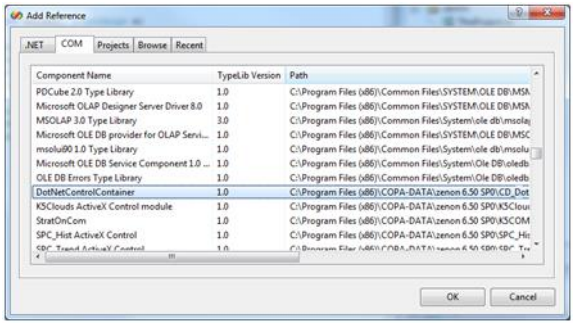
Sie können nun die neue Eigenschaft **Information** im Auswahlfenster des Controls sehen und auch einen Wert setzen.



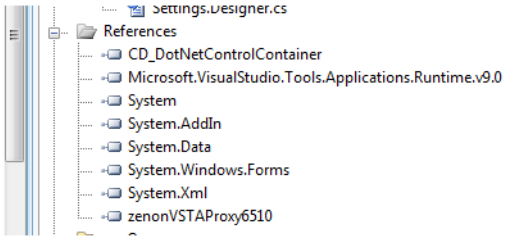
Dieser Wert wird auch im Control gesetzt („myInformation")



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:

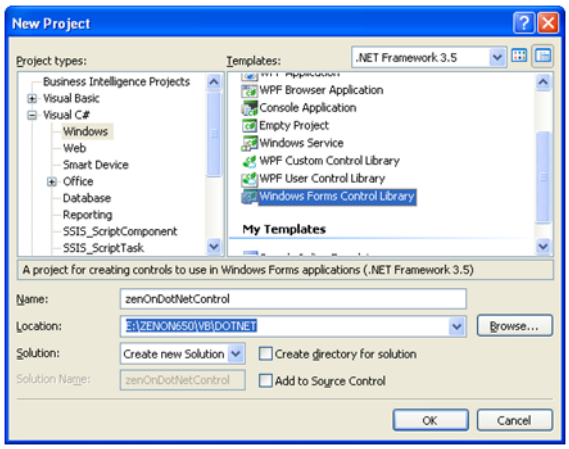
1. Windows Form Control erzeugen (auf Seite 33)
2. .NET User Control in Dual Control umwandeln (auf Seite 36)
3. Im Editor über VBA mit ActiveX arbeiten (auf Seite 41)
4. zenon Variablen mit dem .NET User Control verbinden (auf Seite 42)

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

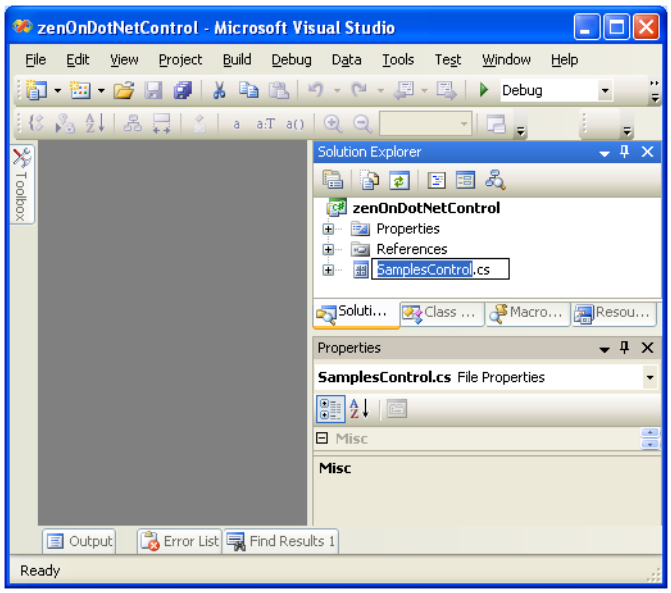
5.3.1 Windows Form Control erzeugen

Um ein Windows Form Control zu erzeugen:

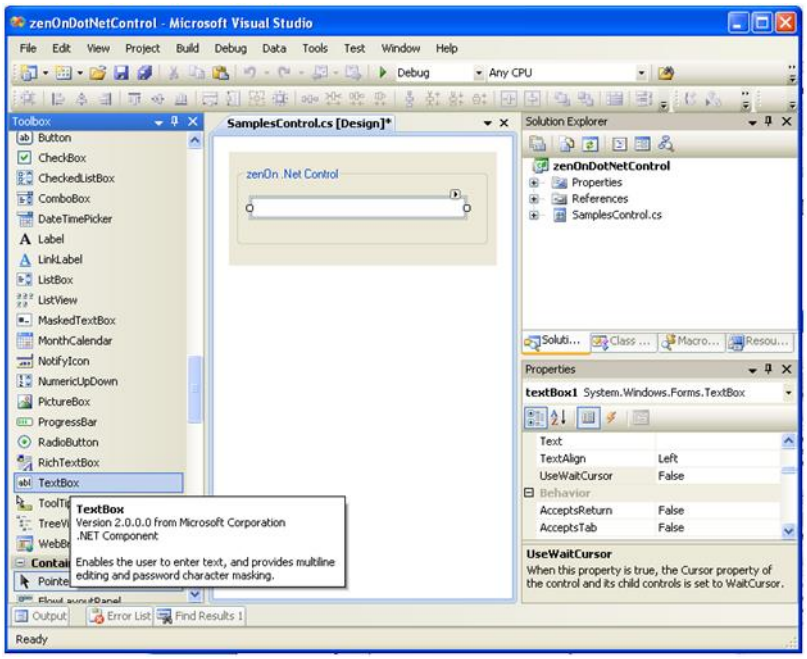
1. Starten Sie Visual Studio 2008 und erzeugen Sie ein neues Windows **Form Control Library** Projekt:



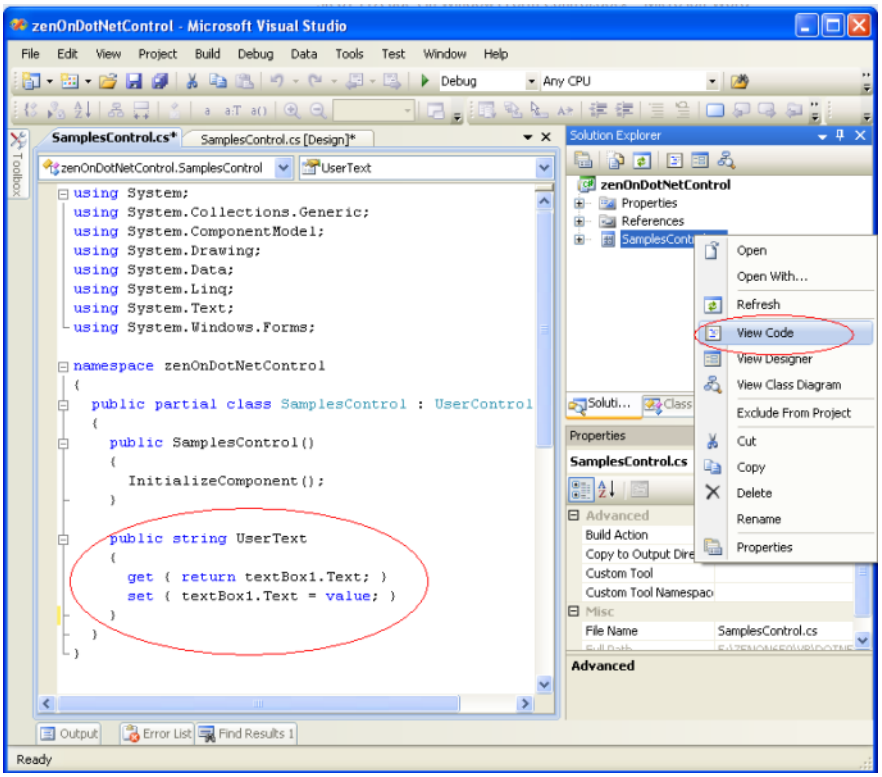
2. Benennen Sie das Default Control auf den gewünschten Controlnamen um.
   In unserem Beispiel: **SampesControl.cs**.



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



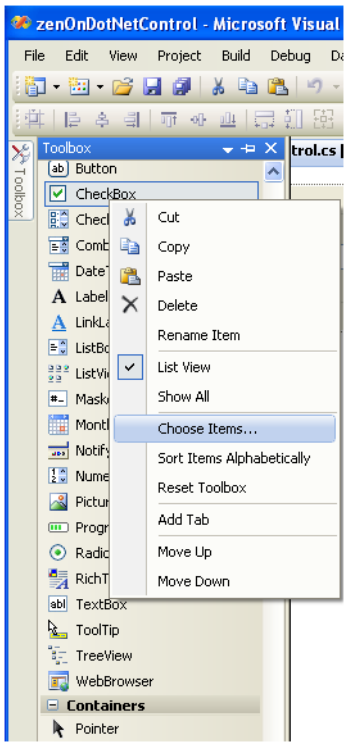
4. Controls verfügen normalerweise über Eigenschaften. Öffnen Sie den Code Designer über **View Code** und fügen Sie die gewünschten extern verfügbaren Properties hinzu.
   In unserem Beispiel: Extern sichtbares Property „**UserText**" mit **get** und **set** Zugriff, das den Text der Textbox enthält:



5. Kompilieren Sie das Projekt.

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

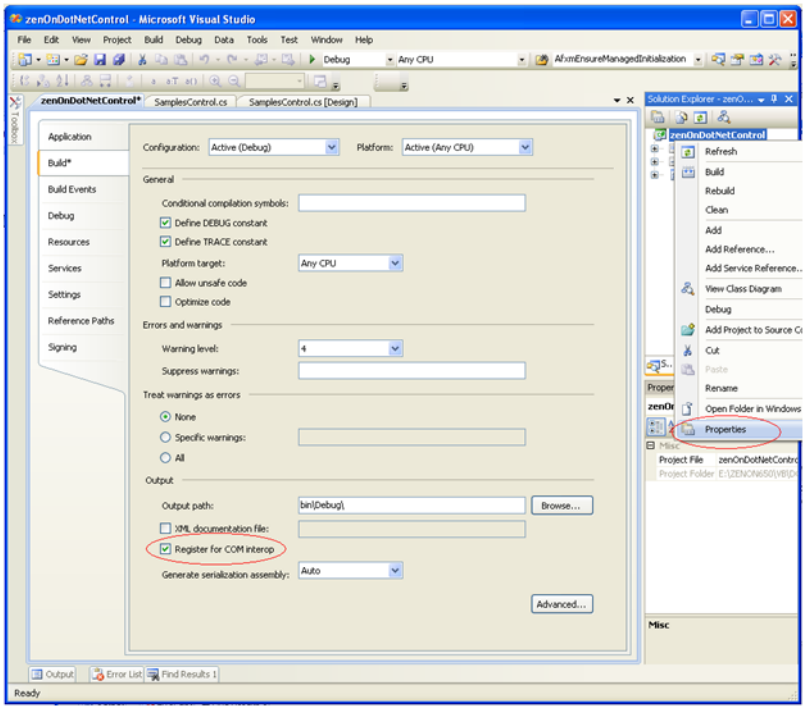
Wichtig: Das Control muss manuell über **Choose Items** in die Control Toolbox eingefügt werden.



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.

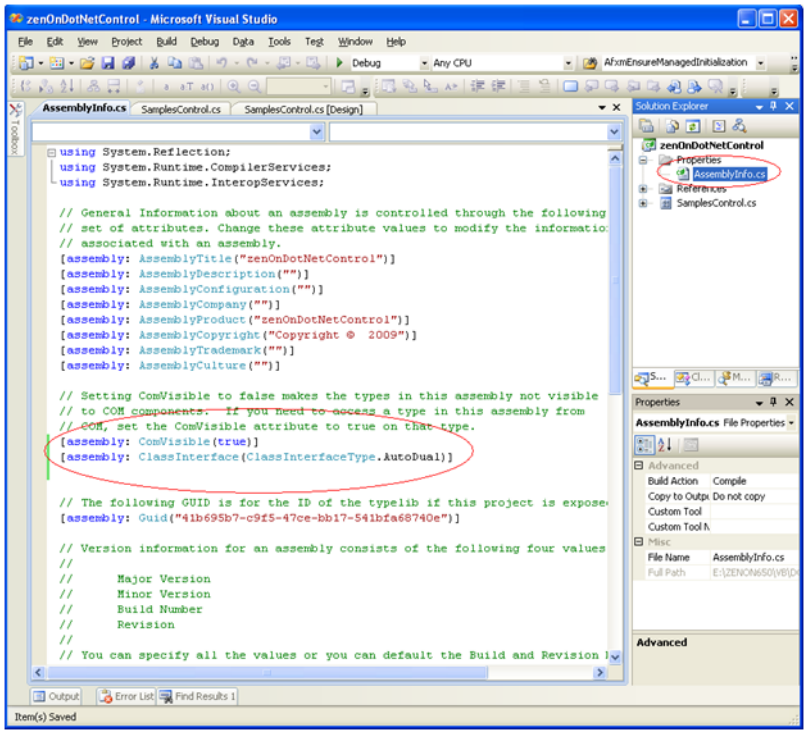
1. Öffnen Sie das Projekt und aktivieren Sie unter den **Build**-Einstellungen die Eigenschaft **Register for COM interop**:



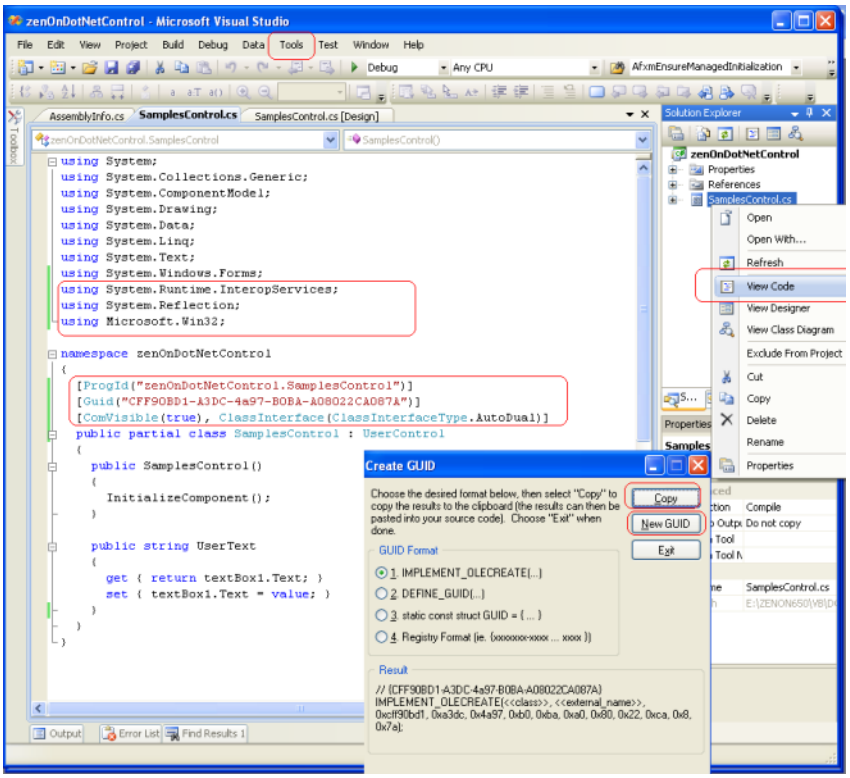
2. Öffnen Sie die Datei **AssemblyInfo.cs** und
   - setzen Sie das Attribut **ComVisible** auf `true`
   - fügen Sie das Attribut **ClassInterface** hinzu

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

[assembly: ClassInterface(ClassInterfaceType.AutoDual)]



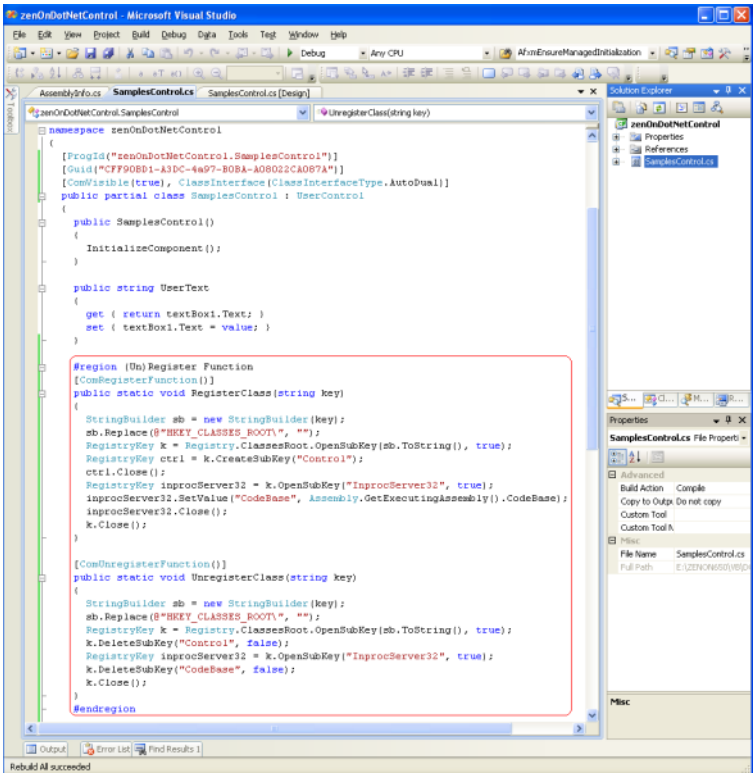
3. Öffnen Sie den Codedesigner über **View Code** und fügen Sie die notwendigen ActiveX Attribute und **using** Einträge hinzu. Erzeugen Sie über das Menü **Tools/Create GUID** eine neue GUID für das GUID-Attribut:



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.

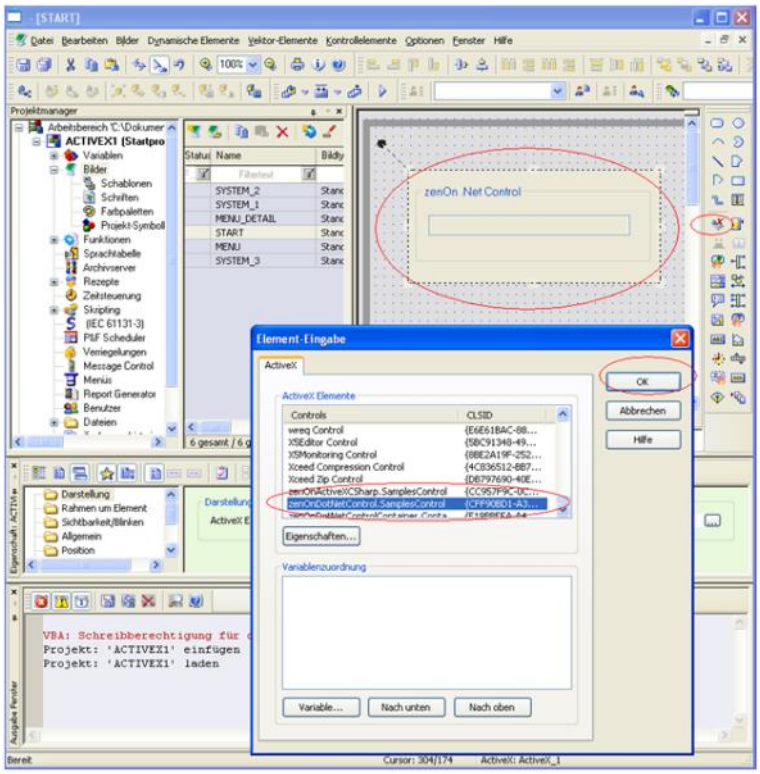
   Das Windows Form Control ist jetzt ActiveX fähig und wurde beim Rebuild automatisch registriert. Eine zusätzliche Typelib-Datei **zenOnDotNetControl.tlb** wurde im Outputverzeichnis erstellt.

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:



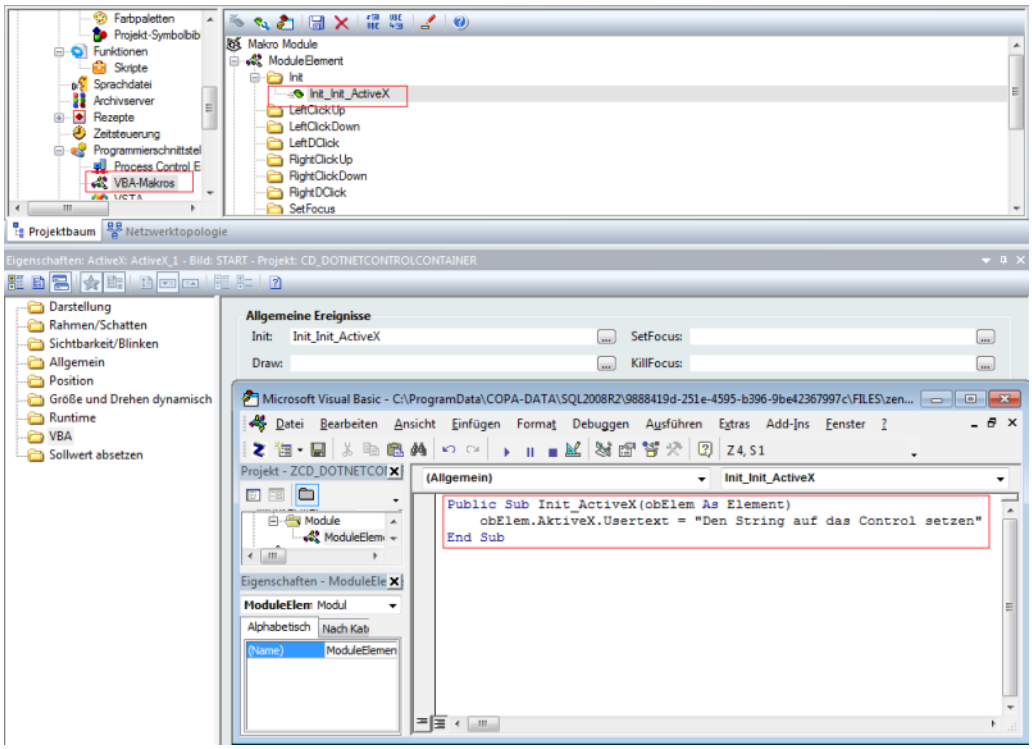
5.3.3 Im Editor über VBA mit ActiveX arbeiten

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.

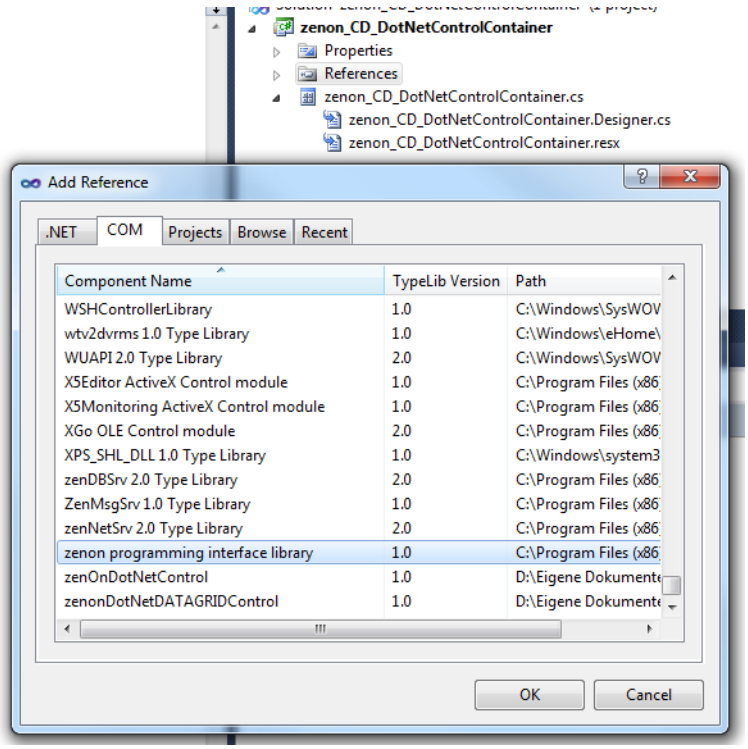
BENÖTIGTE METHODEN

- public bool zenOnInit (auf Seite 44) (Wird beim Control Initialisieren in der zenon Runtime aufgerufen.)
- public bool zenOnInitED (auf Seite 44) (Wird im Editor verwendet.)
- public bool zenOnExit() (auf Seite 45) (Wird beim Control-Zerstören in der zenon Runtime aufgerufen.)

- public bool zenOnExitED() (auf Seite 45) (Wird im Editor verwendet.)
- public short CanUseVariables() (auf Seite 45) (Unterstützt das Verknüpfungen von Variablen.)
- public short VariableTypes() (auf Seite 45) (Vom Control unterstützte Datentypen)
- public MaxVariables() (auf Seite 46) (Maximale Anzahl der Variablen, die mit dem Control verknüpft werden können.)

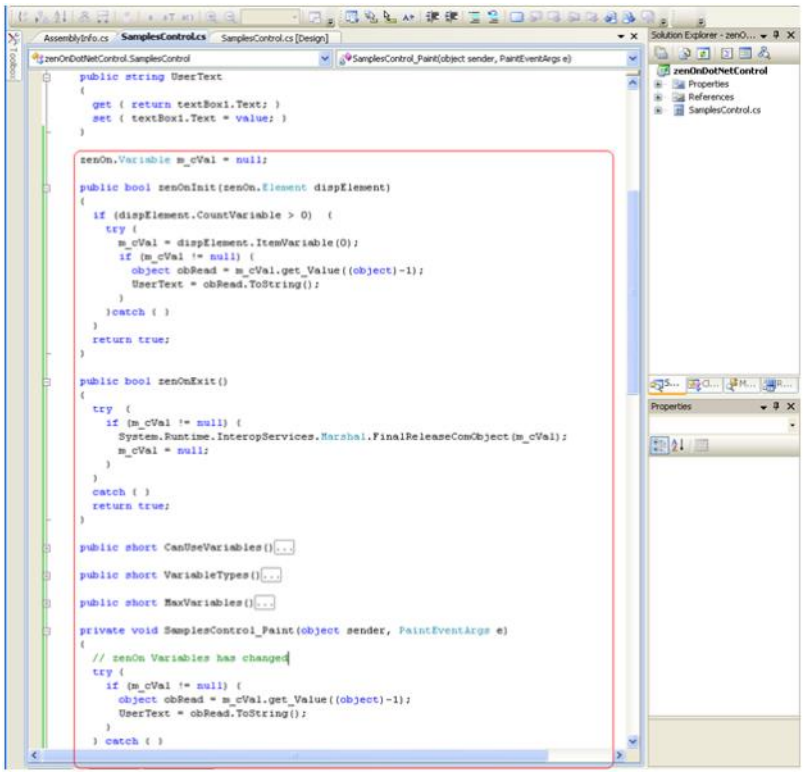
REFERENZ HINZUFÜGEN

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 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 44)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 45) 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-Element

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

> **Info**
>
> *Alle Marken- und Produktnamen in dieser Dokumentation sind Warenzeichen oder eingetragene Warenzeichen der jeweiligen Titelhalter.*

6.1 Grundlagen

XAML

XAML steht für **Extensible Application Markup Language** und wird [ˈzæ:məl] ausgesprochen. Die von Microsoft entwickelte XML basierte Beschreibungssprache definiert grafische Elemente, Animationen, Transformationen, Darstellungen von Farbverläufen, etc. in Silverlight- und WPF Oberflächen. Die Verwendung von XAML erlaubt es, Design und Programmierung strikt zu trennen. Der Designer bereitet beispielsweise die grafische Oberfläche vor und erstellt Basis-Animationen, die dann vom Entwickler/Projektant der die Anwendungslogik erstellt, aufgegriffen werden.

WPF

WPF steht für Windows Presentation Foundation und bezeichnet ein Grafik-Framework, welches Teil vom Microsoft .NET Framework ist.

- WPF stellt ein umfangreiches Modell für den Programmierer bereit.
- XAML beschreibt, basierend auf XML, als Auszeichnungssprache die Oberflächenhierarchie.
  Je nach Aufbau der XAML-Datei besteht die Möglichkeit, Eigenschaften, Ereignisse und Transformationen von WPF-Elementen mit Variablen und Funktionen von zenon zu verknüpfen.

- Das Framework vereint die verschiedenen Bereiche einer Präsentation wie Benutzerschnittstelle, Zeichnen, Grafiken, Audio, Video, Dokumente und Typographie.

Zur Ausführung in zenon wird die Microsoft .NET Framework Version 3.5 oder höher benötigt.

6.1.1 WPF in der Prozessvisualisierung

Für zenon eröffnet XAML individuelle grafische Gestaltungsmöglichkeiten. Anzeigenelemente und dynamische Elemente können unabhängig von der Projektierung grafisch angepasst werden. Zum Beispiel werden aufwändige Illustrationen erst von Designern erstellt und dann als XAML-Datei in zenon importiert sowie mit der gewünschten Logik verknüpft. Dafür bieten sich viele Anwendungsmöglichkeiten an, zum Beispiel:

DYNAMISCHE ELEMENTE IM ANALOG-LOOK

Grafiken müssen nicht mehr in zenon gezeichnet werden, sondern können direkt als XAML-Datei importiert werden. Das erlaubt den Einsatz von komplexen, aufwändig illustrierten Elementen in der Prozessvisualisierung. Spiegelungen, Schatten, 3D-Effekte etc. werden zum Beispiel als Grafik unterstützt. Die an die jeweilige Industrie-Umgebung angepassten Elemente ermöglichen intuitives Bedienen, analog den Bedienelementen an der Maschine.

AUFWÄNDIGE ILLUSTRATIONEN FÜR INTUITIVES BEDIENEN

Die Einbindung von XAML basierenden Anzeigeelementen wertet Projekte grafisch auf und ermöglicht sehr einfach die übersichtlichere Prozessdarstellung. Auf Usability optimierte Elemente vereinfachen die Bedienung. Die übersichtliche Darstellung von Daten erleichtert die Rezeption von komplexen Inhalten. Die flexible Anpassungsmöglichkeit der einzelnen Elemente erhöht den Bedienkomfort. So ist es dem Projektanten möglich, eigenständig Anzeigenwerte, Skalen und Einheiten zu bestimmen.

ÜBERSICHTLICHE DATENPRÄSENTATION UND ZUSAMMENFASSUNGEN

Gruppierte Anzeigenelemente ermöglichen die übersichtliche Darstellung der wichtigsten Prozessdaten, so dass der Anlagenbediener stets über aktuelle Prozessabläufe informiert ist. Grafische Auswertungen, Anzeigenwerte und Regler können in einem Element gruppiert werden und ermöglichen eine schnelle und unkomplizierte Steuerung.

INDUSTRIESPEZIFISCHE DARSTELLUNGEN

Elemente wie Thermometer, Skalen oder Bargrafen zählen zu den Basis-Elementen der Prozessvisualisierung. Durch XAML ist es möglich, diese grafisch an die jeweilige Industrie anzupassen. So finden Anlagenbediener die etablierten und gewohnten Elemente, die sie bereits von den Maschinen kennen, in der Prozessvisualisierung am Terminal wieder.

ANPASSUNG AN DAS CORPORATE DESIGN

Illustrationen können an die jeweiligen Stilvorgaben des Unternehmens angepasst werden, um ein durchgängiges Erscheinungsbild bis hin zum einzelnen Prozessbild zu erreichen. Als Vorlage können dazu zum Beispiel die Standard-Bedienelemente von zenon verwendet werden, die dann an Farbwelten, Hausschriften und Illustrationsstile des Corporate Designs angepasst werden.

6.1.2 Referenzierte Assemblies

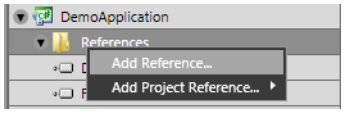
Mittels dem **WPF-Elementen** können nicht nur Standardobjekte (Rechtecke, Grafiken, etc.), bzw. Effekte (Farbverläufe, Animationen, etc.) dargestellt werden, sondern auch customisierte User Controls (mit Logik im Code Behind), die als Assemblies referenziert werden.

So kann zum Beispiel ein User Control erstellt werden welches wie ein Tacho aussehen und spezielle Eigenschaften und optische Effekte anbietet, etwa eine Eigenschaft „Value", die beim Setzen dazu führt, dass der Zeiger des Tachos sich bewegt und/oder der entsprechende Wert in einem Label anzeigt wird.

Der Workflow dazu:

- Das Aussehen eines User Controls wird mit Standard-Objekten gezeichnet, die von WPF angeboten werden.
- Die Eigenschaften und Interaktionen werden programmiert.
- Das Gesamtpaket wird kompiliert und liegt als .NET Assembly vor.

Diese Assembly kann für WPF-Projekte verwendete werden. Dafür muss es im WPF-Editor (z.B.: Microsoft Expression Blend) referenziert (verknüpft) werden. Wählen Sie dazu im zenon Datei-Auswahldialog die Assembly aus:



Ab diesem Zeitpunkt können die WPF User Controls der Assembly in der Toolbox unter **Custom UserControls** ausgewählt und im WPF-Projekt verwendet werden.

Siehe dazu auch im Kapitel: Leitfaden für Entwickler (auf Seite 97).

REFERENZIERTE ASSEMBLIES IN ZENON VERWENDEN

Um eine Assembly in zenon zu verwenden, muss dieses als Datei zur Verfügung gestellt werden. Sammeldateien vom Format **.cdwpf** verwalten diese eigenständig, es sind keine weiteren Konfigurationen nötig. Für Dateien vom Format .xaml müssen Assemblies zum Ordner `Dateien` hinzugefügt werden:

- klicken Sie im Projektbaum auf `Dateien`
- wählen Sie `Sonstige`
- wählen Sie im Kontextmenü **Datei hinzufügen...**
- der Dateiauswahldialog öffnet sich
- fügen Sie die gewünschte Assembly ein

Beim Anzeigen einer WPF-Datei im **WPF-Element** (Editor und Runtime) werden die Assemblies aus diesem Ordner geladen. Damit ist auch sicher gestellt, dass beim Übertragen der Runtime-Dateien mittels **Remote-Transport** alle referenzierten Assemblies auf dem Zielrechner vorhanden sind.

Eine Sammeldatei (**.cdwpf**) und eine gleichnamige XAML-Datei können parallel existieren. Alle Assemblies (*.dll) aus allen Sammeldateien und dem Ordner `Sonstige` werden in einen Arbeitsordner kopiert. Bei mehreren gleichnamigen Assemblies wird nur das mit der höchsten Dateiversion verwendet.

Achtung

Assemblies werden nach dem Laden erst beim Beenden der Applikation wieder entladen. Das bedeutet:

Wenn eine WPF-Datei mit einer referenzierten Assembly in zenon dargestellt wird, dann wird diese Assembly geladen und befindet sich bis zum Beenden von zenon im Speicher, selbst wenn das Bild wieder geschlossen wurde. Möchten Sie eine Assembly aus dem Ordner `Dateien/Sonstige` *entfernen, muss zuvor der Editor neu gestartet werden, damit die Assembly entladen wird.*

MEHRPROJEKTVERWALTUNG

In der Mehrprojektverwaltung muss in allen Projekten dieselbe Assembly verwendet werden. Wird in einem Projekt eine Assembly durch eine andere Version ersetzt, muss sie auch in allen anderen Projekten, die im Editor oder in der Runtime geladen sind, ersetzt werden.

6.1.3 Workflows

Die WPF/XAML-Technologie ermöglicht neue Workflows in der Prozessvisualisierung. Die Trennung von Design und Funktionalität sorgt für klare Aufgabenteilung zwischen Projektant und Designer; Designvorgaben lassen sich durch die Verwendung bereits existierender Designs, die vom Projektant nicht mehr modifiziert werden müssen, leicht erfüllen.

Am Workflow zur Erstellung von WPF-Elementen in zenon sind folgende Personen beteiligt:

- Designer
  - illustriert Elemente
  - sorgt für grafische Aufbereitung für MS Expression Design
- MS Expression Blend Operator
  - animiert Elemente
  - legt Variablen für die Animation der WPF-Elemente in zenon an, auf die der Projektant zugreifen kann
- Projektant
  - bindet Elemente in zenon ein
  - hinterlegt Logik und Funktionalität

Wir unterscheiden:

- Workflow mit Microsoft Expression Blend (auf Seite 84)
- Workflow mit Adobe Illustrator (auf Seite 84)

Workflow mit Microsoft Expression Blend

Bei Verwendung von Microsoft Expression Blend erfolgt die Erstellung eines WPF-Elements in vier Schritten:

1. Illustration des Elements in **MS Expression Blend** (auf Seite 85)
2. Element in **MS Expression Design** öffnen und als WPF exportieren
3. Animation in **MS Expression Blend** (auf Seite 85)
4. Integration in zenon (auf Seite 174)

Ein Beispiel für die Erstellung eines WPF-Elements mit Microsoft Expression Blend finden Sie im Kapitel Button als XAML-Datei mit Microsoft Expression Blend erstellen (auf Seite 85).

Workflow mit Adobe Illustrator

Aufbauend auf traditionelle Gestaltungsabläufe mit **Adobe Illustrator** bietet sich folgender Workflow an:

1. Illustration der Elemente in **Adobe Illustrator** (auf Seite 90)
2. Import der **.ai** Dateien und Aufbereitung in **MS Expression Design** (auf Seite 92)
3. WPF-Export aus **MS Expression Design** (auf Seite 92)
4. Animation in **MS Expression Blend** (auf Seite 93)
5. Integration in zenon (auf Seite 168)

Ein Beispiel zur Erstellung finden Sie im Kapitel Workflow mit Adobe Illustrator (auf Seite 89).

6.2 Leitfaden für Designer

Dieser Abschnitt informiert über die korrekte Erstellung von WPF Dateien im Microsoft Expression Blend und Adobe Illustrator. Die Tutorials zur Erstellung eines Button-Elements (auf Seite 85) und eines Bargraf-Elements (auf Seite 89) zeigen, wie in wenigen Schritten aus bereits existierenden Grafiken voll funktionsfähige WPF-Dateien für zenon erzeugt werden.

Folgende Werkzeuge werden dafür benützt:

- Adobe Illustrator CS3 (AI)
- Microsoft Expression Design 4 (ED)
- Microsoft Expression Blend 4 (EB)
- zenon

Info

Sollen referenzierte Objekte (Assemblies) in WPF verwendet werden, beachten Sie die Hinweise im Kapitel Referenzierte Objekte (auf Seite 81).

6.2.1 Workflow mit Microsoft Expression Blend

Mit Microsoft Expression Blend wird ein WPF-Element:

- illustriert
- über **MS Expression Design** in das WPF-Format umgewandelt
- animiert

Das folgende Beispiel zeigt die Illustration und Umwandlung eines Button-Elements in eine XAML-Datei.

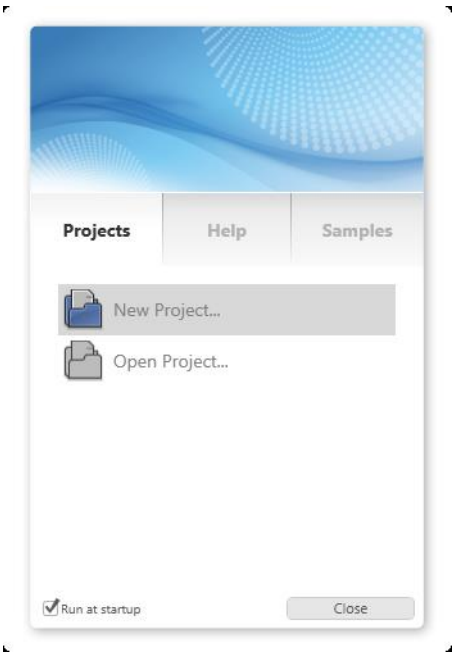
Hinweis: Eine Test-Version von „Microsoft Expression Blend" steht auf der Microsoft Web-Seite zum Download zur Verfügung.

Button als XAML-Datei mit Microsoft Expression Blend erstellen

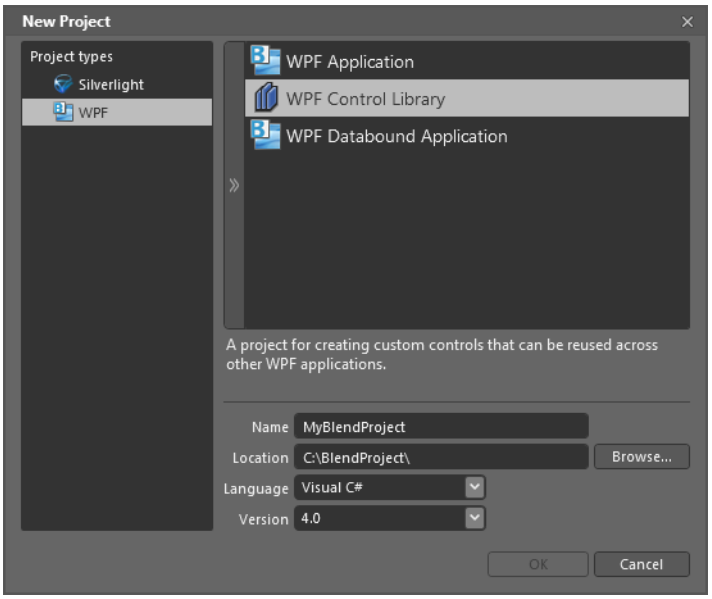
BUTTON ERSTELLEN

1. Starten Sie Expression Blend

2. wählen Sie die Option **New Projekt**



3. wählen Sie als Projekttyp `WPF`
4. vergeben Sie einen beliebigen Pfad und Namen (z. B. MyBlendProject)



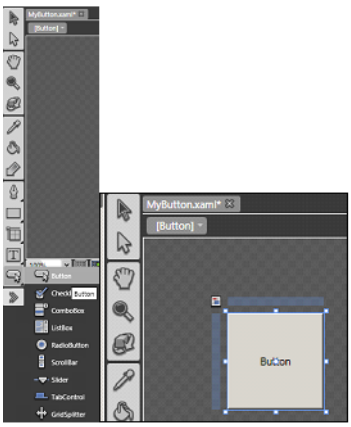
Die Einstellungen **Language** und **Version** können ignoriert werden, da keine Funktionalität programmiert werden soll.

5. Nachdem der Dialog mit **OK** bestätigt wurde, erzeugt Microsoft Blend ein neues Projekt mit den gewählte Einstellungen. Expression Blend fügt eine leere XAML-Datei ein, die bereits eine Klassen-Referenz beinhaltet.
6. Löschen Sie, die an die XAML-Datei dazugehörende, CS-Datei über das Kontextmenü.

7. Benennen Sie die XAML-Datei **MainControl.xaml** auf **MyButton.xaml** um.
8. Die Development-Größe der Datei wird standardmäßig auf 640 mal 480 Pixel gestellt und muss noch geändert werden:
   a) wechseln Sie in die Ansicht **XAML**
   b) korrigieren Sie die Größe auf 100 mal 100 Pixel
   c) löschen Sie die Klassenreferenz x:Class="MyBlendProject.MyButton"

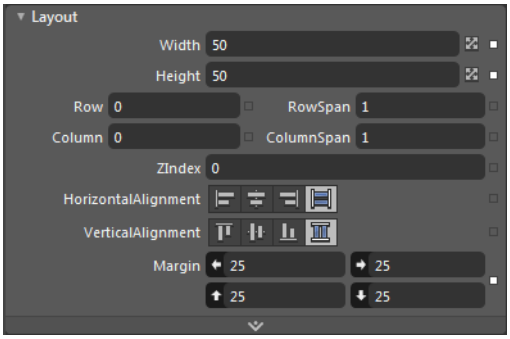
```
MyButton.xaml
1  <UserControl
2      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6      mc:Ignorable="d"
7      x:Name="UserControl"
8      d:DesignWidth="100" d:DesignHeight="100">
9
10     <Grid x:Name="LayoutRoot" />
11
12 </UserControl>
```

9. wechseln Sie in die Ansicht **Design**

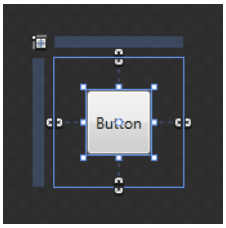


10. fügen Sie über die Symbolleiste einen Button ein
11. definieren Sie die Eigenschaften
    - Width: 50
    - Height: 50

- Margins: 25



Damit befindet sich der Button jetzt in der Mitte des Controls.



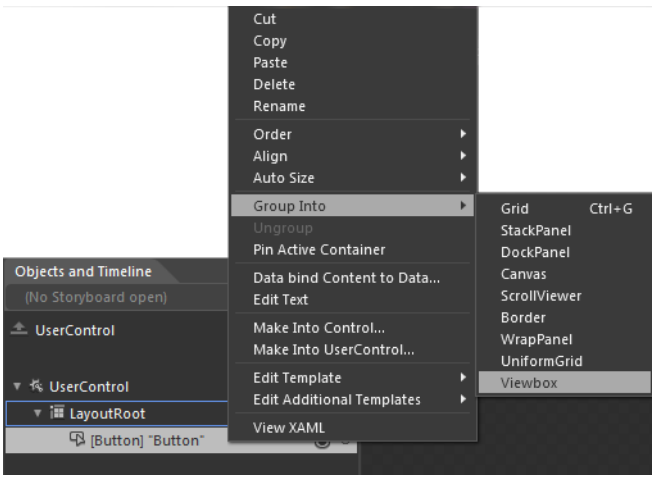
12. Speichern Sie die Änderungen und öffnen Sie die Datei zur Kontrolle im Internet Explorer. Sie sehen, dass der Button in der Größe von 50 x 50 Pixel dargestellt wird.

BUTTON SKALIERBAR MACHEN

Wenn Sie diese Datei in diesem Zustand in zenon einbinden, wird der Button dort immer exakt die Größe von 50 x 50 Pixel haben. Da der Button skalierbar eingesetzt werden soll, wechseln Sie wieder zu Expression Blend:

1. selektieren Sie den Button in der Baumansicht
2. wählen Sie im Kontextmenü des Buttons Group Into->Viewbox
3. der Button wird in einer **Viewbox** eingefügt
4. definieren Sie die Eigenschaften der Viewbox
   - Width: Auto
   - Height: Auto

5. speichern Sie die Datei



6. Wenn Sie die Datei jetzt im Internet Explorer öffnen, wird der Button bei Größenänderungen des IE-Fensters automatisch skaliert. Diese Datei würde sich jetzt auch in zenon automatisch an Größenänderungen des **WPF-Elements** anpassen.

NAMEN ÄNDERN

Bevor Sie die Datei in zenon einbinden können, müssen Sie dem **WPF-Element** einen Name geben. Standardmäßig sind **WPF-Elemente** in Expression Blend nicht benannt, und werden mit eckige Klammer und dessen Typ gekennzeichnet. Die Zuordnung von zenon-Content auf WPF-Content geschieht über den Namen der **WPF-Elemente**:

- ändern Sie in der Baumansicht den Namen
  - des Buttons auf **MyButton**
  - der ViewBox auf **MyViewBox**

Dieser Button kann jetzt als XAML-Datei in zenon eingebunden (auf Seite 174) werden.

6.2.2 Workflow mit Adobe Illustrator

Beim Einsatz von **Adobe Illustrator** wird ein WPF-Element:

- in **Adobe Illustrator** illustriert
- in **MS Expression Design** in ein WPF umgewandelt
- in **MS Expression Blend** animiert

Das folgende Beispiel zeigt die Illustration und Umwandlung eines Bargraf-Elements in eine XAML-Datei.

Illustration Bargraf

Ein Bargraf wird in Adobe Illustrator erstellt.

1. AI: Ausgangselement Bargraf

   Illustriert in Adobe Illustrator CS3.

2. AI: Pfadansicht Bargraf in Adobe Illustrator

   - alle Effekte müssen umgewandelt sein (**Objekt -> Aussehen umwandeln**)
   - alle Linien werden in Pfade umgewandelt (**Objekt -> Pfad -> Konturlinie**)
   - keine Filter wie Schlagschatten, Weichzeichner etc. verwenden

HINWEISE ZUR KOMPATIBILTÄT

Illustrationen, die mit Adobe Illustrator erstellt werden, eignen sich grundsätzlich für den WPF-Export. Allerdings können nicht alle Illustrator-Effekte in Expression Design/Blend entsprechend werden. Beachten Sie:

Effekt	Beschreibung
Schnittmasken	In Adobe Illustrator erzeugte Schnittmasken werden von Expression Design nicht korrekt interpretiert. Meist werden sie in Blend als schwarze Farbflächen dargestellt. Wir empfehlen, die Illustrationen ohne Schnittmasken anzulegen.
Filter und Effekte	Nicht jeder Adobe Illustrator Filter wird in Expression Design entsprechend übernommen: So funktionieren z. B. Weichzeichnungsfilter, Schlagschatten sowie Eckeneffekte aus Illustrator in Expression Design nicht. Lösung: ▸ Über den Befehl **Objekt -> Aussehen Umwandeln** in Adobe Illustrator lassen sich die meisten Effekte so umwandeln, dass sie von Expression Design korrekt gelesen werden können. ▸ Eckeneffekte aus Adobe Illustrator werden von MS Design korrekt interpretiert, wenn sie in AI in Pfade umgewandelt werden.
Textfelder	Um Textfelder mit Code verknüpfen zu können, müssen diese in Expression Blend gesondert angelegt werden. „**Labels**" werden für dynamische Texte benötigt, für statische Informationen genügen einfache „**Textfelder**". Die Möglichkeit, Textlabels zu erstellen, wird in MS Design nicht angeboten. Diese müssen direkt in MS Blend erstellt werden.
Transparenzen und Gruppentransparenzen	Zu Schwierigkeiten kann es bei der korrekten Interpretation von Transparenz-Einstellungen, insbesondere von Gruppentransparenz-Einstellungen, in Adobe Illustrator kommen. MS Expression Blend sowie MS Expression Design bieten aber die Möglichkeit, Transparenz-Einstellungen neu anzulegen.
Ebenen multiplizieren	Diese Ebenen-Einstellungen in Adobe Illustrator werden von MS Expression Blend nicht immer richtig dargestellt. Es gibt aber die Möglichkeit, „**Ebene multiplizieren**" direkt in Expression Design einzustellen.
Zeigerinstrumente und Standardpositionen	Um die Grafiken optimal für die Animation vorzubereiten, müssen Zeiger und Regler immer auf der Ausgangsposition, meist `0` oder `12:00 Uhr` stehen. So werden die Positions-Parameter für Rotationen etc. gleich richtig in Blend angegeben und eine Animation kann ohne Umrechnung von Positionsangaben erfolgen.

WPF-Export

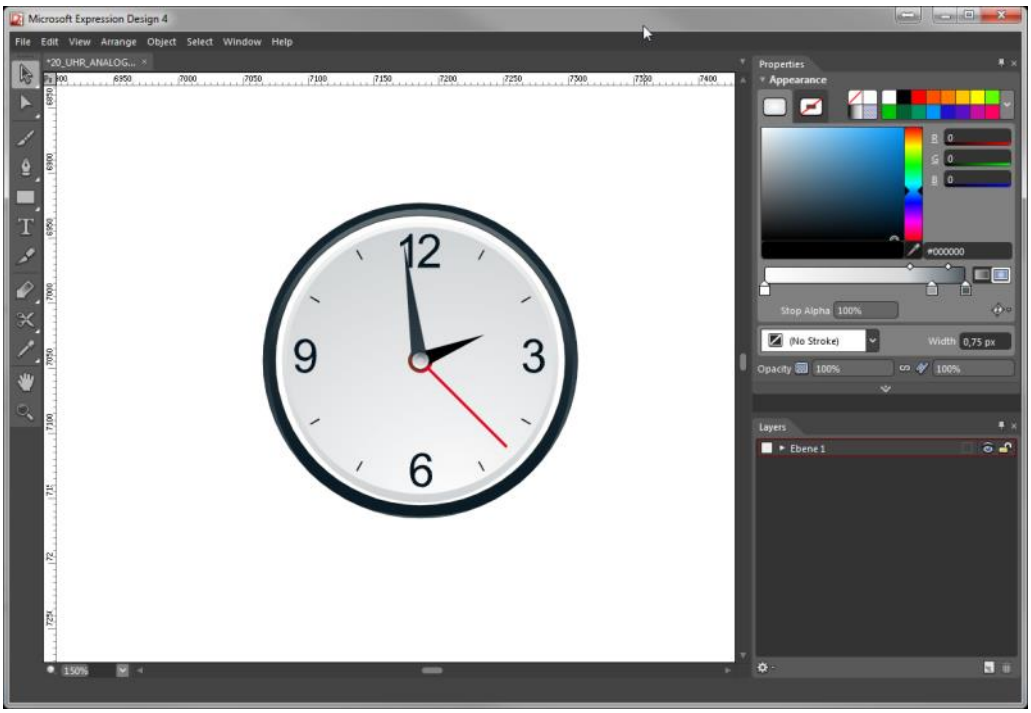
Für die Animation in Microsoft Expression Blend werden WPF-Dateien benötigt. Wir empfehlen Microsoft Expression Design für diesen Export, da es gute Ergebnisse liefert und die meisten Illustrator Effekte korrekt interpretiert werden.

Hinweis: Im Internet wird ein gratis Plug-in für den direkten Export von WPF-Dateien aus Adobe Illustrator angeboten. Dieses Plug-in bietet einen schnellen, unkomplizierten Export aus Illustrator, ist für die aktuelle Anwendung jedoch weniger gut geeignet, da es zu grafischen Verlusten kommt. Auch Farbabweichungen vom ursprünglichen Dokument sind möglich.

Dateien im Format **.ai** können regulär in Expression Design importiert werden, die Pfade bleiben dabei erhalten.
Achtung: Einige geläufige Illustrator Effekte können jedoch von Expression Design nicht richtig dargestellt werden (siehe Kapitel Illustration (auf Seite 90)).

In 5 Schritten exportieren wir das bereits erstellte Bargraf-Element:

1. ED: Import
   - importieren Sie die aufbereitete Illustrator-Datei (auf Seite 90) in **Microsoft Expression Design** über Datei -> Importieren
2. ED: Optimierung



   - wird die Ausgangsdatei in MS Expression Design nicht richtig angezeigt, kann sie hier noch nachbearbeitet und optimiert werden

3. ED: Auswahl

- markieren Sie mit dem **Direktauswahl**-Pfeil in MS Expression Design das Element für den WPF-Export, in diesem Fall die gesamte Uhr

4. ED: Export starten

- starten Sie den Export über Datei -> Exportieren
- der Dialog zur Konfiguration der Exporteinstellungen öffnet sich

5. ED: Exporteinstellungen

- Nehmen Sie folgende Exporteinstellungen vor:

a) **Format**: `XAML Silverlight 4 / WPF Canvas`

Objekte immer benennen: Mit Häkchen aktivieren

Platzieren Sie gruppierte Objekte in einem XAML Layoutcontainer: Mit Häkchen aktivieren

b) **Text:** `Bearbeitbarer Textblock`

c) **Linieneffekte:** `Alle rastern`

Die exportierte Datei hat die Dateiendung **.xaml.** Sie wird im nächsten Schritt in MS Expression Blend aufbereitet und animiert (auf Seite 93).

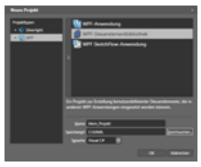
Animation in Blend

Mit MS Expression Blend werden:

- die statischen XAML-Dateien aus MS Expression Design animiert
- Variablen für die Steuerung der Effekte, die von zenon angesprochen werden können, angelegt

In 13 Schritten gelangen wir vom statischen XAML zum animierten Element, das in zenon eingebunden werden kann:

1. EB:Projekt erstellen



   a) öffnen Sie Microsoft Expression Blend

   b) erstellen Sie ein neues Projekt

   c) wählen Sie als **Projekttyp** WPF -> WPF Control Library aus

   d) vergeben Sie einen Namen (in unserem Tutorial: **Mein_Projekt**)

   e) wählen Sie einen Speicherort

   f) wählen Sie eine Sprache (in unserem Tutorial: C#)

   g) wählen Sie die Framework Version 3.5

2. EB: MainControl.xaml.cs löschen

   a) navigieren Sie zu **MainControl.xaml.cs**

   b) löschen Sie diese Datei über den Befehl **Löschen** im Kontextmenü

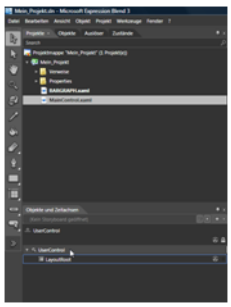
3. EB: exportierte XAML-Datei öffnen

   a) öffnen Sie das Kontextmenü für **Mein_Projekt** (rechte Maustaste)

   b) wählen Sie **Vorhandenes Element hinzufügen...** aus

   c) wählen Sie die aus Microsoft Expression Design exportierte XAML-Datei, um diese in Microsoft Expression Blend zu öffnen

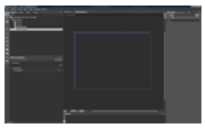
4. EB: MainControl.xaml öffnen



a) öffnen Sie die automatisch erstellte **MainControl.xaml**

b) navigieren Sie im Bereich **Objekte und Zeitachsen** zum Eintrag **UserControl**

5. EB: XAML-Code anpassen



a) klicken Sie mit der rechten Maustaste auf **UserControl**

b) wählen Sie im Kontextmenü **XAML anzeigen**

c) löschen Sie m XAML-Code die Zeilen 7 und 9:

```
x:Class="Mein_Projekt.MainControl"
d:DesignWidth="640" d:DesignHeight="480"
```

6. EB: XAML-Code überprüfen

- Nun sollte der XAML Code so aussehen:

```
<UserControl
    xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
    xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
    xmlns:d=http://schemas.microsoft.com/expression/blend/2008
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"


    mc:Ignorable="d"
    x:Name="UserControl">


    <Grid x:Name="LayoutRoot"/>
</UserControl>
```

7. EB: Elemente kopieren

a) öffnen Sie die von Expression Design importiertes XAML-Datei

b) markieren Sie alle Elemente

c) wählen Sie im Kontextmenü **Kopieren**

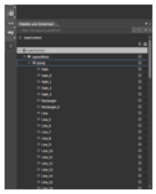
d) wechseln Sie zurück zur automatisch erstellten XAML-Datei

8. EB: Elemente einfügen

a) klicken Sie mit der rechten Maustaste auf **Layout Root**

b) wählen Sie **Einfügen**

9. EB: Layouttyp anpassen

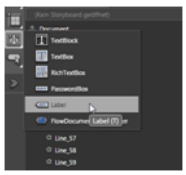


a) klicken Sie mit der rechten Maustaste auf LayoutRoot -> Layouttyp ändern -> Viewbox

b) nun sollte die Struktur so aussehen: **UserControl -> LayoutRoot -> Grid -> Elemente**

c) vergeben sie einen Namen für **LayoutRoot** und **Grid** durch Doppelklick auf den Namen

10. EB: Texte und Werte



- dynamische und statische Texte werden mit Textfeldern gezeichnet
- Werte (Zahlen) werden mit **Labels** ausgegeben

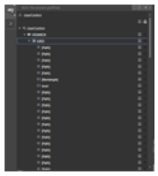
11. EB: Labels einfügen

- Labels ersetzen Zahlen, die später mittels Int-Variablen verknüpft werden sollen (muss für alle Zahlen-Elementen durchgeführt werden)

12. EB: Eigenschaft setzen

- um 100% anzuzeigen, setzen Sie die Eigenschaft **MaxHeight** des Bargrafanzeige-Elements auf `341` (die maximale Höhe des Indikator-Elements entspricht `340`)

13. EB: für Verwendung in zenon vorbereiten



a) löschen Sie alle Namensbezeichnungen (Namen dürfen nur für Elemente vergeben werden, die über zenon angesprochen werden sollen)

b) speichern Sie die XAML-Datei mit einem beliebigen Namen ab

c) binden Sie die XAML-Datei in zenon ein (auf Seite 168)

Tipp zur Kontrolle: Wenn die XAML-Datei im Microsoft Internet Explorer einwandfrei dargestellt wird und sich auch der Fenstergröße des Internet Explorers anpasst, wird sie auch in zenon korrekt verwendet.

6.3 Leitfaden für Entwickler

Dieser Abschnitt behandelt die Erstellung eines einfachen WPF User Controls mit Code Behind Funktionalität mittels Microsoft Visual Studio und das Debuggen dieses User Controls zur Runtime.

Folgende Werkzeuge werden dafür benützt:

- Microsoft Visual Studio 2015
- zenon

> **Info**
>
> *Es wird eine Microsoft Visual Studio Version ab 2012 empfohlen, aufgrund des besseren integrierten XAML Designers.*

6.3.1 Erstellung eines einfachen WPF User Controls mit Code Behind-Funktionalität

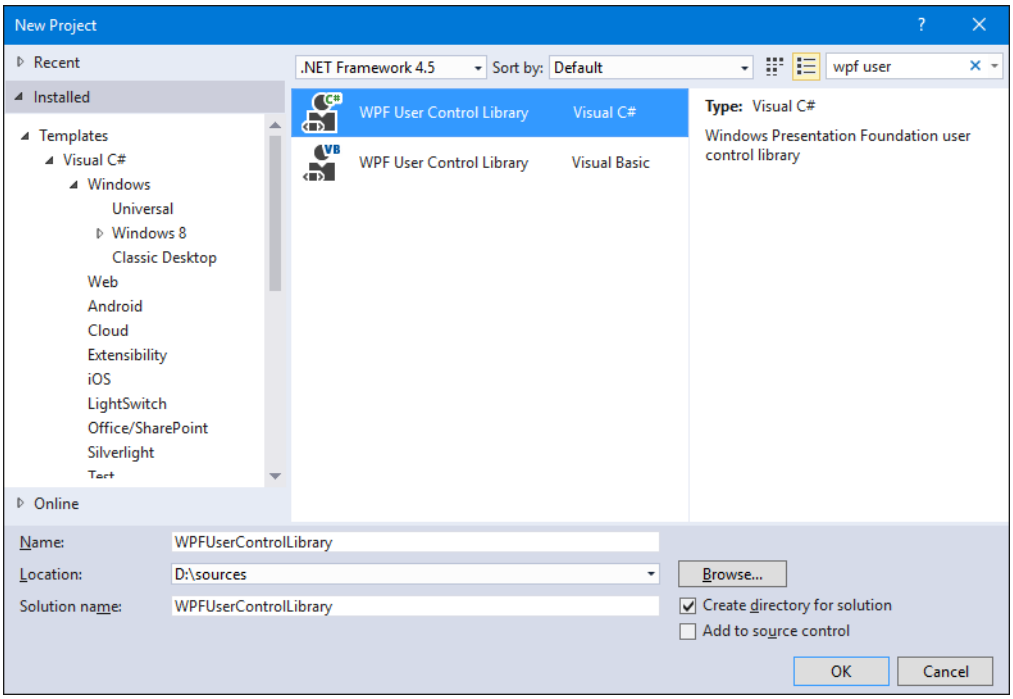
In diesem Kapitel wird die Erstellung, sowie die Einbindung eines einfachen WPF User Controls beschrieben. Da hier nur der grundlegende Mechanismus/Ablauf für die Integration in zenon beschrieben wird , beschränkt sich die Funktionalität des User Controls auf die Addition von zwei Werten. Auf erweitertetes Fehlerhandling sowie explizite Vollständigkeit wurde zur Beibehaltung der Einfachheit dieses Beispiels bewusst verzichtet.

WPF USER CONTROL ERSTELLEN

1. Erstellen Sie in Visual Studio eine neue **Solution** und darin ein **WPF User Control Library** Projekt.

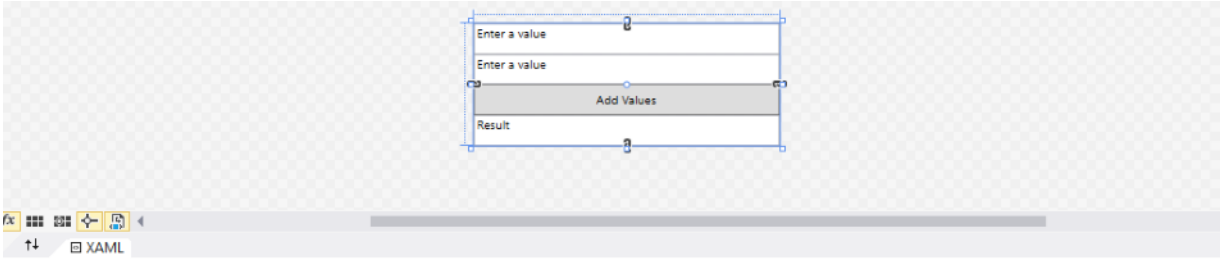
   In diesem Beispiel wurde die .NET Framework Version 4 gewählt. Es kann auch eine andere Version gewählt werden, die am Zielsystem, auf dem die Runtime später gestartet werden soll, installiert sein muss.

   Info: Erscheint die entsprechende Projektvorlage nicht in der Liste der verfügbaren Vorlagen, kann diese über die Suche (Feld rechts oben im Dialog) hinzugefügt werden.



   In unserem Beispiel erhält das Projekt den Namen **WPFUserControlLibrary**.

2. Erstellen Sie 3 Textboxen sowie einen Button in der Datei UserControl1.xaml:




```
<UserControl x:Class="WPFUserControlLibrary.UserControl1"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:WPFUserControlLibrary"
             mc:Ignorable="d"
             d:DesignHeight="120" d:DesignWidth="300">
    <Grid Height="120" Width="Auto">
    <StackPanel>
      <TextBox x:Name="textBoxA" Height="30" TextWrapping="Wrap" Text="Enter a value" HorizontalAlignment="Stretch"/>
      <TextBox x:Name="textBoxB" Height="30" TextWrapping="Wrap" Text="Enter a value" HorizontalAlignment="Stretch"/>
      <Button x:Name="buttonAdd" Height="30" Content="Add Values" HorizontalAlignment="Stretch" VerticalAlignment="Top" Click="buttonAdd_Click"/>
      <TextBox x:Name="textBoxC" Height="30" TextWrapping="Wrap" Text="Result" HorizontalAlignment="Stretch" IsReadOnly="True"/>
    </StackPanel>
  </Grid>
</UserControl>
```

3. Fügen Sie folgenden Code im Click-Event des Buttons ein:

```
private void buttonAdd_Click(object sender, RoutedEventArgs e)
{
  try
  {
    textBoxC.Text = (Convert.ToInt32(textBoxA.Text) + Convert.ToInt32(textBoxB.Text)).ToString();
  }
  catch (Exception ex)
  {
    textBoxC.Text =  "Error adding values: " + ex.Message;
  }
}
```

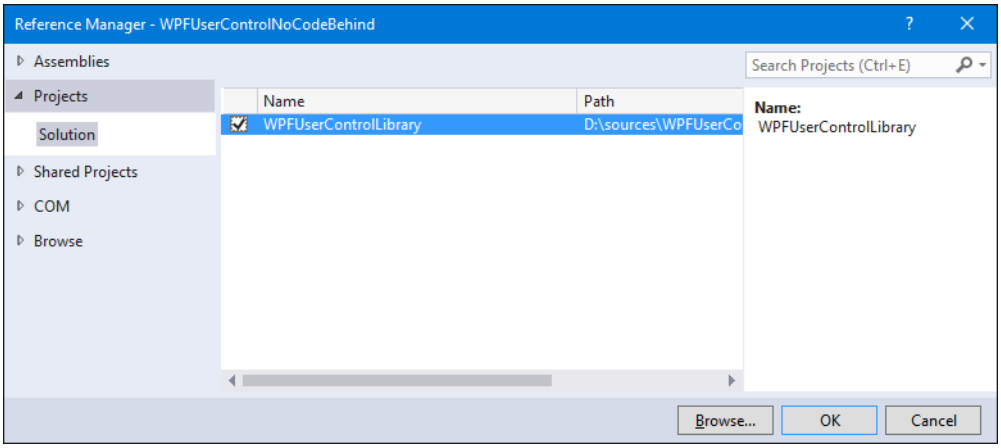
Sie verfügen jetzt über das User Control, dass die benötigte Funktionalität zur Verfügung stellt. Da zenon allerdings nur XAML-Dateien darstellen kann, die nicht zu einer Code-Behind Datei verlinken, wird eine zusätzliche XAML-Datei benötigt, die die eben gebaute Bibliothek (Assembly) referenziert.

ERSTELLUNG DER XAML-DATEI (OHNE CODE-BEHIND) FÜR ZENON

Zur Erstellung der in zenon benötigten XAML-Datei gehen Sie wie folgt vor.

1. Erstellen Sie ein weiteres Projekt, wieder als **WPF User Control Library**
2. In unserem Beispiel wurde es **WPFUserControlNoCodeBehind** benannt.

3. Fügen Sie in diesem neuen Projekt eine Referenz auf das eben gebaute Projekt ein.



4. Die XAML-Datei (**UserControl1.xaml**) sieht folgendermaßen aus:

```
<UserControl x:Class="WPFUserControlNoCodeBehind.UserControl1"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:WPFUserControlNoCodeBehind"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
  <Grid>

  </Grid>
</UserControl>
```

5. Da alle nötigen Inhalte, in der zuvor erstellten DLL enthalten sind und keine Code-Behind Datei verwendet werden kann, löschen Sie die folgenden Zeilen:

   x:Class="`WPFUserControlNoCodeBehind.UserControl1`"

   xmlns:local="`clr-namespace:WPFUserControlNoCodeBehind`"

6. Löschen Sie zusätzlich (für die Größeneinstellung des Designers) folgende Zeilen:

   mc:Ignorable="`d`"

   d:DesignHeight="`300`" d:DesignWidth="`300`"

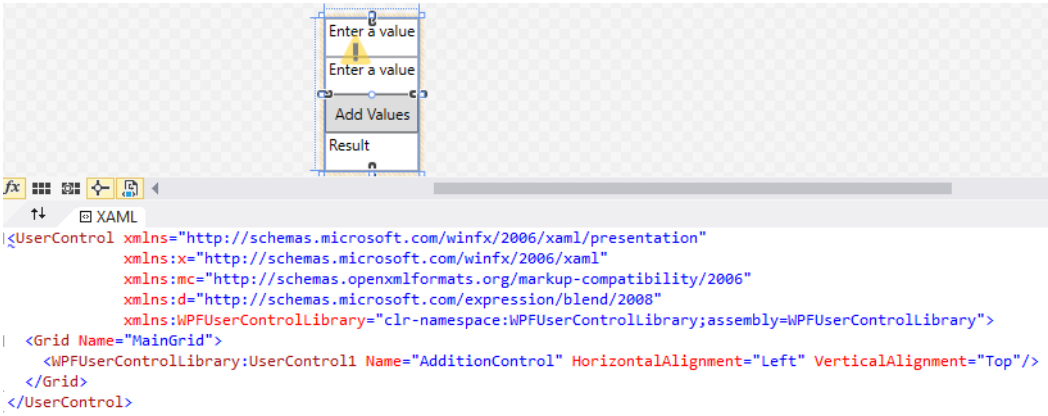
7. Löschen Sie die Code-Behind Datei (**UserControl1.xaml.cs**) in diesem Projekt.

8. Ziehen Sie das zuvor erstellte UserControl (aus dem Projekt **WPFUserControlLibrary**) über die Toolbox in den XAML-Designer.

9. Vergeben Sie einen Namen für den Grid, sowie das UserControl.

   Achtung: Wird hier kein Name vergeben, so erscheinen diese Elemente nicht im Verknüpfungsdialog im zenon Editor und können somit nicht dynamisiert werden.

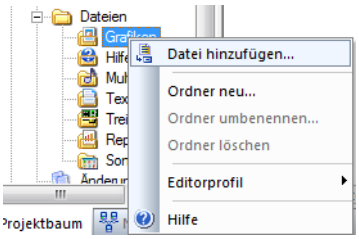
10. Die XAML-Datei sollte nun wie folgt aussehen:



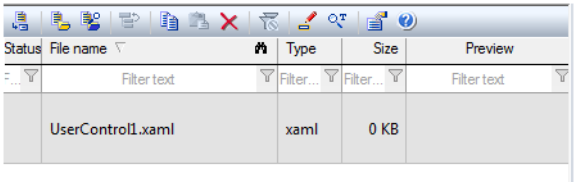
Im nächsten Schritt wird erklärt, wie die DLL sowie die XAML-Datei in zenon eingebunden werden.

SCHRITTE IN ZENON

1. Öffnen Sie den zenon Editor.
2. Navigieren Sie zum Knoten `Dateien -> Grafiken`.
3. Wählen Sie im Kontextmenü **Datei hinzufügen...**.



4. Wählen Sie die XAML-Datei am Speicherort (**UserControl1.xaml** aus dem Projekt **WPFUserControlNoCodeBehind**) aus und fügen Sie diese ein:

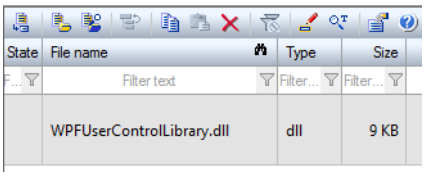


5. Fügen Sie die DLL mit der Funktionalität für die XAML-Datei hinzu.

   Dazu:

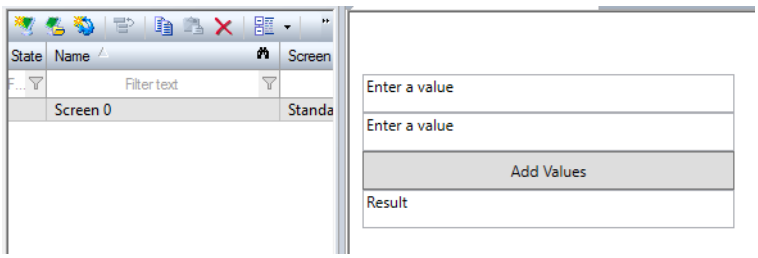
   a) Wählen Sie im Kontextmenü des Knotens Datei -> Sonstige **Datei hinzufügen...**.

b) Wählen Sie die Datei **WPFUserControlLibrary.dll** (aus dem Ausgabepfad) des ersten Projekts (**WPFUserControlLibrary**) aus.

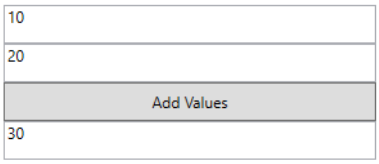


6. Legen Sie ein zenon Bild an.
7. Fügen Sie ein WPF-Element ein und wählen Sie die zuvor eingebundene XAML-Datei aus.

   Sie sollten jetzt im zenon Editor folgendes sehen:



8. Starten Sie die zenon Runtime um auch dort das Control zu testen.



Info

Die XAML-Datei und referenzierte Assemblies können auch zusammengefasst als *.cdwpf Datei gespeichert werden. Somit muss im Editor nur eine Datei (unter `Dateien -> Grafiken`) importiert werden. Weitere Informationen dazu sind im Kapitel CDWPF-Dateien (Sammeldateien) (auf Seite 121) zu finden.

Tipp: Während der Entwicklung eines WPF User Controls ist es meistens praktikabler, die XAML-Datei und die referenzierte(n) DLL(s) separat einzufügen. Dies erleichtert den Austausch der DLL und das Debuggen. Weitere Informationen zum Thema Debuggen sind im Kapitel Debuggen des WPF User Controls zur Runtime (auf Seite 103) zu finden.

Achtung

Assemblies werden nach dem Laden erst beim Beenden der Applikation wieder entladen. Das bedeutet:

Wenn eine WPF-Datei mit einer referenzierten Assembly in zenon dargestellt wird, dann wird diese Assembly geladen und befindet sich bis zum Beenden von zenon im Speicher, selbst wenn das Bild wieder geschlossen wurde. Möchten Sie eine Assembly aus dem Ordner `Dateien/Sonstige` *entfernen, muss zuvor der Editor neu gestartet werden, damit die Assembly entladen wird.*

Weitere Beispiele sind im Kapitel Beispiele: Integration von WPF in zenon (auf Seite 168) zu finden.

6.3.2 Debuggen des WPF User Controls zur Runtime

Um das WPF User Control zur Runtime zu debuggen, gehen Sie wie folgt vor.

In diesem Beispiel wird das, im Kapitel Erstellung eines einfachen WPF User Controls mit Code Behind-Funktionalität (auf Seite 97) beschriebene Control verwendet.

DEBUGGEN ÜBER ATTACH TO PROCESS

1. Stellen Sie sicher, dass die zenon Runtime gestartet ist und ein Bild mit dem WPF User Control geöffnet ist.

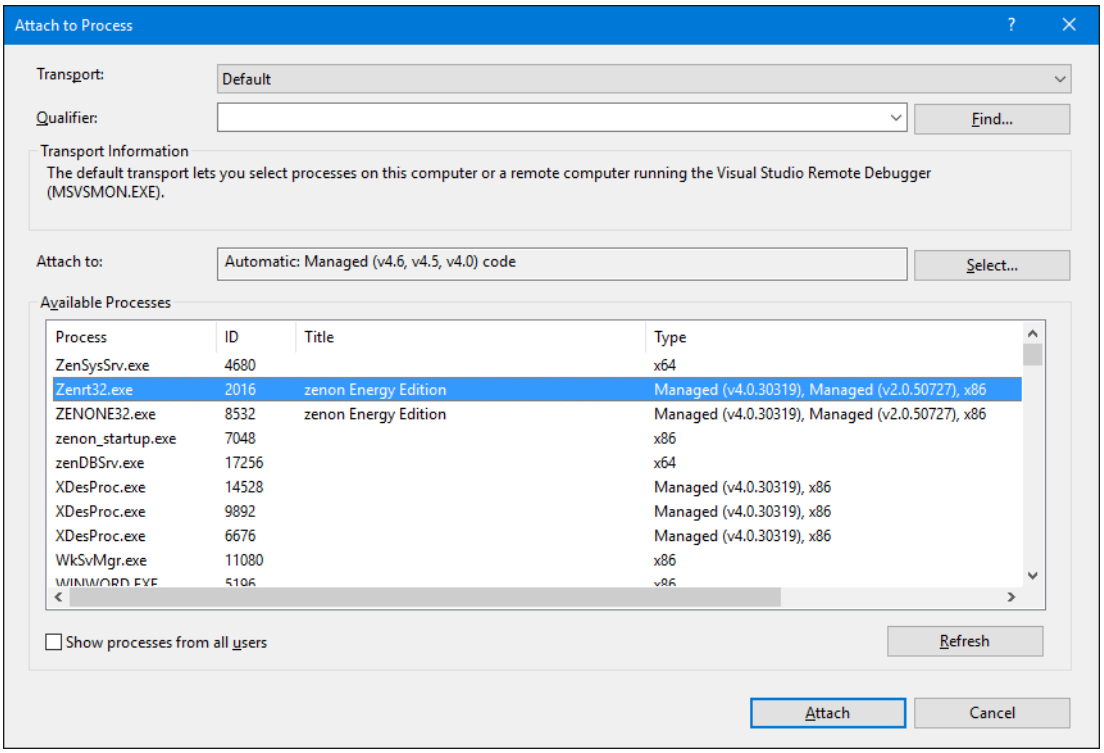
   Stellen Sie weiters sicher, dass die gerade verwendete DLL dem aktuellen Build (Version) des User Control Projekts (**WPFUserControlLibrary**) entspricht.

2. Setzen Sie in Visual Studio Projekt einen Breakpoint im Klick-Event des Buttons

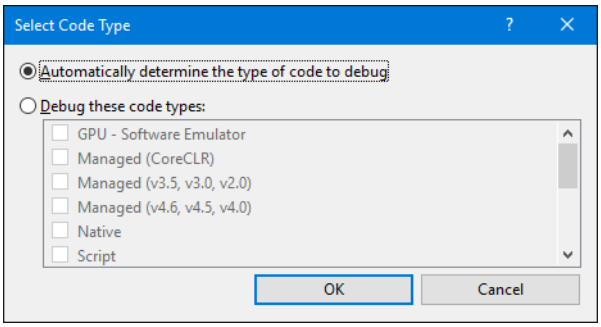
```
28  private void buttonAdd_Click(object sender, RoutedEventArgs e)
29  {
30    try
31    {
32      textBoxC.Text = (Convert.ToInt32(textBoxA.Text) + Convert.ToInt32(textBoxB.Text)).ToString();
33    }
34    catch (Exception ex)
35    {
36      textBoxC.Text =  "Error adding values: " + ex.Message;
37    }
38  }
```

3. Wählen Sie in Visual Studio unter **Debug** den Menüpunkt **Attach to Process**.

4. Wählen Sie den zenon Runtime Prozess

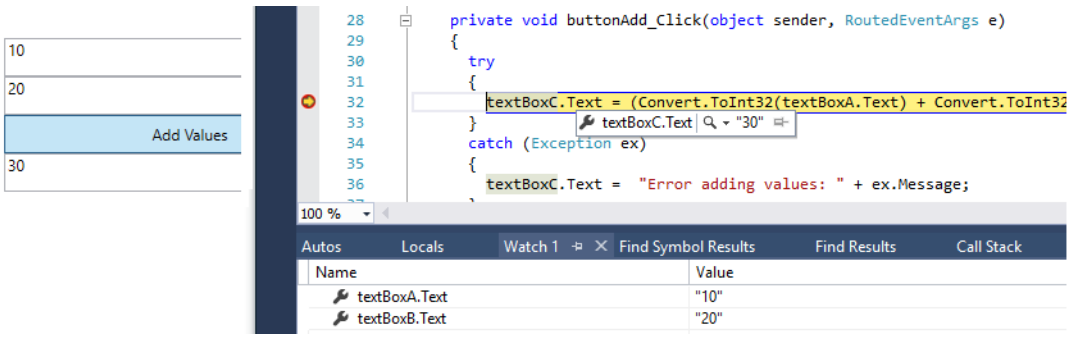


5. Wählen Sie unter **Attach to** entweder **Automatisch** oder die entsprechende .NET Framework Version (**v4.x** in diesem Fall)



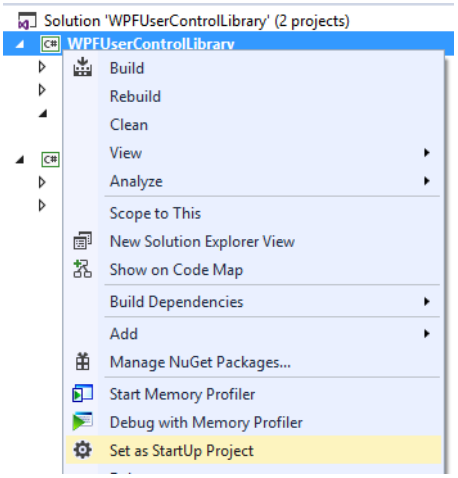
6. Klicken Sie auf **Attach**

7. Triggern Sie nun den Breakpoint, indem Sie in der zenon Runtime Werte im WPF Control eingeben und auf den Button klicken



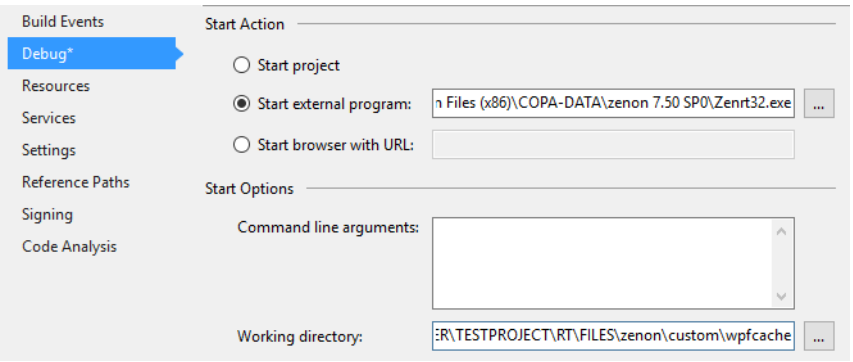
DEBUGGEN ÜBER START EXTERNAL PROGRAM

1. Stellen Sie sicher, dass die zenon Runtime beendet ist.
2. Stellen Sie sicher, dass im zenon Editor das Projekt welches das WPF User Control beinhaltet als Startprojekt gesetzt ist.
3. Stellen Sie sicher, dass das User Control Projekt (**WPFUserControlLibrary**) in Visual Studio als Startprojekt gesetzt ist.

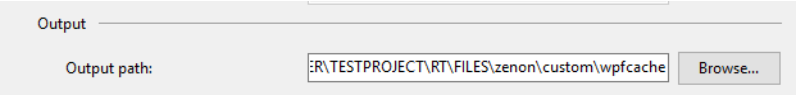


4. In den Projekteigenschaften des Visual Studio Projekts wählen Sie unter **Debug**, bei **Start action**: **Start external program**
5. Für **Start external program** wählen Sie den Pfad der zenon Runtime Anwendung.
6. Wählen Sie unter Working Directory den **\wpfcache** Ordner der Runtimedateien (**...\PROJECTNAME\RT\FILES\zenon\custom\wpfcache**)

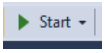
Tipp: Drücken Sie bei selektiertem Projekt im zenon Editor die Tastenkombination **STRG+ALT+R** um direkt in das Rootverzeichnis der Runtimedateien zu springen.



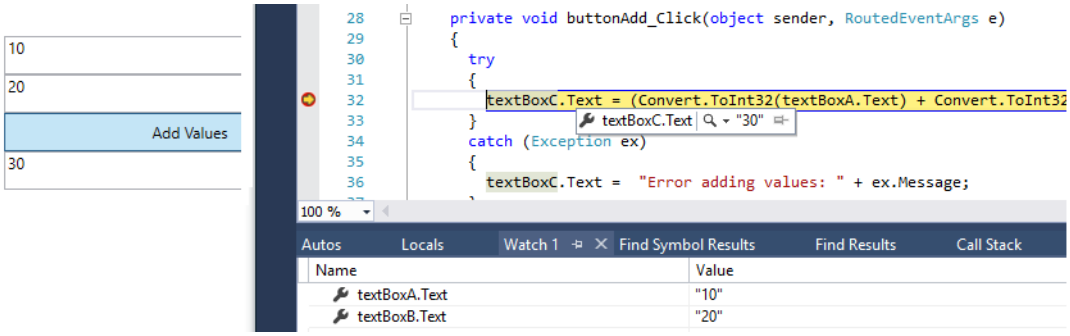
7. In den Projekteigenschaften, geben Sie unter **Build** als **Output path** ebenfalls den **\wpfache** Ordner der Runtimedateien an



8. Bauen Sie das Projekt in Visual Studio
9. Starten Sie in Visual Studio über **Start** das Debugging



10. Die zenon Runtime wird nun automatisch gestartet.
11. Triggern Sie den Breakpoint, indem Sie in der zenon Runtime Werte im WPF Control eingeben und auf den Button klicken



Info

Beim Starten der zenon Runtime werden die in WPF User Controls referenzierten Assemblies (DLLs) aus dem Ordner **\FILES\zenon\custom\additional**, bzw. die Assemblies aus **CDWPF** Dateien in den Ordner **\FILES\zenon\custom\wpfcache** kopiert. Ist die Dateiversion der im **\wpfache** Ordner vorhandenen DLL gleich oder höher der Version der "Originaldatei" wird diese nicht ersetzt!

Für das Debuggen ist es somit ausreichend, nur die Datei direkt im Ordner **\wpfache** zu ersetzen.

Für die Auslieferung muss sichergestellt sein, dass die aktuelle Version der DLL im **\additional** Ordner bzw. der **CDWPF** Datei vorhanden ist!

Achtung: Wenn nur die DLL im **\additional** Ordner bzw. im **CDWPF** aktualisiert wird, die Versionsnummer aber nicht erhöht wird, muss die DLL im **\wpcache** Ordner zuvor manuell gelöscht werden, da sie sonst nicht aktualisiert wird (aufgrund des oben beschriebenen Mechanismus).

Achtung

Assemblies werden nach dem Laden erst beim Beenden der Applikation wieder entladen. Das bedeutet:

Wenn eine WPF-Datei mit einer referenzierten Assembly in zenon dargestellt wird, dann wird diese Assembly geladen und befindet sich bis zum Beenden von zenon im Speicher, selbst wenn das Bild wieder geschlossen wurde. Möchten Sie eine Assembly aus dem Ordner `Dateien/Sonstige` *entfernen, muss zuvor der Editor neu gestartet werden, damit die Assembly entladen wird.*

6.3.3 Datenaustausch zwischen zenon und WPF User Controls

Um Daten zwischen zenon und WPF User Controls auszutauschen gibt es verschiedene Möglichkeiten.

- Datenaustausch über Dependency Properties (auf Seite 108)
- Datenaustausch über VSTA (auf Seite 111)

Datenaustausch über Dependency Properties

Der eleganteste und sicherste Weg, um Daten zwischen zenon und selbsterstellten WPF User Controls auszutauschen, ist mittels Dependency Properties.

Als Grundlage dient das, im Kapitel Erstellung eines einfachen WPF User Controls mit Code Behind Funktionalität (auf Seite 97) erstellte WPF User Control Projekt (**WPFUserControlLibrary**).

Auch in diesem Kapitel wird der Fokus rein auf die Kernthematik (in diesem Fall Dependency Properties und der Datenaustausch zwischen dem User Control und zenon) gelegt. Spezifische WPF-Features wie Databinding, etc. sowie explizite Fehlerbehandlung werden nicht behandelt.

ERWEITERUNGEN IM CODE

1. Erzeugen Sie das `TextChanged` Event für das Element `textBoxA` in der Datei **UserControl1.xaml**

   ```
   TextChanged="textBoxA_TextChanged"
   ```

2. Fügen Sie die folgenden Codezeilen in der Klasse `UserControl1` der Code-Behind Datei (**UserControl1.xaml.cs**) ein

```
/// <summary>
/// Gets or sets the ValueA.
/// </summary>
public double ValueA
{
  get
  {
    return (double)GetValue(ValueADependencyProperty);
  }
  set
  {
    SetValue(ValueADependencyProperty, value);
  }
}

/// <summary>
/// Dependency property for ValueA
/// </summary>
public static readonly DependencyProperty ValueADependencyProperty =
     DependencyProperty.Register("ValueA", typeof(double),
     typeof(UserControl1), new FrameworkPropertyMetadata(0.0, new
PropertyChangedCallback(OnValueADependencyPropertyChanged)));
```

```
    /// <summary>
    /// Called when [value a dependency property changed].
    /// </summary>
    /// <param name="source">The source.</param>
    /// <param name="e">The <see cref="DependencyPropertyChangedEventArgs"/> instance
containing the event data.</param>
    private static void OnValueADependencyPropertyChanged(DependencyObject source,
DependencyPropertyChangedEventArgs e)
    {
      UserControl1 control = source as UserControl1;
      if (control != null)
      {
        try
        {
          control.ValueA = (double)e.NewValue;
          control.textBoxA.Text = control.ValueA.ToString();
        }
        catch (Exception)
        {}
      }
    }

    /// <summary>
    /// Handles the TextChanged event of the textBoxA control.
    /// </summary>
    /// <param name="sender">The source of the event.</param>
    /// <param name="e">The <see cref="TextChangedEventArgs"/> instance containing the
event data.</param>
    private void textBoxA_TextChanged(object sender, TextChangedEventArgs e)
    {
      try
      {
        ValueA = Convert.ToDouble(textBoxA.Text);
      }
      catch (Exception)
      {}
    }
```

Abschließend bauen Sie die Solution.

> **Info**
>
> In diesem Beispiel wird ein numerisches Property (`double`) verwendet. Es können auch andere einfache Datentypen (wie `bool`, `string`, `int`, etc.) verwendet werden.

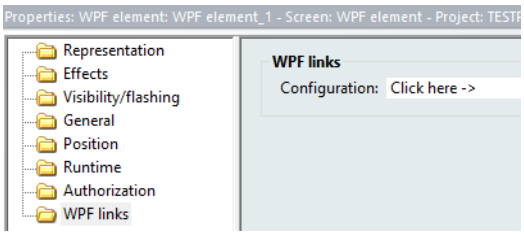
VERKNÜPFUNG IN ZENON

1. Aktualisieren Sie das WPF User Control (DLL) im zenon Editor.

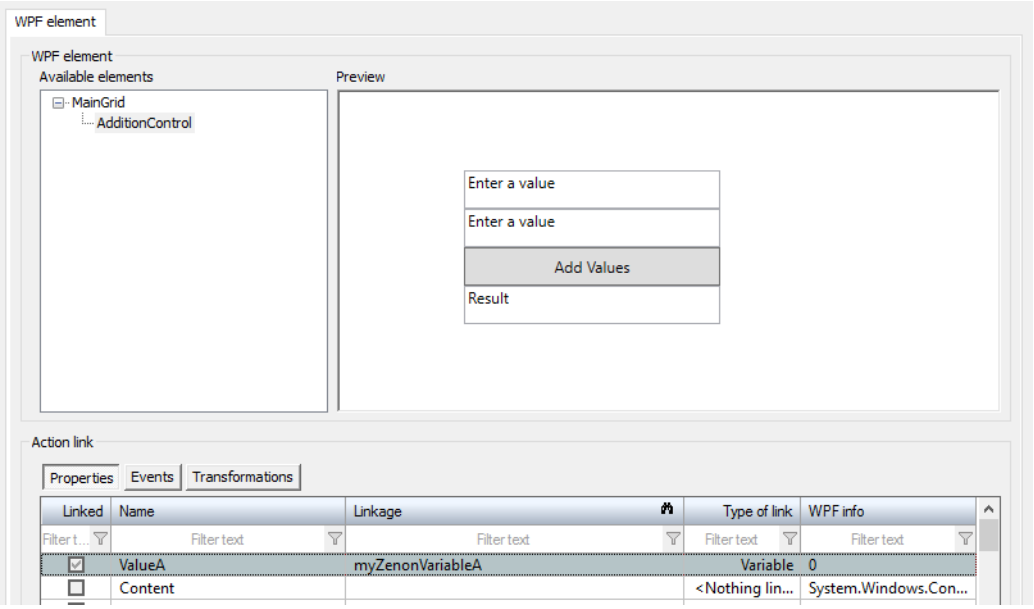
   Gehen Sie dazu wie in Kapitel Erstellung eines einfachen WPF User Controls mit Code Behind Funktionalität (auf Seite 97) vor und beachten Sie, dass unter Umständen der zenon Editor neu gestartet werden muss, falls zuvor schon eine andere Version des Assemblys (DLL) vom Editor geladen wurde.

2. Erstellen Sie eine numerische Variable in zenon und verlinken Sie diese zu einem Dynamischen Textelement im Bild neben dem WPF Element mit Ihrem User Control.

3. Öffnen Sie das Bild welches das WPF Element beinhaltet und wählen Sie beim WPF Element unter **WPF links**: **Configuration**



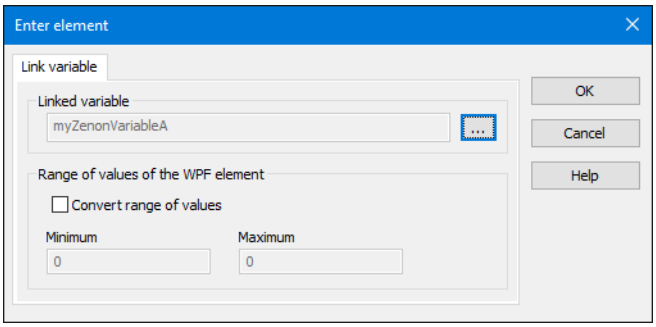
4. Expandieren Sie die Knoten im Tree links oben und wählen Sie **AdditionControl**



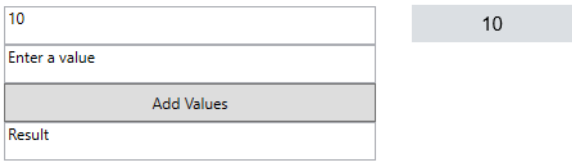
5. Selektieren Sie die Zeile mit `ValueA` (dies ist der Name des zuvor im Code erstellten Properties) und wählen Sie bei **Type of link: Variable**

   Tipp: Geben Sie Properties einen Prefix, damit diese leichter gefunden werden können, z.B: `_ValueA`

6. Wählen Sie dazu in der Spalte unter **Linkage** die zuvor in zenon erstellte Variable aus



7. Bestätigen Sie die Dialoge mit OK und erstellen Sie die Runtime Dateien neu.
8. Starten Sie die Runtime um das WPF User Control zu testen



9. Wird der Wert im User Control geändert, ändert sich automatisch der Wert in zenon und umgekehrt.
10. Natürlich können Sie das Control wie im Kapitel Debuggen des WPF User Controls zur Runtime (auf Seite 103) beschrieben debuggen sowie weitere Dependency Properties erstellen.

> **Info**
>
> *Um (automatisch) beim Initialisieren (z.B. beim Aufschalten des User Controls) auf die Werte von DependencyProperties zuzugreifen kann dass* `UserControl_Loaded` *Event verwendet werden.*

Datenaustausch über VSTA

Über VSTA können ebenfalls Daten zwischen zenon und WPF User Controls ausgetauscht werden.

Folgende Methoden der API werden dazu verwendet:

- `get_WPFProperty` (lesen von Werten)
- `set_WPFProperty` (schreiben von Werten)

Das hier verwendete Beispiel basiert auf dem im Kapitel Datenaustausch über Dependency Properties (auf Seite 108) behandelten Beispiel.

ERSTELLUNG EINES VSTA MAKROS FÜR DEN DATENAUSTAUSCH ZWISCHEN ZENON UND DEM WPF USER CONTROL

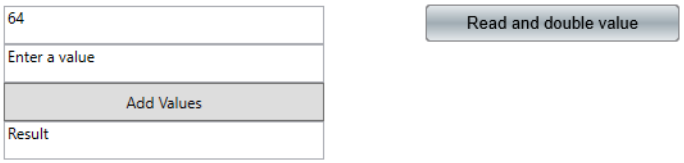
1. Erzeugen Sie folgendes VSTA Makro im ProjectAddin des zenon Projekts

```
/// <summary>
/// Sample Macro for data exchange between VSTA and a WPF User Control
/// </summary>
public void MacroWPFAccess()
{
    //Get the Screen and Element hosting the WPF User Control
    zenOn.IDynPicture myWPFScreen = this.DynPictures().Item("Screen");
    zenOn.IElement myWPFElement = myWPFScreen.Elements().Item("WPF_Element");

    //Read the current value from the WPF Element property
    double currentValue = Convert.ToDouble(myWPFElement.get_WPFProperty("AdditionControl", "ValueA"));

    //Double the value and write it back to the WPF Element property
    myWPFElement.set_WPFProperty("AdditionControl", "ValueA", currentValue * 2);
}
```

   Wobei:

   - `"Screen"` der Name des zenon Bildes in dem sich das WPF Element befindet ist
   - `"WPF_Element"` der Name des WPF Elements welches das WPF User Control behinhaltet ist
   - `"AdditionControl"` der Name des WPF User Controls selbst ist (definiert in der Datei (**UserControl1.xaml**)
   - `"ValueA"` der Name des User Control Properties ist

2. Erzeugen Sie eine VSTA-Makro ausführen Funktion und verlinken Sie diese mit einem Button in dem Bild in dem sich auch das WPF Element befindet
3. Starten Sie die Runtime, um die Änderungen zu testen



   Beim Ausführen des Makros wird der Wert vom Control gelesen, verdoppelt und zurückgeschrieben.

> **Info**
>
> *Die, für diese Methode zum Datenaustausch, verwendeten User Control Properties müssen nicht zwingend als Dependency Properties, wie in diesem Beispiel ausgeführt werden. Es können auch "Standard" Properties verwendet werden, siehe dazu auch im Kapitel Zugriff über VSTA "Variablen-Link" (auf Seite 113).*

6.3.4 Zugriff auf das zenon (Runtime) Objektmodell aus einem WPF User Control

Für den Zugriff auf das zenon Objektmodell aus einem WPF User Control heraus, gibt es verschiedene Möglichkeiten. In den folgenden Kapiteln werden diese näher erläutert.

Zugriff über VSTA "Variablen-Link"

Um Zugriff auf die zenon Runtime COM Schnittstelle per "Variablen-Link" zu bekommen gehen Sie wie folgt vor. Als Ausgangsbeispiel dient das im Kapitel Erstellung eines einfachen WPF User Controls mit Code-Behind Funktionalität (auf Seite 97).

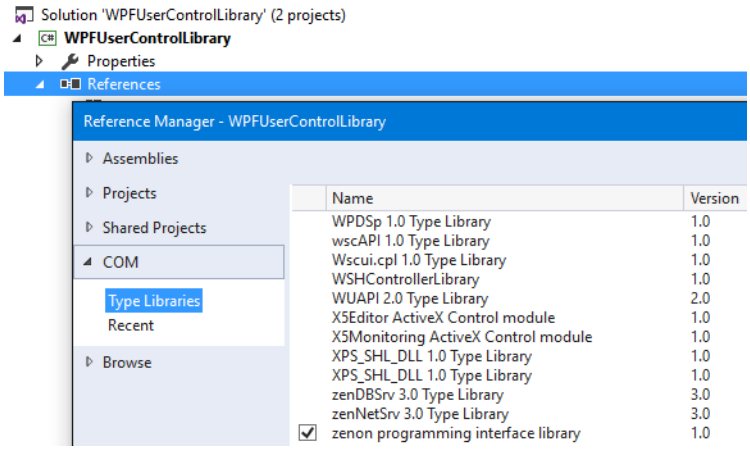
Info

Der folgende Code soll exemplarisch zeigen, wie der COM Zugriff auf die zenon Runtime realisiert werden kann und beschränkt sich dabei auf die Basisfunktionalität. Auf explizites Fehlerhandling, etc. wird verzichtet.

NOTWENDIGE ANPASSUNGEN IM WPF USER CONTROL

Im WPF User Control Projekt (**WPFUserControlLibrary**) sind die folgenden Schritte notwendig.

Zunächst muss eine Referenz auf die zenon COM Schnittstelle eingebunden werden.



Danach muss in der Klasse `UserControl1` der folgende Code eingefügt werden:

```
//The zenon Project
zenOn.Project zenonProject = null;

/// <summary>
/// Property for the Variable link via VSTA
/// </summary>
public object zenonVariableLink
{
  get { return null; }
  set
  {
    if (value != null && zenonProject == null)
    {
      zenOn.Variable zenonVariable;
      try
      {
        zenonVariable = (zenOn.Variable)value;
      }
      catch (Exception)
      {
        return;
      }
      if ((zenonVariable!= null) && (!string.IsNullOrEmpty(zenonVariable.Name)))
      {
        zenonProject = zenonVariable.Parent.Parent;
      }
    }
  }
}

/// <summary>
/// Trigger used to notify the control from VSTA to release the COM resources
/// </summary>
public object zenonReleaseTrigger
{
  get { return null; }
  set
  {
    if ((bool)value && zenonProject != null)
```

```
    {
      try
      {
        Marshal.FinalReleaseComObject(zenonProject);
      }
      catch (Exception)
      {
        return;
      }
      zenonProject = null;
      GC.Collect();
      GC.WaitForPendingFinalizers();
      GC.Collect();
    }
  }
}
```

Wobei auf die Properties `zenonVariableLink` (zur Initialisierung des COM Objekts) und `zenonReleaseTrigger` (zum Freigegeben des COM Objekts) später von VSTA aus (schreibend) zugegriffen wird.

Um den COM Zugriff schnell auf sehr einfachem Weg zu testen, kann im bestehenden Button Klick Event des User Controls die folgende Code Zeile eingefügt werden.

```
private void buttonAdd_Click(object sender, RoutedEventArgs e)
{
  if (zenonProject != null)
  {
    MessageBox.Show(zenonProject.Name);
  }
  return;
  ...
```

Info

In diesem Beispiel wird eine Variable vom Typ `zenOn.Project` verwendet. Es können natürlich auch andere Objekte sowie Events, etc. des zenon Objektmodels verwendet werden.

NOTWENDIGE ANPASSUNGEN IM ZENON PROJEKT/VSTA CODE

Im VSTA Code sind die folgenden Schritte notwendig:

Erstellung eines VSTA-Makros für die Initialisierung

```
/// <summary>
/// Macro for API initialization in the WPF User Control
/// </summary>
public void MacroWPFInit()
{
  zenOn.IDynPicture myWPFScreen = this.DynPictures().Item("Screen");
  zenOn.IElement myWPFElement = myWPFScreen.Elements().Item("WPF_Element");
  myWPFElement.set_WPFProperty("AdditionControl", "zenonVariableLink",
this.Variables().Item(0));
}
```

Erstellung eines VSTA-Makros für das Freigeben

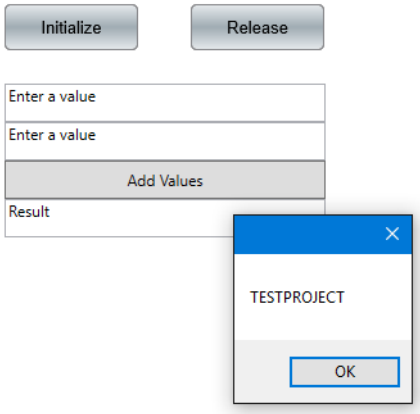
```
/// <summary>
/// Macro for API release in the WPF User Control
/// </summary>
public void MacroWPFRelease()
{
  zenOn.IDynPicture myWPFScreen = this.DynPictures().Item("Screen");
  zenOn.IElement myWPFElement = myWPFScreen.Elements().Item("WPF_Element");
  myWPFElement.set_WPFProperty("AdditionControl", "zenonReleaseTrigger", true);
}
```

Erstellen Sie zwei VSTA Makro ausführen Funktionen, die mit Buttons verknüpft werden, welche sich im selben Bild wie das WPF Element befinden.

Starten Sie nun die Runtime um die Funktionalität zu testen

- Führen Sie das Makro zum Initialisieren aus

- Betätigen Sie den Button im WPF User Control, es erscheint eine MessageBox mit dem Projektnamen des Projektes



- Führen Sie das Makro zum Freigeben aus

Um das User Control zu debuggen kann wie im Kapitel Debuggen des WPF User Controls zur Runtime (auf Seite 103) beschrieben vorgegangen werden.

Tipp

Das Initialisieren und Freigeben des COM Objekts in diesem Beispiel erfolgt zur einfacheren Demonstration über VSTA Makro Funktionen. Abhängig vom Anwendungsfall, bzw. im praxisnahen Einsatz sind dafür Events in VSTA besser geignet.

Beispielsweise kann der Code zum Initialiseren im `_Open` Event des Bildes mit dem WPF Element ausgeführt werden und der Code zum Freigeben im `_Close` Event.

Der hier beschriebene Mechanismuss wird auch im Kapitel Anzeige von WPF-Elementen im zenon Web Client (auf Seite 164) verwendet.

Achtung

Werden COM Objekte in WPF User Controls verwendet, müssen diese immer vor Zerstörung des WPF User Controls (vor dem Schließen des Bildes, vor dem Beenden der Runtime, vor dem Nachladen) explizit freigegeben werden.

Zugriff über Marshalling

Um Zugriff auf die zenon Runtime COM Schnittstelle per Marshalling zu bekommen, gehen Sie wie folgt vor. Als Ausgangsbeispiel dient das im Kapitel Erstellung eines einfachen WPF User Controls mit Code-Behind Funktionalität (auf Seite 97).

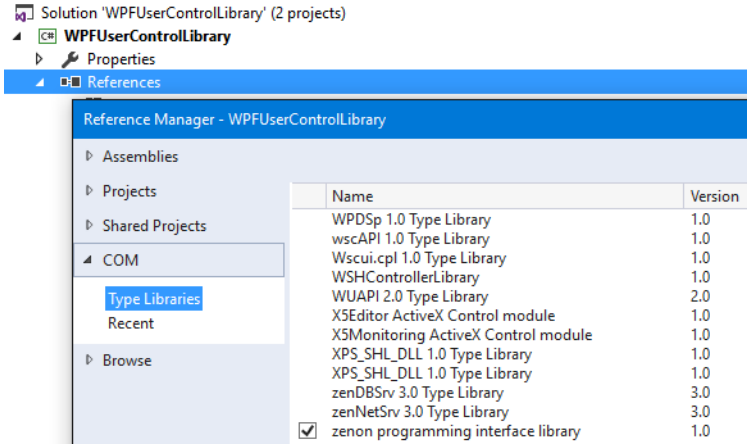
Info

Der folgende Code soll exemplarisch zeigen wie der COM Zugriff auf die zenon Runtime realisiert werden kann und beschränkt sich dabei auf die Basisfunktionalität. Auf explizites Fehlerhandling, etc. wird verzichtet.

NOTWENDIGE ANPASSUNGEN IM WPF USER CONTROL

Im WPF User Control Projekt (**WPFUserControlLibrary**) sind die folgenden Schritte notwendig.

Zunächst muss eine Referenz auf die zenon COM Schnittstelle eingebunden werden.



Danach muss in der Klasse `UserControl1` der folgende Code eingefügt werden:

```
//The zenon Project
zenOn.Project zenonProject = null;
```

Weiters muss der Konstruktor des User Controls um die untenstehenden Zeilen erweitert werden (zur Initialisierung des COM Objekts):

```
/// <summary>
/// Constructor for UserControl1, initialize COM Object
/// </summary>
public UserControl1()
{
  InitializeComponent();

  try
  {
```

```
    zenonProject =
((zenOn.Application)Marshal.GetActiveObject("zenOn.Application")).Projects().Item("TES
TPROJECT");
  }
  catch (Exception)
  {
  }
}
```

Im `UserControl_Unloaded` Event muss das COM Objekt freigegeben werden:

```
/// <summary>
/// Release COM Object
/// </summary>
private void UserControl_Unloaded(object sender, RoutedEventArgs e)
{
  try
  {
    if (zenonProject != null)
    {
      Marshal.FinalReleaseComObject(zenonProject);
      zenonProject = null;
    }
  }
  catch (Exception)
  {
  }
}
```

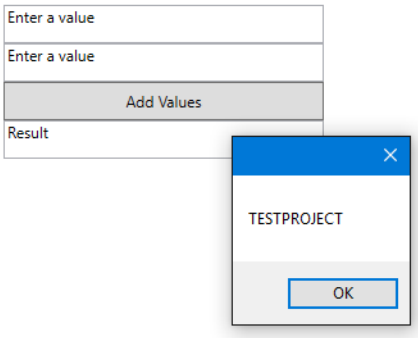
Um den COM Zugriff schnell auf sehr einfachem Weg zu testen, kann im bestehenden Button Klick Event des User Controls die folgende Code Zeile eingefügt werden.

```
private void buttonAdd_Click(object sender, RoutedEventArgs e)
{
  if (zenonProject != null)
  {
    MessageBox.Show(zenonProject.Name);
  }
  return;
```

. . .

Bauen Sie nun die Solution und aktualisieren sie das WPF User Control im zenon Projekt.

Starten Sie die Runtime, um das User Control zu testen.



Um das User Control zu debuggen, kann wie im Kapitel Debuggen des WPF User Controls zur Runtime (auf Seite 103) beschrieben, vorgegangen werden.

Info

In diesem Beispiel wird eine Variable vom Typ `zenOn.Project` verwendet. Es können natürlich auch andere Objekte sowie Events, etc. des zenon Objektmodels verwendet werden.

Achtung

Werden COM Objekte in WPF User Controls verwendet, müssen diese immer vor Zerstörung des WPF User Controls (vor dem Schließen des Bildes, vor dem Beenden der Runtime, vor dem Nachladen) explizit freigegeben werden.

Info

Im zenon WebClient ist kein Zugriff per Marshalling möglich. Wird dort ein Zugriff auf die COM Schnittstelle benötigt, muss die im Kapitel Zugriff über VSTA "Variablen-Link" (auf Seite 113) beschriebene Methode verwendet werden.

6.4 Engineering in zenon

Um WPF User Controls in zenon nutzen zu können, muss sowohl auf dem Editor-Rechner wie auch auf dem Runtime Rechner das Microsoft Framework in der Version 3.5 (oder höher, je nach verwendeter .NET Framework Version im User Control) installiert sein.

BEDINGUNGEN FÜR DIE WPF-DARSTELLUNG IN ZENON

Die Dynamisierung steht zurzeit für einfache Variablentypen (numerische Datentypen sowie String) zur Verfügung. Arrays und Strukturen können nicht dynamisiert werden.

Daher sind folgende WPF-Funktionen in zenon einsetzbar:

- Element-Eigenschaften, die einfachen Datentypen entsprechen wie `String, Int, Double, Bool` usw.
- Element-Eigenschaften vom Typ „Object", die mit einfachen Datentypen gesetzt werden können
- Element-Ereignisse können mit Funktionen verwendet werden, die Parameter der Ereignisse sind aber in zenon nicht verfügbar und auswertbar
- Elementtransformation, für die ein **RenderTransform** beim Element in der XAML-Datei vorhanden ist

  Achtung: Befindet sich bei Transformationen der Content außerhalb des Bereichs des WPF-Elements wird dieser nicht gezeichnet

Hinweis zu Schatten: Für WPF-Elemente kann in zenon kein Schatten dargestellt werden.

> **Achtung**
>
> *Wenn Runtime Dateien für ein Projekt für eine Version vor 6.50 erzeugt werden, dann werden vorhandene **WPF-Elemente** nicht in die Runtime-Bilder eingebunden.*

6.4.1 CDWPF-Dateien (Sammeldateien)

Eine CDWPF-Datei (mit der Endung ***.cdwpf**) ist eine umbenannte ZIP-Datei welche die folgende Komponenten beinhaltet:

- XAML-Datei (zum Referenzieren des User Control Assembly)
- DLL-Datei (das eigentliche WPF User Control, optional)
- Vorschaugrafik (zur Vorschau, optional)

Regeln für die Verwendung von Sammeldateien:

- Die Dateien (XAML, DLL, Vorschaugrafik) können direkt in der CDWPF-Datei oder in einem gemeinsamen Ordner liegen.
- Der Name der Sammeldatei muss dem Namen der XAML-Datei entsprechen.
- Es darf nur eine XAML-Datei enthalten sein.
- Die Vorschaugrafik sollte klein und nicht über 64 Pixel hoch sein.
  Name der Vorschaudatei: **preview.png** oder der Name der XAML Datei mit der Endung **png**.
- Es können beliebig viele Assemblies verwendet werden. Die Unterscheidung erfolgt auf Basis der Dateiversion.
- Sammeldateien müssen keine Assembly enthalten.
- Es werden alle Unterordner durchsucht und nur Dateien vom Typ ***.dll**, ***.xaml** oder ***.png** berücksichtigt.
- Wird eine Sammeldatei (***.cdwpf**) durch eine Datei mit einer anderen Version ersetzt, müssen alle entsprechenden CDWPF-Dateien in allen Symbolen und Bildern in allen Projekten angepasst werden.

6.4.2 WPF-Element anlegen

Um ein WPF-Element anzulegen:

1. Wählen Sie in der Symbolleiste der Elemente das Symbol für **WPF-Element** oder den entsprechenden Eintrag im Menü **Elemente.**
2. Wählen Sie im Hauptfenster den Startpunkt.
3. Ziehen Sie das Element mit der Maus auf.
4. Wählen Sie in den Eigenschaften in der Gruppe **Darstellung** die Eigenschaft **XAML-Datei**.
5. Der Datei-Auswahldialog öffnet sich.
6. Wählen Sie die gewünschte Datei aus.
   Gültige sind Dateien vom Format:
   - *.xaml: Extensible Application Markup Language
   - *.cdwpf: WPF Sammeldatei, zeigt auch Vorschaubild

   (Die Datei muss bereits im Projektmanager unter **Dateien/Grafiken** vorhanden sein oder im Dialog neu angelegt werden.)
7. Konfigurieren Sie die Verknüpfungen (auf Seite 123).

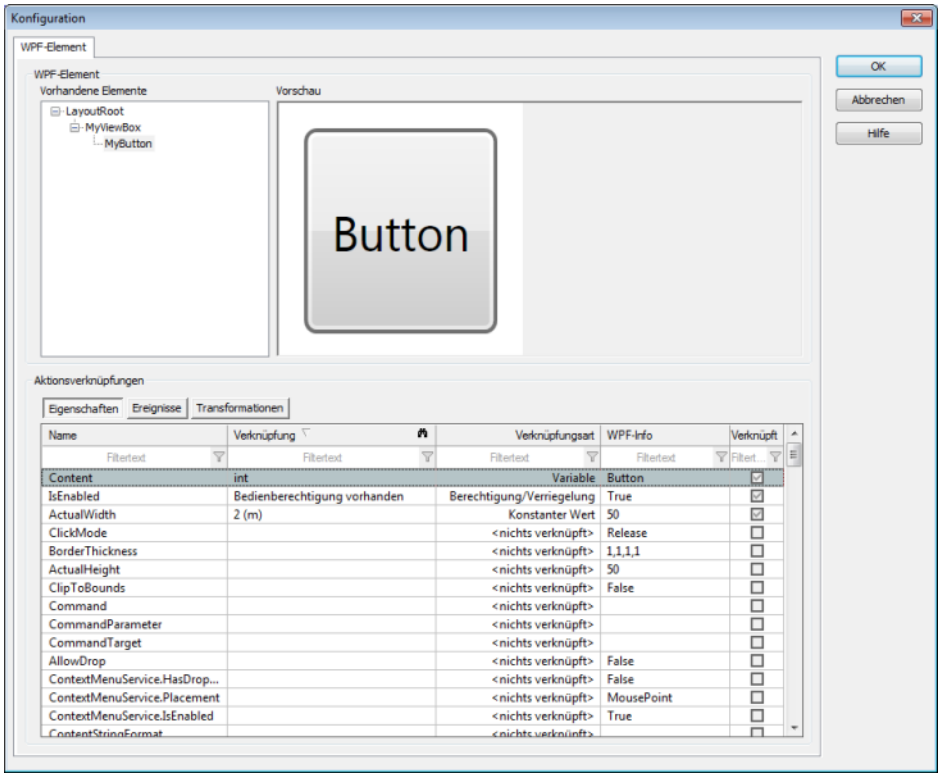
> **Info**
>
> *Sollen referenzierte Assemblies verwendet werden, beachten Sie die Hinweise im Kapitel Referenzierte Assemblies (auf Seite 81).*

6.4.3 Konfiguration der Verknüpfung

Um ein WPF-Element zu konfigurieren

1. Wählen Sie in den Eigenschaften in der Gruppe **WPF-Verknüpfungen** die Eigenschaft **Konfiguration**.
2. Der Dialog mit drei Registerkarten öffnet sich mit einer Vorschau der XAML-Datei sowie der in der Datei vorhandenen Elemente.

DIALOG KONFIGURATION



Parameter	Beschreibung
Vorhandene Elemente	Zeigt in einer Baumstruktur die benannten Elemente der Datei an. Das gewählte Element kann mit Prozessdaten verknüpft werden. Die Zuordnung von **WPF** zur Prozessdaten basiert auf den Elementname. Daher werden Elemente nur angezeigt, wenn sie und die dazugehörigen übergeordneten Elemente einen Namen besitzen. Zuordnungen werden in den Registerkarten **Eigenschaften**, **Ereignisse** und **Transformationen** konfiguriert und angezeigt. Tipp: Werden die entsprechenden Elemente nicht angezeigt, prüfen Sie in der XAML-Datei ob diese mit einem Namen versehen ist (z.B.: `<Grid Name="GridName">`).
Vorschau	Das ausgewählte Element wird in der Vorschau blinkend dargestellt.
Eigenschaften (auf Seite 125)	Konfiguration und Darstellung der Eigenschaften (Variablen, Bedienberechtigungen, Verriegelungen, verknüpfte Werte).
Ereignisse (auf Seite 131)	Konfiguration und Darstellung der Ereignisse (Funktionen).
Transformationen (auf Seite 133)	Konfiguration und Darstellung der Transformationen.
Name	Name der Eigenschaft.
Verknüpfung	Auswahl der Verknüpfung.
Verknüpfungsart	Art der Verknüpfung (Variable, Berechtigung, Funktion)
WPF-Info	Zeigt für Eigenschaften den aktuellen Wert im WPF-Content an. Für den Anwender ist direkt ersichtlich, um welche Art von Eigenschaft es sich handelt (Boolean, String, etc.).
Verknüpft	Zeigt an, ob eine Eigenschaft aktuell verwendet wird. Per Default nicht in der Ansicht enthalten, kann über Kontextmenü->Spaltenauswahl ausgewählt werden.

Info

Im Konfigurationsdialog können nur logische Objekt angezeigt werden. Visuelle Objekte werden nicht angezeigt. Hintergründe und wie visuelle Objekte dennoch dynamisiert werden können lesen Sie im Abschnitt Zuordnung zenon Objekt zu WPF Content.

VERKNÜPFUNGEN BEARBEITEN

Alle konfigurierten Verknüpfungen können über die Eigenschaften des Elements bearbeitet werden. Klicken Sie dazu auf das Element und öffnen Sie die Eigenschaftengruppe **WPF-Verknüpfungen**. Hier können Sie Verknüpfungen weiter konfigurieren, ohne den Dialog öffnen zu müssen.

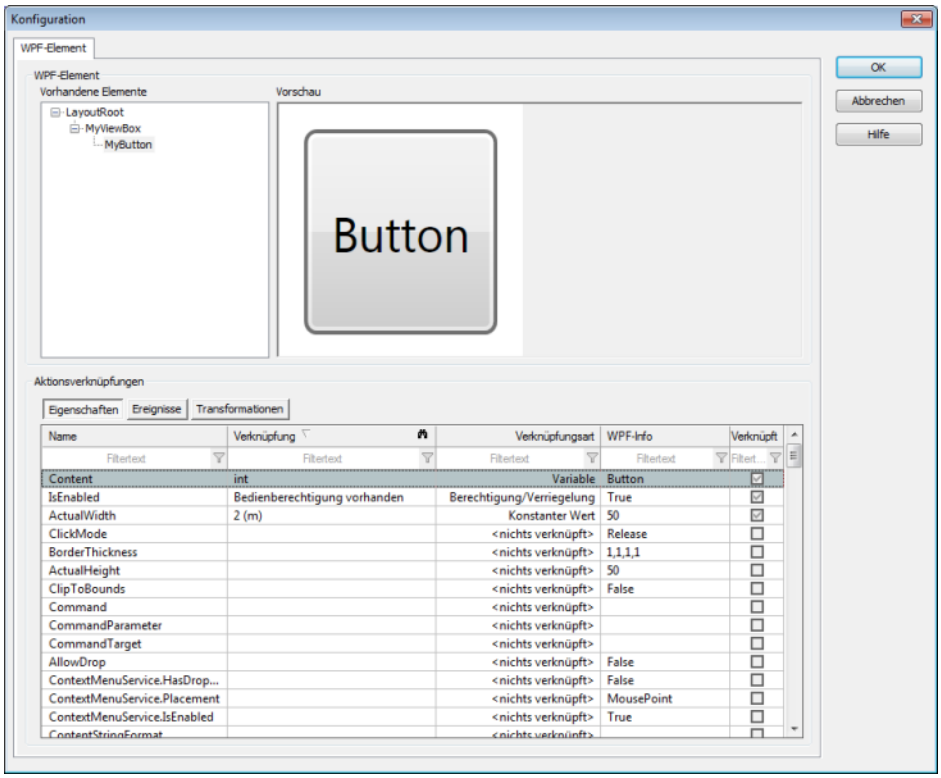
Einschränkungen:

- Der Verknüpfungstyp lässt sich hier nicht ändern.
- Neue Verknüpfungen können nur über der Konfigurationsdialog angelegt werden.
- Einfügen eines WPF-Elements in ein Symbol: WPF-Verknüpfungen können nicht exportiert werden.

Eigenschaften

Die Eigenschaften ermöglichen das Verknüpfen von:

- Variablen (auf Seite 127)
- Werten (auf Seite 128)
- Bedienberechtigungen und Verriegelungen (auf Seite 130)



Parameter	Beschreibung
Name	Name der Eigenschaft.
Verknüpfung	Verknüpfte Variable, Bedienberechtigung oder verknüpfter Wert. Klick in die Spalte öffnet den jeweiligen Auswahldialog, abhängig vom Eintrag in der Spalte **Verknüpfungsart.**
Verknüpfungsart	Auswahl einer Verknüpfung.
WPF-Info	Zeigt für Eigenschaften den aktuellen Wert im WPF-Content an. Für den Anwender ist direkt ersichtlich, um welche Art von Eigenschaft es sich handelt (Boolean, String, etc.).
Verknüpft	Zeigt an, ob eine Eigenschaft aktuell verwendet wird. Per Default nicht in der Ansicht enthalten, kann über Kontextmenü->Spaltenauswahl ausgewählt werden.

VERKNÜPFUNG ERSTELLEN

Um eine Verknüpfung zu erstellen:

1. markieren Sie die Zeile mit der Eigenschaft, die verknüpft werden soll
2. klicken Sie in die Zelle **Verknüpfungsart**
3. wählen Sie aus der Dropdownliste die gewünschte Verknüpfung.

   Zur Verfügung stehen:

   - `<nicht verknüpft>` (löst eine bestehende Verknüpfung)
   - `Berechtigung/Verriegelung`
   - `Konstanter Wert`
   - `Variable`
4. klicken Sie in die Zelle **Verknüpfung**
5. der Dialog zur Konfiguration der gewünschten Verknüpfung öffnet sich

Info

Eigenschaften von WPF und zenon können sich unterscheiden. Wird zum Beispiel die Eigenschaft ***Sichtbarkeit*** *verknüpft, stehen in .NET drei Werte zur Verfügung:*

0 - sichtbar

1 - unsichtbar

2- eingeklappt

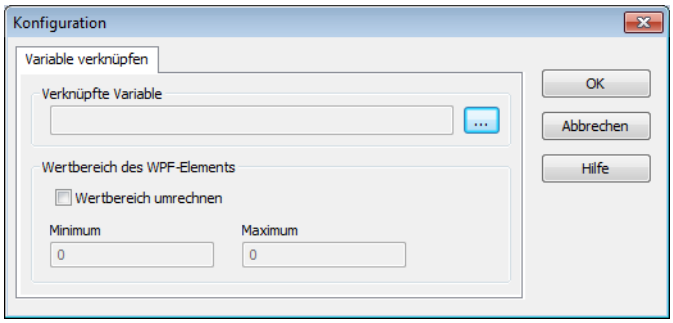
Diese Werte müssen über die verknüpfte zenon Variable abgebildet werden.

Variable verknüpfen

Um eine Variable mit einer WPF-Eigenschaften zu verknüpfen:

1. markieren Sie die Zeile mit der Eigenschaft, die verknüpft werden soll
2. klicken Sie in die Zelle **Verknüpfungsart**
3. wählen Sie aus der Combobox **Variable**
4. klicken Sie in die Zelle **Verknüpfung**
5. der Dialog zur Konfiguration der Variablen öffnet sich

Dieser Dialog gilt auch für Auswahl der Variablen bei Transformationen (auf Seite 133). Die Konfiguration ermöglicht auch Umrechnung von zenon in WPF-Einheiten.



Parameter	Beschreibung
Verknüpfte Variable	Auswahl der zu verknüpfenden Variablen. Klick auf Schaltfläche **...** öffnet Dialog zur Auswahl.
Wertbereich des WPF-Elements	Angaben zur Umrechnung der Variablenwerte in WPF-Werte.
Wertbereich umrechnen	`Aktiv`: WPF-Einheitenumrechnung ist eingeschaltet. Auswirkung zur Runtime: Der aktuelle zenon Wert (inkl. zenon Einheit) wird mittels normierter Minimum- und Maximum-Wert auf den WPF-Bereich umgerechnet. Zum Beispiel: Der Wert einer Variablen variiert von 100 bis 200. Bei der Variablen ist der normierte Bereich auf 100 - 200 eingestellt. Ziel ist, diese Wertänderung mittels eines WPF-Drehknopfs darzustellen. Dafür wird: ▸ bei **Transformation** die Eigenschaft **RotateTransform.Angle** mit der Variablen verknüpft ▸ **Wert anpassen** aktiviert ▸ ein WPF-Wertbereich von `0` bis `360` konfiguriert Jetzt wird der Drehknopf bei einen Wert von z. B. 150 um 180 Grad gedreht sein.
Minimum	Definiert den niedrigsten WPF-Wert.
Maximum	Definiert den höchsten WPF-Wert.
OK	Übernimmt Einstellungen und beendet Dialog.
Abbrechen	Verwirft Einstellungen und beendet Dialog.
Hilfe	Öffnet die Online-Hilfe.

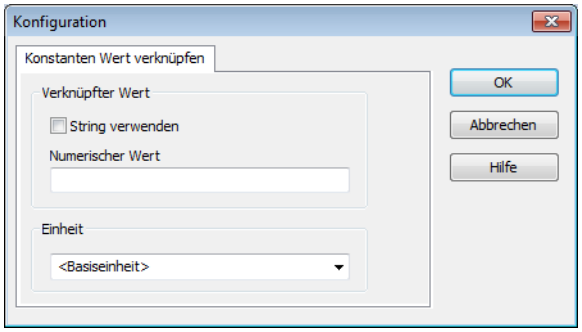
Werte verknüpfen

Verknüpfte Werte können entweder ein **String** oder ein numerischer Wert vom Typ **Double** sein. Beim Aufschalten des Bildes wird nach dem Laden des WPF-Contents der gewählte Wert im WPF-Content abgesetzt.

Um einen Wert mit einer WPF-Eigenschaften zu verknüpfen:

1. markieren Sie die Zeile mit der Eigenschaft, die verknüpft werden soll
2. klicken Sie in die Zelle **Verknüpfungsart**

3. wählen Sie aus der Combobox **Werteverknüpfungen**
4. klicken Sie in die Zelle **Verknüpfung**
5. der Dialog zur Konfiguration der Werteverknüpfung öffnet sich



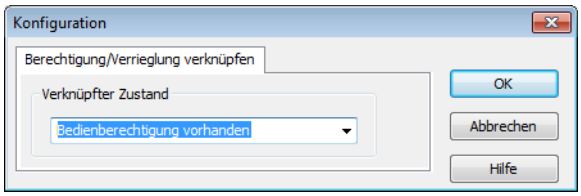
Parameter	Beschreibung
Verknüpfter Wert:	Eingabe eines numerischen Werts oder Stringwerts.
String verwenden	`Aktiv`: Ein String-Wert wird statt eines numerischen Werts verwendet. Stringwerte sind sprachumschaltbar. Zur Runtime wird der Text beim Aufschalten des Bildes übersetzt und im WPF-Content abgesetzt. Wird während das Bild geöffnet ist eine Sprachumschaltung durchgeführt, wird der Stringwert neu übersetzt und abgesetzt.
Stringwert/Numerischer Wert	Abhängig von der Auswahl der Eigenschaft **String verwenden** wird in dieses Feld ein numerischer Wert oder ein Stringwert eingegeben. Bei numerischen Werten kann zusätzlich eine Einheit Maßeinheit ausgewählt werden.
Einheit:	Auswahl einer Maßeinheit aus der Dropdownliste. Diese muss zuvor in der Einheitenumschaltung definiert worden sein. Die Maßeinheit wird dem numerischen Wert zugeordnet. Wird zur Runtime eine Einheitenumschaltung durchgeführt, wird der Wert umgerechnet auf die neue Maßeinheit und zum WPF-Content abgesetzt.
OK	Übernimmt Einstellungen und beendet Dialog.
Abbrechen	Verwirft Einstellungen und beendet Dialog.
Hilfe	Öffnet die Online-Hilfe.

Bedienberechtigung oder Verriegelung verknüpfen

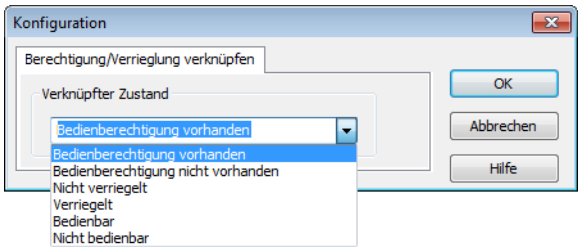
Bedienberechtigungen können nicht für das gesamte WPF-Element vergeben werden. Dem Element wird eine Benutzerebene zugewiesen. Berechtigungen werden innerhalb der Benutzerebene für einzelne Controls erteilt. Ist eine Berechtigung aktiv, wird der Wert `1` auf die Eigenschaft im Element geschrieben.

Um eine Bedienberechtigung oder Verriegelung mit einer WPF-Eigenschaften zu verknüpfen:

1. markieren Sie die Zeile mit der Eigenschaft, die verknüpft werden soll
2. klicken Sie in die Zelle **Verknüpfungsart**
3. wählen Sie aus der Dropdownliste **Bedienberechtigung/Verriegelung**
4. klicken Sie in die Zelle **Verknüpfung**
5. der Dialog zur Konfiguration der Berechtigungen öffnet sich



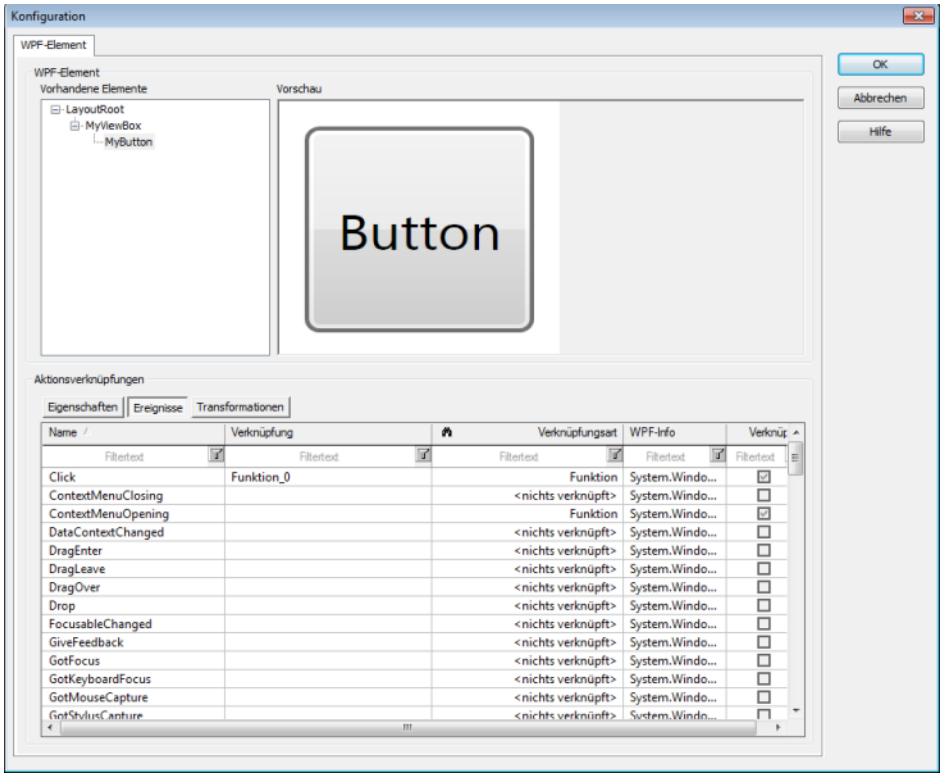
Parameter	Beschreibung
Berechtigung/Verriegelung verknüpfen	Festlegung der Berechtigungen.
Verknüpfter Zustand	Auswahl der Berechtigung, die mit einem WPF-Control verknüpft wird, aus der Dropdownliste. Zum Beispiel kann Sichtbarkeit und Bedienbarkeit eines WPF-Buttons vom Status des Benutzers abhängen.



Berechtigung	Beschreibung
`Bedienberechtigung vorhanden`	Besitzt der Benutzer genügend Rechte, um das **WPF-Element** zu bedienen, wird der Wert `1` auf die Eigenschaft geschrieben.
`Bedienberechtigung nicht vorhanden`	Besitzt der Benutzer nicht genügend Rechte, um das **WPF-Element** zu bedienen, wird der Wert `1` auf die Eigenschaft geschrieben.
`Nicht verriegelt`	Ist das Element nicht durch die Verriegelung gesperrt, wird der Wert `1` auf die Eigenschaft geschrieben.
`Verriegelt`	Ist das Element durch die Verriegelung gesperrt, wird der Wert `1` auf die Eigenschaft geschrieben.
`Bedienbar`	Wenn die Bedienberechtigung vorhanden ist und das Element nicht verriegelt ist, dann wird der Wert `1` auf die Eigenschaft geschrieben.
`Nicht bedienbar`	Fehlt die Bedienberechtigung oder ist das Element verriegelt, dann wird der Wert `1` auf die Eigenschaft geschrieben.

Ereignisse

Ereignisse ermöglichen das Verknüpfen von zenon Funktionen mit einem WPF-Element.

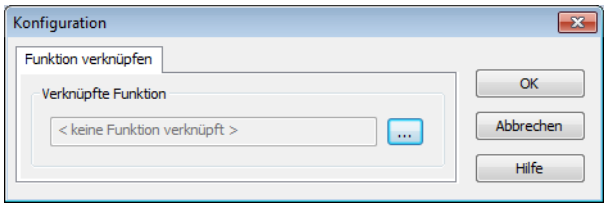


Parameter	Beschreibung
Name	Name der Eigenschaft.
Verknüpfung	Verknüpfte Funktion. Klick in die Zelle öffnet den Konfigurationsdialog**.**
Verknüpfungsart	Auswahl einer Verknüpfung. Klick in die Zelle öffnet den Auswahldialog.
WPF-Info	Zeigt für Eigenschaften den aktuellen Wert im WPF-Content an. Für den Anwender ist direkt ersichtlich, um welche Art von Eigenschaft es sich handelt (Boolean, String, etc.).
Verknüpft	Zeigt an, ob eine Eigenschaft aktuell verwendet wird. Per Default nicht in der Ansicht enthalten, kann über Kontextmenü->Spaltenauswahl ausgewählt werden.

FUNKTIONEN VERKNÜPFEN

Um eine Verknüpfung zu erstellen:

1. markieren Sie die Zeile mit der Eigenschaft, die verknüpft werden soll
2. klicken Sie in die Zelle **Verknüpfungsart**
3. wählen Sie aus der Dropdownliste Funktion
4. klicken Sie in die Zelle **Verknüpfung**
5. der Dialog zur Konfiguration der Funktion öffnet sich

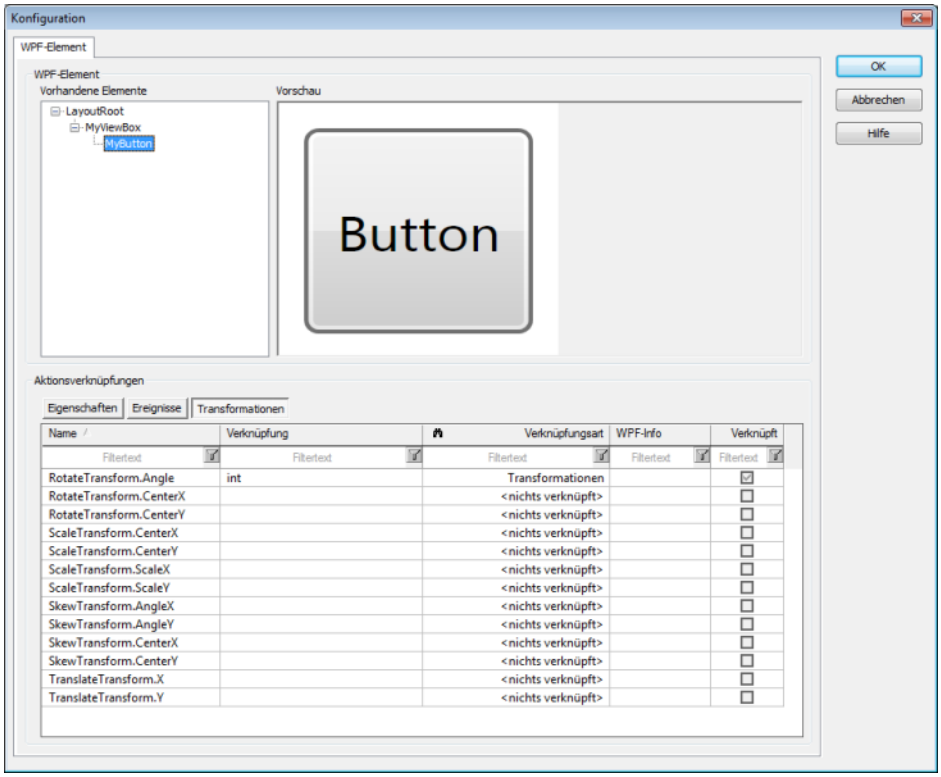


Parameter	Beschreibung
Verknüpfte Funktion	Auswahl der zu verknüpfenden Funktion. Klick auf Schaltfläche ... öffnet den Dialog zur Funktionsauswahl.
OK	Übernimmt Auswahl und schließt Dialog.
Abbrechen	Verwirft Änderungen und schließt Dialog.
Hilfe	Öffnet Online-Hilfe.

Transformation

Das **WPF-Element** unterstützt keine Dreh-Funktionalität. Befindet sich zum Beispiel das **WPF-Element** in einem Symbol und das Symbol wird gedreht, so dreht das **WPF-Element** nicht mit. Daher gibt es für WPF mit **Transformation** einen eigenen Mechanismus, um Elementen zu drehen oder anderweitig zu transformieren. Diese Transformationen werden in der Registerkarte **Transformation** konfiguriert.

Achtung: Falls sich Content außerhalb des Bereichs des **WPF-Elementes** befindet, geht dieser Teil des Contents verloren, oder wird nicht gezeichnet.



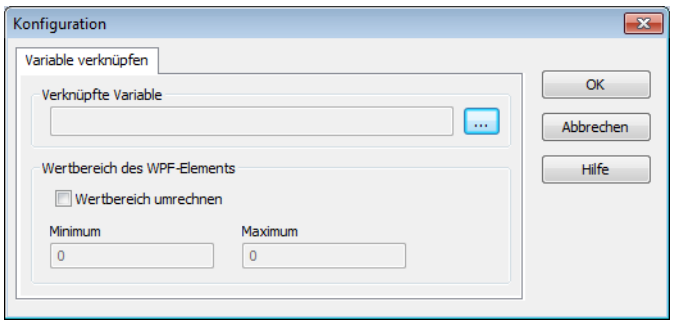
Parameter	Beschreibung
Name	Name der Eigenschaft.
Verknüpfung	Auswahl der verknüpften Variablen. Transformationen werden in XAML als Transformationsobjekte mit eigenen Eigenschaften dargestellt. Unterstützt ein Element eine Transformation, dann werden die möglichen Eigenschaften der Transformationsobjekte in der Listenansicht dargestellt. (Mehr dazu in: Button als WPF-XAML in zenon einbinden. (auf Seite 174) Wird zum Beispiel die verknüpfte Variable auf den Wert `10` gesetzt, dann wird dieser Wert auf das WPF-Ziel geschrieben und das Element von WPF um 10° gedreht.
Verknüpfungsart	Auswahl der Verknüpfungsart Transformation.
WPF-Info	Zeigt für Eigenschaften den aktuellen Wert im WPF-Content an. Für den Anwender ist direkt ersichtlich, um welche Art von Eigenschaft es sich handelt (Boolean, String, etc.).
Verknüpft	Zeigt an, ob eine Eigenschaft aktuell verwendet wird. Per Default nicht in der Ansicht enthalten, kann über Kontextmenü->Spaltenauswahl ausgewählt werden.

TRANSFORMATION VERKNÜPFEN

Um eine Transformation mit einer WPF-Eigenschaften zu verknüpfen:

1. markieren Sie die Zeile mit der Eigenschaft, die verknüpft werden soll
2. klicken Sie in die Zelle **Verknüpfungsart**
3. wählen Sie aus der Combobox **Transformation**
4. klicken Sie in die Zelle **Verknüpfung**
5. der Dialog zur Konfiguration der Variablen öffnet sich

Die Konfiguration ermöglicht auch Umrechnung von zenon in WPF-Einheiten.



Parameter	Beschreibung
Verknüpfte Variable	Auswahl der zu verknüpfenden Variablen. Klick auf Schaltfläche **...** öffnet Dialog zur Auswahl.
Wertbereich des WPF-Elements	Angaben zur Umrechnung der Variablenwerte in WPF-Werte.
Wertbereich umrechnen	`Aktiv`: WPF-Einheitenumrechnung ist eingeschaltet. Auswirkung zur Runtime: Der aktuelle zenon Wert (inkl. zenon Einheit) wird mittels normierter Minimum- und Maximum-Wert auf den WPF-Bereich umgerechnet. Zum Beispiel: Der Wert einer Variablen variiert von 100 bis 200. Bei der Variablen ist der normierte Bereich auf 100 - 200 eingestellt. Ziel ist, diese Wertänderung mittels eines WPF-Drehknopfs darzustellen. Dafür wird: ▸ bei **Transformation** die Eigenschaft **RotateTransform.Angle** mit der Variablen verknüpft ▸ **Wert anpassen** aktiviert ▸ ein WPF-Wertbereich von `0` bis `360` konfiguriert Jetzt wird der Drehknopf bei einen Wert von z. B. 150 um 180 Grad gedreht sein.
Minimum	Definiert den niedrigsten WPF-Wert.
Maximum	Definiert den höchsten WPF-Wert.
OK	Übernimmt Einstellungen und beendet Dialog.
Abbrechen	Verwirft Einstellungen und beendet Dialog.
Hilfe	Öffnet die Online-Hilfe.

6.4.4 Gültigkeit von XAML-Dateien

Die Gültigkeit von XAML-Dateien ist an einige Voraussetzungen gebunden:

- Richtige Namespaces
- Keine Class-Referenzen
- Skalierbarkeit

RICHTIGE NAMSPACES

Das **WPF-Element** kann ausschließlich WPF-Content darstellen, das heißt:

Nur XAML-Dateien mit richtigem WPF-Namespace können vom **WPF-Element** dargestellt werden. Dateien die einen Silverlight-Namespace verwenden, können nicht geladen oder angezeigt werden. In den meisten Fällen reicht es aber aus, den Silverlight-Namespace durch den WPF-Namespace zu ersetzen.

WPF-Namespaces:

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

KEINE VERWENDUNG VON CLASS-REFERENZEN

Da die XAML-Dateien dynamisch geladen werden, ist es nicht möglich, XAML-Dateien zu verwenden, die Referenzen auf Klassen (Schlüssel „Class" im Header) beinhalten. Funktionalität, die in einer eigenen C#-Datei programmiert wurde, kann nicht verwendet werden.

Um WPF User Controls mit Code Behind zu verwenden, muss wie im Kapitel Erstellung eines einfachen WPF User Controls mit Code Behind-Funktionalität (auf Seite 97) vorgegangen werden.

SKALIERBARKEIT

Soll der Inhalt eines **WPF-Elements** an die Größe des **WPF-Elements** angepasst werden, dann müssen die Controls des **WPF-Elements** in ein Control verschachtelt werden, das diese Funktionalität bietet, zum Beispiel eine **ViewBox**. Zusätzlich muss darauf geachtet werden, dass für diese Elemente die **Höhe** und **Breite** als **automatisch** konfiguriert werden.

ÜBERPRÜFUNG EINER XAML-DATEI AUF KORREKTHEIT

Um zu überprüfen, ob eine XAML-Datei das richtige Format hat:

- XAML-Datei im Internet-Explorer öffnen
  - lässt sie sich ohne zusätzliche Plugins (Java oder ähnliche) öffnen, dann kann diese Datei auch mit hoher Sicherheit von zenon geladen oder angezeigt werden
  - treten beim Laden der Datei Probleme auf, so werden diese im Internet-Explorer eingeblendet und es ist klar erkennbar, in welcher Zeile es Probleme gibt

Auch die Skalierung kann auf diese Art und Weise getestet werden: Ist die Datei richtig aufgebaut, wird sich der Inhalt an die Größe des Internet Explorer Fensters anpassen.

FEHLERMELDUNG

Wird in zenon eine ungültige Datei verwendet, so wird beim Laden der Datei im WPF-Element eine Fehlermeldung im Ausgabefenster angezeigt.

Zum Beispiel:

„Fehler beim Laden vom
`xaml-Datei:C:\ProgramData\COPA-DATA\SQL\781b1352-59d0-437e-a173-08563c3142e9\FILES\zenon\custom\media\UserControl1.xaml`

`Das Attribut "Class" ist im XML-Namespace "http://schemas.microsoft.com/winfx/2006/xaml" nicht vorhanden. Zeile 7 Position 2."`

6.4.5 Vorgefertigte Elemente

Eine Reihe von WPF-Elementen werden bereits mit zenon ausgeliefert oder stehen im Webshop zum Download bereit.

Alle WPF-Elemente haben Eigenschaften, welche die grafische Ausprägung des jeweiligen Elements bestimmen (Dependency Properties). Das Setzen der Werte mittels einer XAML-Datei oder die Verknüpfung der Eigenschaft über zenon kann das Aussehen in der Runtime direkt verändert werden. Die Tabellen in der Beschreibung der einzelnen Elemente enthalten die jeweiligen Dependency Properties, abhängig vom Control.

Verfügbare Elemente:

- Analoguhr (auf Seite 138)
- Bargraf vertikal (auf Seite 139)
- Comtrade-Viewer (auf Seite 140)
- Energieklassen-Diagramm (auf Seite 151)
- Fortschrittsanzeige (auf Seite 140)
- Pareto-Diagramm (auf Seite 152)
- Sankey-Diagramm (auf Seite 159)
- Rundanzeige (auf Seite 156)
- Temperaturanzeige (auf Seite 161)
- Universalregler (auf Seite 162)
- Wasserfall-Diagramm (auf Seite 163)

ASSEMBLY MIT NEUER VERSION ERSETZEN

Pro Projekt kann sowohl im zenon Editor als auch in der Runtime nur eine Assembly für ein WPF-Element genutzt werden. Sind in einem Projekt zwei Versionen einer Assembly vorhanden, wird die zuerst geladene Datei verwendet. Es erfolgt eine Benutzerabfrage, welche Version verwendet werden soll. Bei Beibehaltung der bisher genutzten Version sind keine weiteren Aktionen nötig. Wird eine neuere Version gewählt, müssen alle entsprechenden CDWPF-Dateien in allen Symbolen und Bildern in allen Projekten angepasst werden.

Hinweis Mehrprojektverwaltung: Wird in einem Projekt eine Assembly durch eine neuere Version ersetzt, muss sie in allen im Editor oder in der Runtime geladenen Projekten ersetzt werden.

Analoguhr - AnalogClockControl

Property	Funktion	Wert
ElementStyle	Form/Art des Elements.	Enum: ▸ SmallNumbers ▸ BigNumbers ▸ No
ElementBackgroundBrush	Farbe des Elementhintergrunds.	Brush
ElementGlasReflection	Aktivierung des Glaseffekts auf dem Element.	Visibility
Offset	Wert in Stunden (h), der die Zeitverschiebung zur Systemuhr darstellt.	Int16
OriginText	Text, der in der Uhr angezeigt wird (z. B. Ortsangabe).	String

Bargraf vertikal - VerticalBargraphControl

Property	Funktion	Wert
CurrentValue	Aktueller Wert, der angezeigt werden soll.	`Double`
MinValue	Minimalwert der Skala.	`Double`
MaxValue	Maximalwert der Skala.	`Double`
MajorTicksCount	Anzahl der Hauptticks der Skala.	`Integer`
MinorTicksCount	Anzahl der Nebenticks der Skala.	`Integer`
MajorTickColor	Farbe der Hauptticks der Skala.	`Color`
MinorTickColor	Farbe der Nebenticks der Skala.	`Color`
ElementBorderBrush	Farbe des Elementrahmens.	`Brush`
ElementBackgroundBrush	Farbe des Elementhintergrunds.	`Brush`
ElementGlasReflection	Aktivierung des Glaseffekts auf dem Element.	`Visibility`
ElementFontFamily	Elementschriftart.	`Font`
ScaleFontSize	Schriftgröße der Skala.	`Double`
ScaleFontColor	Schriftfarbe der Skala.	`Color`
IndicatorBrush	Farbe der Bargraf-Füllfarbe.	`Brush`
BargraphSeparation	Anzahl der Bargraf-Unterteilungstriche.	`Integer`
BargraphSeparationColor	Farbe der Skalenunterteilungen.	`Color`

Fortschrittsanzeige - ProgressBarControl

Property	Funktion	Wert
CurrentValue	Aktueller Wert, der angezeigt werden soll.	`Double`
MinValue	Minimalwert des Wertebereichs.	`Double`
MaxValue	Maximalwert des Wertebereichs.	`Double`
ProgressbarDivisionCount	Anzahl der Unterteilungen der (Boxed) Fortschrittsanzeige.	`Integer`
VisibilityText	Sichtbarkeit der Wertanzeige.	`Boolean`
TextSize	Schriftgröße der Wertanzeige.	`Double`
TextColor	Farbe der Wertanzeige.	`Color`
ProgressBarBoxedColor	Farbe der Rahmen der (Boxed) Fortschrittsanzeige.	`Color`
ProgressBarMarginDistance	Abstand der Progressbar-Box vom Elementrand (links, oben, rechts, unten).	`Double`
ProgressBarInactiveBrush	Indikatorfarbe nicht aktiv.	`Brush`
ProgressBarActiveBrush	Indikatorfarbe aktiv.	`Brush`
ProgressBarPadding	Abstand der Fortschrittsanzeige von der Progressbar-Box (links, oben, rechts, unten).	`Double`
ElementBorderBrush	Farbe des Elementrahmens.	`Brush`
ElementBackgroundBrush	Farbe des Elementhintergrunds.	`Brush`

COMTRADE-Viewer

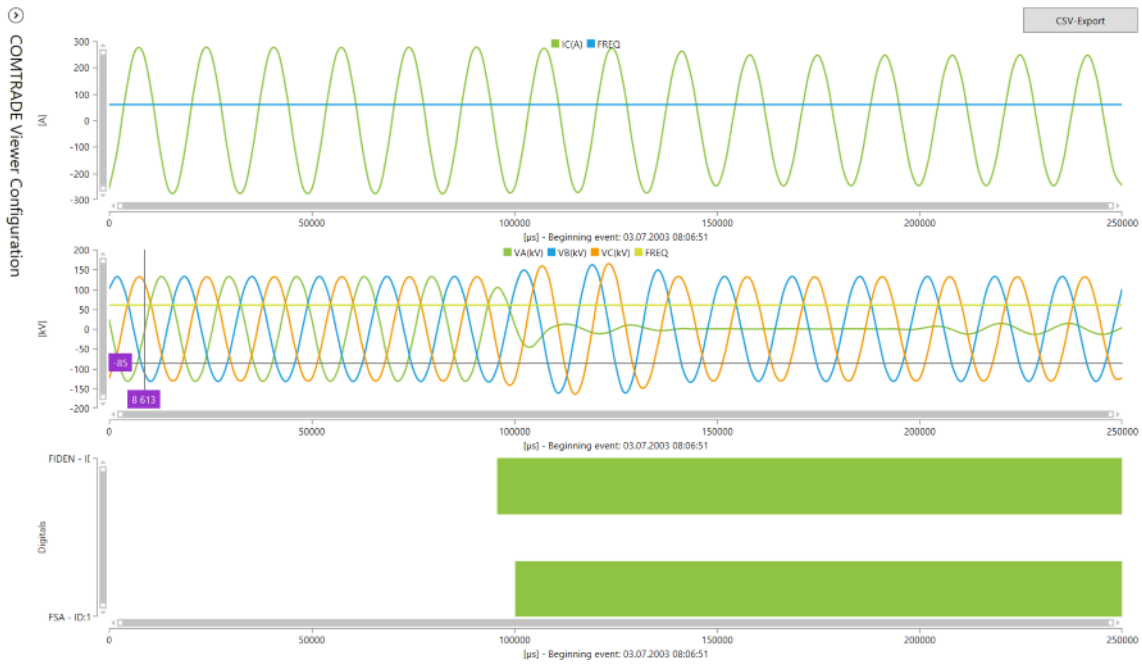
Das **COMTRADE-Viewer** WPF-Element steht Partnern von COPA-DATA zur Verfügung und ist für diese über das Partner Portal erhältlich.

Es dient zur grafischen Analyse von digitalen Stör- und Ereignisdatenaufzeichnungen einer COMTRADE-Datei.

Info

Das Control unterstützt IEEE C37.111 (IEEE Standard Common Format for Transient Data Exchange (COMTRADE) for Power Systems) normkonforme Dateien. Es können ASCII oder Binär Dateien nach Edition von 1999 oder 2013 visualisiert werden.

Ältere Dateien oder Dateien ohne Jahreskennung werden nicht unterstützt. Dies wird zur Runtime mit einem Warndialog angezeigt.



Möglichkeiten des **COMTRADE-Viewer** WPF-Controls in der zenon Runtime:

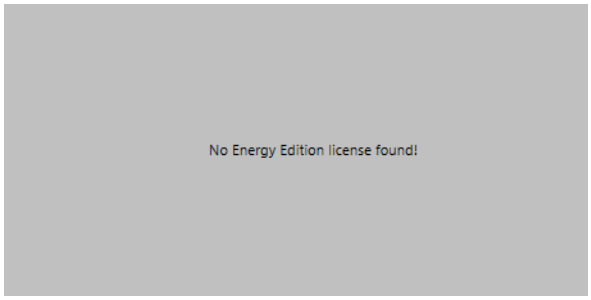
- Auswahl einer Datei im COMTRADE-Dateiformat
- Export der ausgewählten Objekte in eine CSV-Datei
- Visualisierung der gewählten COMTRADE-Datei:
  Hinweis: Die Anzeigefarben sind im zenon Editor konfigurierbar.
  - Strom (Sinuswellen-Darstellung)
  - Spannung (Sinuswellen-Darstellung)
  - digitale Signale (binäre Balkendiagramm-Darstellung)
  - Anzeige von Werten an einer gewählten Cursorposition.
  - Wird ein Element ausgewählt, welches weder Strom oder Spannung repräsentiert (z.B. Frequenz), wird dieses in beiden analogen Bereichen (Strom und Spannung) visualisiert.
- Navigation:

- Zoom-In sowie Zoom-Out per Mausrad, Bildlaufleiste und Multi-Touch Gesten
- Vergrößerung eines Bereichs
  Auswahl des Bereichs per Mausklick
- Verschieben des Anzeigebereichs per rechter Maustaste, Bildlaufleiste oder Multi-Touch Gesten.

> **Tipp**
>
> Um COMTRADE-Dateien auf den zenon Runtimerechner zu transportieren können Sie auch den Filetransfer des **IEC 61850 Treibers** oder den **FTP-Funktionsblock** von zenon Logic nutzen.
>
> Weitere Informationen dazu finden Sie in der Treiberdokumentation des IEC 61850 Treibers oder in der zenon Logic Dokumentation.

LIZENZIERUNG

Das **COMTRADE-Viewer** kann nur in einem zenon Editor mit einer gültigen Energy Edition Lizenz konfiguriert werden. Ist keine gültige Lizenz vorhanden, wird das WPF im Editor ausgegraut dargestellt. Für die Darstellung zur Runtime ist ebenfalls eine gültige Energy Edition Lizenz erforderlich.



Ansicht zur Runtime

Das **COMTRADE** WPF-Element bietet zur Runtime zwei Ansichten:

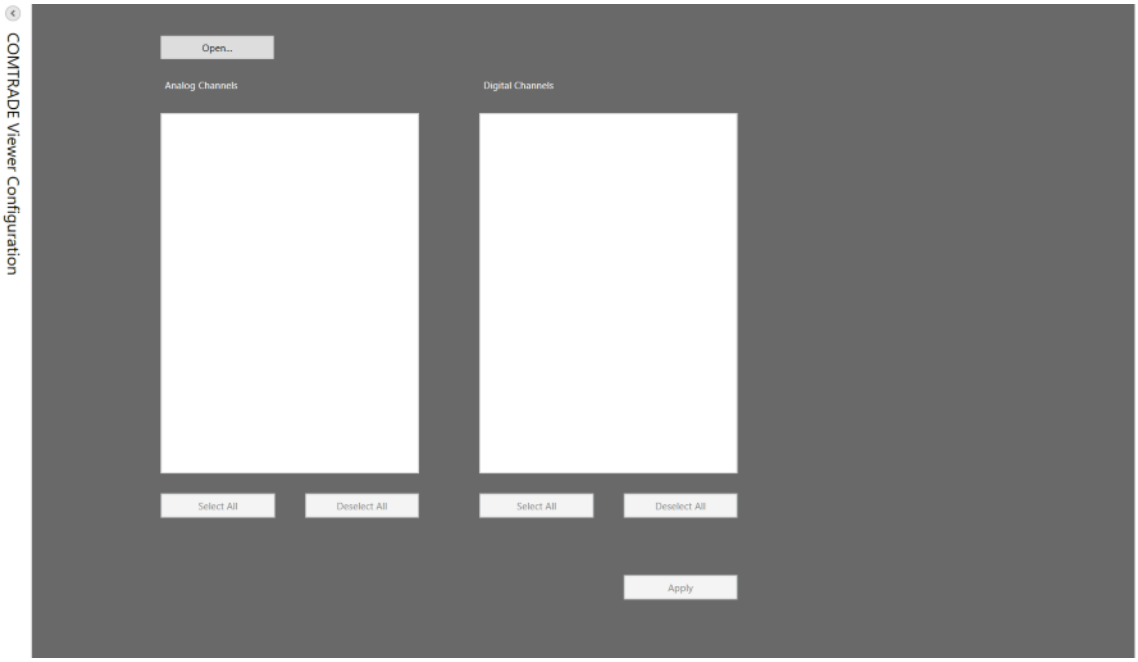
- Konfigurationsansicht
  - Auswahl einer COMTRADE-Konfigurationsdatei
  - Auswahl der anzuzeigenden Elemente
- Graphen Ansicht
  - Zoom-In und Zoom-Out
  - Anzeige von Werten an einer gewählten Cursorposition.
  - Export der gewählten Elemente in eine CSV-Datei

> **Info**
>
> Die Umschaltung zwischen den Ansichten ist im WPF-Element integriert. Eine zusätzliche Projektierung einer Bildumschaltfunktion ist nicht notwendig.

Runtime Ansicht - Konfigurationsseite

Wird in der Runtime ein Bild mit einem projektierten **COMTRADE-Viewer** WPF-Element aufgeschalten so ist die Anzeige der Konfigurationsseite jeweils leer.

Hinweis: Dies gilt auch, wenn in der zenon Runtime von einem anderen Bild zu diesem Bild mit der Bildumschaltfunktion gewechselt wird.



COMTRADE VIEWER CONFIGURATION

Die seitlich vertikal angeordnete Umschaltung **COMTRADE Viewer Configuration** wechselt die Anzeige von der Konfiguration zur Graphen Ansicht und umgekehrt.

DATEI AUSWÄHLEN

Der Button **Open...** öffnet den Dateiauswahldialog zur Auswahl einer Datei.

Für die Anzeige im Dateiauswahldialog findet eine Vorauswahl statt:

- Dabei werden Dateipaare von `*.cfg`– und `*.dat`-Dateien erkannt.
  Hinweis: Optionale Dateien vom Dateityp `*.hdr` oder `*.inf` werden nicht berücksichtigt.
- Angezeigt werden nur die entsprechenden `*.dat`-Dateien.
- Mit Klick auf die gewünschte Datei und Klick auf den Button **OK**, werden alle zugehörigen Dateien (`*.dat, *.cfg`) geladen.
- Es kann jeweils nur eine Datei geladen werden.
- Nach Laden der Datei werden die Inhalte der Datei in den Spalten **Analog Channels** und **Digital Channels** angezeigt.
  Die Bezeichnungen und Einheiten der Elemente haben Ihren Ursprung in der COMTRADE Konfiguration und können nicht geändert werden.

ANALOG CHANNELS

Parameter	Beschreibung
`[Liste der verfügbaren Kanäle]`	Auswahl der zu visualisierenden Elemente. Mehrfachauswahl mit Klick auf den gewünschten Eintrag in der Liste. Ausgewählte Elemente werden farbig hinterlegt dargestellt. Ein nochmaliger Mausklick hebt die Auswahl des Eintrags auf.
Select All	Wählt alle Elemente der Liste aus.
Deselect All	Deaktiviert die vorhandene Auswahl der Elemente.

DIGITAL CHANNELS

Parameter	Beschreibung
`[Liste der verfügbaren Kanäle]`	Auswahl der zu visualisierenden Elemente Mehrfachauswahl mit Klick auf den gewünschten Eintrag in der Liste. Ausgewählte Elemente werden farbig hinterlegt dargestellt. Ein nochmaliger Mausklick hebt die Auswahl des Eintrags auf.
Select All	Wählt alle Elemente der Liste aus.
Deselect All	Deaktiviert die vorhandene Auswahl der Elemente.

AUSWAHL ANZEIGEN

Um Ihre Auswahl in der Graphen Ansicht anzuzeigen, klicken Sie auf den Button **Apply**.

Hinweis: Ein Klick auf die seitlich vertikal angeordnete Umschaltung **COMTRADE Viewer Configuration** wechselt nur die Ansicht. Eine geänderte Auswahl der Kanäle wird dabei nicht berücksichtigt.

Runtime Ansicht - Visualisierung von COMTRADE Daten

In der Graphen Ansicht des **COMTRADE-Viewer** WPF-Elements werden die gewählten Kanäle visualisiert. Die Farbgebung kann im zenon Editor projektiert werden.

EXPORT DER AUSGEWÄHLTEN DATEN

Die ausgewählten analogen und digitalen Kanäle können mit dem Button **CSV-Export** in eine CSV-Datei exportiert werden.

GRAPHEN ANSICHT

Die Graphen Ansicht des **COMTRADE-Viewers** ist in drei Abschnitte aufgeteilt:

- Stromstärke
  Oberer Bereich
- Spannung
  Mittlerer Bereich
- Digitale Kanäle
  Unterer Bereich

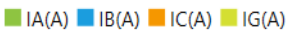
ACHSENBESCHRIFTUNG

- Horizontale Achse
  Die horizontale Achse repräsentiert den gesamten Zeitraum, wie in der COMTRADE-Datei

(`*.dat`) aufgezeichnet.
Die Skalierung dieser Zeitachse ist abhängig von der Vergrößerungsstufe. Je größer die Vergrößerung gewählt ist, umso detaillierter wird auch die Zeitdarstellung.

- Vertikale Achse
  Die vertikale Achse repräsentiert die Werte.
  - Die Skalierung der Werteachse ist abhängig von der Vergrößerungsstufe. Je größer die Vergrößerung gewählt ist, umso detaillierter wird auch die Wertedarstellung.
  - Die Bezeichnung der analogen Kanäle wird vertikal bei den Werten angezeigt und entspricht der Maßeinheit, wie in der COMTRADE-Datei (`*.cfg`) definiert.
  - Die Darstellung der digitalen Kanäle erfolgt in der Reihenfolge, wie in der COMTRADE-Datei (`*.cfg`) definiert.
    Als Bezeichner dient der `Channel identifier` der COMTRADE-Datei.

LEGENDE



Die Farblegende der Graphen wird am Kopf des Graphen angezeigt.

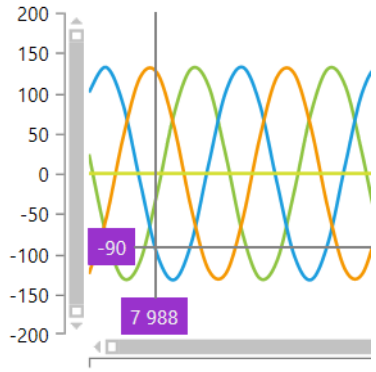
- Die Bezeichnung der digitalen Kanäle entspricht der Kanalbezeichnung, wie in der COMTRADE-Datei (`*.cfg`) definiert.
- Die Farbzuordnung pro Kanal erfolgt automatisch mit der projektierten Farbpalette.
- Die Zeit wird in einer Fußleiste unter dem Graphen angezeigt. Die Startzeit wird als Text angezeigt.

NAVIGATION UND ZOOM

Die Navigation (Scrollen und Zoomen) wird immer auf alle drei Bereiche der grafischen Darstellung angewendet.

- Mit der Bildlaufleiste verschieben Sie die Darstellung innerhalb der horizontalen Zeitlinie.
- Zoom-In und Zoom-Out
  - Mit dem Mausrad zoomen Sie an der aktuellen Position des Mauszeigers in die Graphen Ansicht oder verkleinern die Vergrößerung.
  - Mit Auswahl eines Anzeigebereichs bei gedrückter Maustaste wird der ausgewählte Bereich angezeigt.
    Hinweis: Die Anzeige der Werte wird immer dem gewählten Bereich angepasst. Dadurch kann es zu einer Abflachung der Kurve in der vergrößerten Graphen Ansicht kommen.
  - Ein Doppelklick auf die Bildlaufleiste setzt die Vergrößerung zurück.

ANALYSE



Die exakten Werte an der Position des Mauszeigers werden mit einer Darstellung in Werteblöcken visualisiert. Ein Fadenkreuz bietet eine zusätzlich optisch Unterstützung bei der exakten Bestimmung der Ableseposition.

Konfigurierbare Control-Eigenschaften - Farbdarstellung

PROJEKTIEREN IM EDITOR

Das Element mit dem Namen **COMTRADE.CDWPF** kann in jedem zenon Bildtyp projektiert und platziert werden.
Die Projektierung von **Breite [Pixel]** und **Höhe [Pixel]** des Elements sind proportional abhängig. Dadurch wird verhindert, dass der **COMTRADE-Viewer** verzerrt in der Runtime dargestellt wird.

Hinweis: Achten Sie bei der Projektierung auf eine ausreichende Größe, um die Übersichtlichkeit zu bewahren.

GRAFISCHE ANPASSUNGEN

Die grafische Ausprägung konfigurieren Sie in den Eigenschaften des WPF-Elements.
Weitere Informationen dazu finden Sie im Kapitel Konfiguration der Verknüpfung (auf Seite 123) in diesem Handbuch.

Mögliche Farbwerte:

- Hexadezimale Farbwerte
  `#RRGGBB`
  Beispiel-Farbwerte: #000000 = schwarz, #FFFFFF = weiß, #FF0000 = rot
- Farbwerte per Namen
  Referenz: https://msdn.microsoft.com/en-us/library/system.drawing.color.aspx
  (https://msdn.microsoft.com/en-us/library/system.drawing.color.aspx)

> **Tipp**
>
> Die Eigenschaften für das **COMTRADE-Viewer** WPF-Elements haben als Anfangsbuchstaben ein "z". Nutzen Sie für eine übersichtliche Darstellung bei der Konfiguration der Verknüpfung die Namensfilterung.

KONFIGURATIONSSEITE

Text- und Hintergrundfarbe der Konfigurationsseite.

Analog Channels

Parameter	Beschreibung	Wert
zConfiguratinPageTextColor	Textfarbe der Konfigurationsseite	String
zConfigurationPageBackgroundColor	Hintergrundfarbe der Konfigurationsseite	String

BUTTONS

Text- und Hintergrundfarbe der Buttons.

Open...

Parameter	Beschreibung	Wert
zButtonTextColor	Textfarbe der Buttons	String
zButtonBackgroundColor	Hintergrundfarbe der Buttons	String

CHART

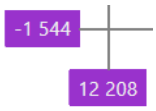
Textfarbe der Achsenbeschriftung oder Legende und Hintergrundfarbe.

50000 60000 70000

Parameter	Beschreibung	Wert
zChartTextColor	Textfarbe der Achsenbeschriftung.	`String`
zChartBackgroundColor	Hintergrundfarbe der Achsenbeschriftung	`String`

LABEL

Text- und Hintergrundfarbe der Anzeige von Werten an einer gewählten Cursorposition.



Parameter	Beschreibung	Wert
zChartLabelTextColor	Textfarbe der Wertanzeige	`String`
zChartLabelBackgroundColor	Hintergrundfarbe der Wertanzeige	`String`

CHART

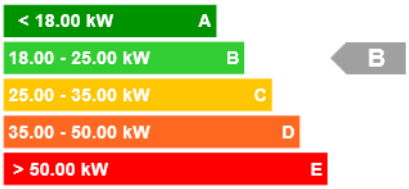
Farbpalette der Graphen Ansicht und der dazugehörigen Legenden.

Parameter	Beschreibung	Wert
zChartPalette	Farbpalette der Graphen- und Legendenfarben. Referenzierung mit Farbpalettenname (siehe Übersicht). Default: ist keine Farbpalette konfiguriert, wird die Farbpalette vom Betriebssystem des Rechners übernommen.	`String`

MÖGLICHE FARBPALETTEN - ÜBERSICHT

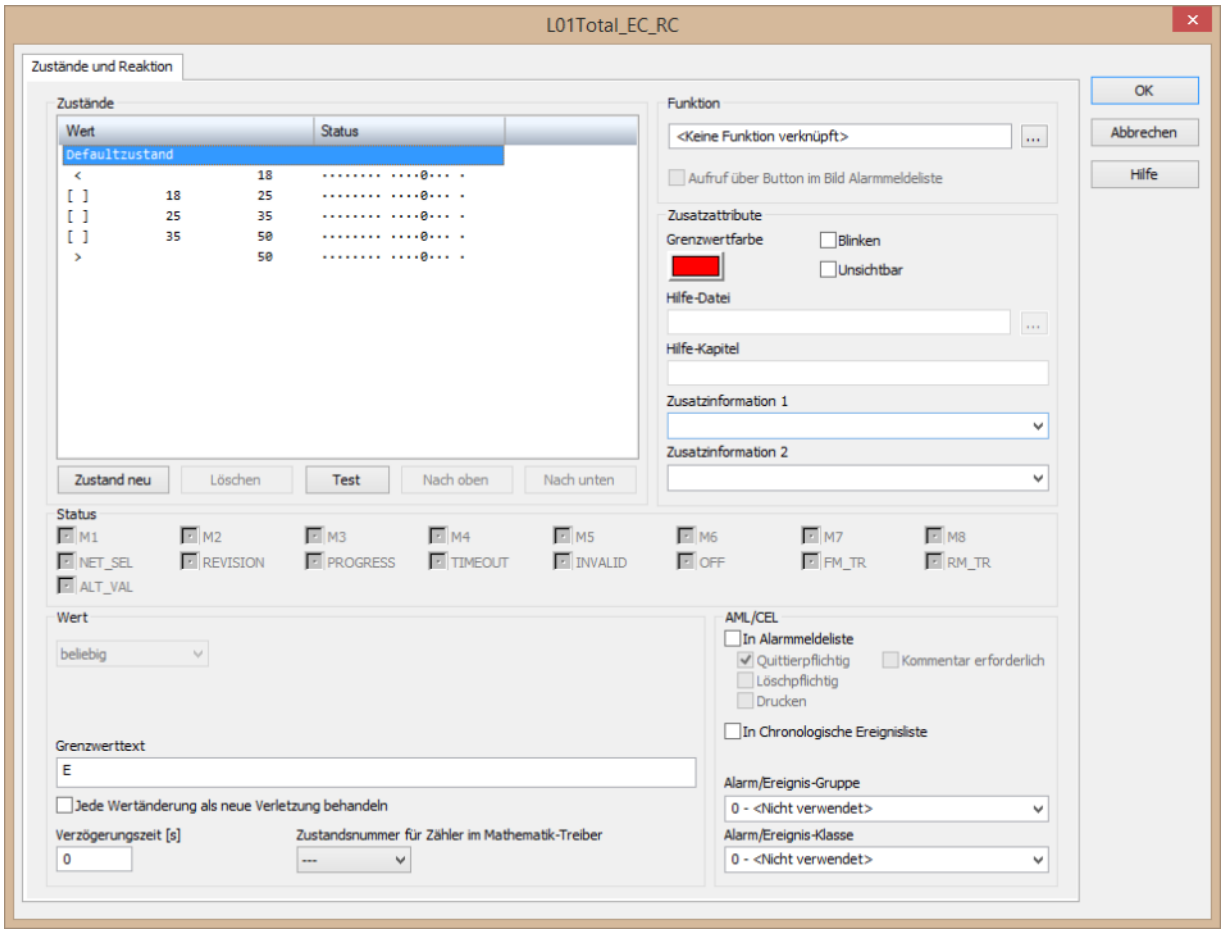
Energieklassen-Diagramm

Das Energieklassen-Diagramm, WPF-Element steht Partnern von COPA-DATA zur Verfügung und ist für diese über das Partner Portal erhältlich.



Zur Modellierung eines Energieklassen-Diagramms muss eine Reaktionsmatrix verwendet werden. Diese Reaktionsmatrix muss mit jener Variable verknüpft werden, deren Wert zur Anzeige und Einteilung in Energieklassen vorgesehen ist. Der Name der Variable muss dem Property "**zVariableName**" übergeben werden.

REAKTIONSMATRIX FÜR ENERGIEKLASSEN-DIAGRAMM



Die verknüpfte Reaktionsmatrix muss folgendem Schema entsprechen:

- Der erste Zustand muss ein Bereich, oder eine „kleiner als"- Definition sein
- Anschließend können beliebig viele Bereiche definiert werden.
- Der letzte Zustand muss ein Bereich, oder eine „größer als"- Definition sein.

Für die Projektierung gilt:

1. Sollten der erste Zustand ein Bereich sein und der Wert der Variable fällt unter diesen Bereich, so wird trotzdem der erste Zustand in dem Diagramm angezeigt. Das gleiche gilt für den letzten Zustand in umgekehrter Weise.
2. Die Farben, die das WPF-Diagramm für die Klassen verwendet, sind die Grenzwertfarben, die in der Reaktionsmatrix definiert wurden.
3. Die Buchstaben für die Klassen werden mit „A" beginnend in alphabetischer Reihenfolge gesetzt.

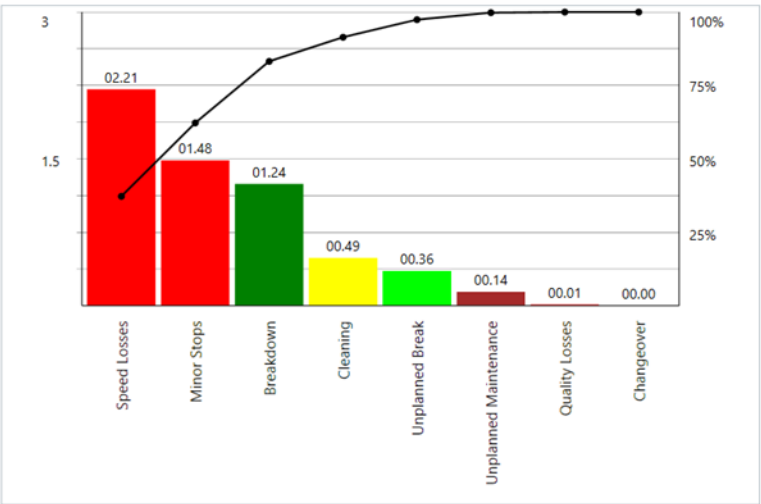
Property	Beschreibung	Wert
zenonFontID	ID für eine Schrift aus der ersten Schriftliste (Schriftgröße wird nicht berücksichtigt)	Integer
zenonNumberOfDecimalPlaces	Anzahl der angezeigten Nachkommastellen	Integer
zenonVariableName	Name der anzuzeigenden Variable	String

Hinweis: Für die Darstellung des Energieklassen-Diagramms im zenon Web Client sind zusätzliche VSTA-Programmierungen notwendig. Details dazu erhalten Sie im Kapitel Anzeige von WPF-Elementen im zenon Web Client (auf Seite 164).

Pareto-Diagramm

Das Pareto-Diagramm, WPF-Element steht Partnern von COPA-DATA zur Verfügung und ist für diese über das Partner Portal erhältlich.

Im Folgenden wird ein Beispiel eines Pareto-Diagrammes in der Runtime dargestellt:



Folgende Einstellungen können im WPF-Konfigurationsfenster unter **COPADATA-ELEMENT** gemacht werden:

Property	Funktion	Wert
zenonBarColor1	Farbe des 1. Balken	Color (String)
zenonBarColor2	Farbe des 2. Balken	Color (String)
zenonBarColor3	Farbe des 3. Balken	Color (String)
zenonBarColor4	Farbe des 4. Balken	Color (String)
zenonBarColor5	Farbe des 5. Balken	Color (String)
zenonBarColor6	Farbe des 6. Balken	Color (String)
zenonBarColor7	Farbe des 7. Balken	Color (String)
zenonBarColor8	Farbe des 8. Balken	Color (String)
zenonBarColor9	Farbe des 9. Balken	Color (String)
zenonBarColor10	Farbe des 10. Balken	Color (String)
zenonColorPercentageLine	Farbe der Prozentlinie (relative Summenhäufigkeit).	Color (String)
zenonLineVisibility	Sichtbarkeit der Prozentlinie (relative Summenhäufigkeit).	Boolean
zenonVariable1_Label	Beschriftung für den 1. Balken	String
zenonVariable1_Value	Wert des 1. Balken	Double
zenonVariable2_Label	Beschriftung für den 2. Balken	String
zenonVariable2_Value	Wert des 2. Balken	Double
zenonVariable3_Label	Beschriftung für den 3. Balken	String
zenonVariable3_Value	Wert des 3. Balken	Double
zenonVariable4_Label	Beschriftung für den 4. Balken	String
zenonVariable4_Value	Wert des 4. Balken	Double
zenonVariable5_Label	Beschriftung für den 5. Balken	String
zenonVariable5_Value	Wert des 5. Balken	Double
zenonVariable6_Label	Beschriftung für den 6. Balken	String
zenonVariable6_Value	Wert des 6. Balken	Double
zenonVariable7_Label	Beschriftung für den 7. Balken	String

zenonVariable7_Value	Wert des 7. Balken	`Double`
zenonVariable8_Label	Beschriftung für den 8. Balken	`String`
zenonVariable8_Value	Wert des 8. Balken	`Double`
zenonVariable9_Label	Beschriftung für den 9. Balken	`String`
zenonVariable9_Value	Wert des 9. Balken	`Double`
zenonVariable10_Label	Beschriftung für den 10. Balken	`String`
zenonVariable10_Value	Wert des 10. Balken	`Double`

Folgende Events können verwendet und mit zenon Funktionen verknüpft werden:

Event	Funktion	Wert
zenonBar1Click	Funktion, die beim Klick auf den 1. Balken ausgeführt wird.	Funktion
zenonBar2Click	Funktion, die beim Klick auf den 2. Balken ausgeführt wird.	Funktion
zenonBar3Click	Funktion, die beim Klick auf den 3. Balken ausgeführt wird.	Funktion
zenonBar4Click	Funktion, die beim Klick auf den 4. Balken ausgeführt wird.	Funktion
zenonBar5Click	Funktion, die beim Klick auf den 5. Balken ausgeführt wird.	Funktion
zenonBar6Click	Funktion, die beim Klick auf den 6. Balken ausgeführt wird.	Funktion
zenonBar7Click	Funktion, die beim Klick auf den 7. Balken ausgeführt wird.	Funktion
zenonBar8Click	Funktion, die beim Klick auf den 8. Balken ausgeführt wird.	Funktion
zenonBar9Click	Funktion, die beim Klick auf den 9. Balken ausgeführt wird.	Funktion
zenonBar10Click	Funktion, die beim Klick auf den 10. Balken ausgeführt wird.	Funktion

Rundanzeige - CircularGaugeControl

Property	Funktion	Wert
CurrentValue	Aktueller Wert, der angezeigt werden soll.	Double
IsReversed	Skalenausrichtung, in oder gegen Uhrzeigersinn.	Boolean
ElementFontFamily	Elementschriftart.	Font
MinValue	Minimalwert der Skala.	Double
MaxValue	Maximalwert der Skala.	Double
ScaleRadius	Radius der Skala.	Double
ScaleStartAngle	Winkel, bei dem die Skala beginnt.	Double
ScaleLabelRotationMode	Ausrichtung der Skalenbeschriftung.	Enum: ▸ None ▸ Automatic ▸ SurroundIn ▸ SurroundOut
ScaleSweepAngle	Winkelbereich, der die Größe der Skala definiert.	Double
ScaleLabelFontSize	Schriftgröße der Skalenbeschriftung.	Double
ScaleLabelColor	Schriftfarbe der Skalenbeschriftung.	Color
ScaleLabelRadius	Radius, an dem die Skalenbeschriftung ausgerichtet ist.	Double
ScaleValuePrecision	Genauigkeit der Skalenbeschriftung.	Integer
PointerStyle	Form des Zeigers, der den Wert anzeigt.	Enum: ▸ Arrow ▸ Rectangle ▸ TriangleCap ▸ Pentagon ▸ Triangle
MajorTickColor	Farbe der Hauptticks der Skala.	Color
MinorTickColor	Farbe der Nebenticks der Skala.	Color
MajorTickSize	Größe der Hauptticks der Skala.	Size
MinorTickSize	Größe der Nebenticks der Skala.	Size
MajorTicksCount	Anzahl der Hauptticks der Skala.	Integer
MajorTicksShape	Form/Art der Hauptticks der Skala.	Enum: ▸ Rectangle

		▸ Trapezoid ▸ Triangle

MinorTicksShape	Form/Art der Nebenticks der Skala.	Enum: ▸ Rectangle ▸ Trapezoid ▸ Triangle
MinorTicksCount	Anzahl der Nebenticks der Skala.	Integer
PointerSize	Größe des Zeigers.	Size
PointerCapRadius	Größe des Zeigerbefestigungspunkts.	Double
PointerBorderBrush	Farbe des Zeigerrahmens.	Brush
PointerCapStyle	Form/Art des Zeigerbefestigungspunkts.	Enum: ▸ BackCap ▸ FrontCap ▸ Screw
PointerCapBorderBrush	Farbe des Zeigerbefestigungspunkts.	Brush
PointerBrush	Farbe des Zeigers.	Brush
GaugeBorderBrush	Farbe des Elementrahmens.	Brush
GaugeBackgroundBrush	Farbe des Elementhintergrunds.	Brush
PointerCapColorBrush	Farbe des Zeigerbefestigungspunkts.	Brush
GaugeMiddlePlate	Radius der Elementhintergrundmittelplatte.	Double
PointerOffset	Offset des Zeigers (Verschiebung).	Double
RangeRadius	Radius der Bereichsgesamtanzeige.	Double
RangeThickness	Dicke der Bereichsgesamtanzeige.	Double
RangeStartValue	Startwert der Bereichsgesamtanzeige.	Double
Range1EndValue	Endwert des 1. Bereichs und Startwert des 2. Bereichs.	Double
Range2EndValue	Endwert des 2. Bereichs und Startwert des 3. Bereichs.	Double
Range3EndValue	Endwert des 3. Bereichs und Startwert des 4. Bereichs.	Double
Range4EndValue	Endwert des 4. Bereichs und Startwert des 5. Bereichs.	Double
Range5EndValue	Endwert des 5. Bereichs und Startwert des 6. Bereichs.	Double
Range6EndValue	Endwert des 6. Bereichs.	Double
Range1ColorBrush	Farbe des 1. Bereichs.	Brush
Range2ColorBrush	Farbe des 2. Bereichs.	Brush
Range3ColorBrush	Farbe des 3. Bereichs.	Brush
Range4ColorBrush	Farbe des 4. Bereichs.	Brush
Range5ColorBrush	Farbe des 5. Bereichs.	Brush
Range6ColorBrush	Farbe des 6. Bereichs.	Brush

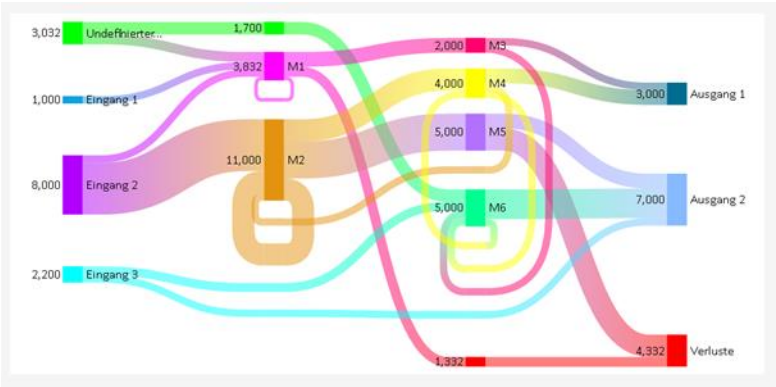
ScaleOuterBorderBrush	Farbe der Skalenumrandung.	`Brush`
ScaleBackgroundBrush	Farbe des Skalahintergrunds.	`Brush`
ValueTextFrameStyle	Form/Art der Wertanzeige.	`Enum:` ▸ `LargeFrame` ▸ `SmallFrame` ▸ `None`
ValueTextContent	Inhalt der Wertanzeige.	`Enum:` ▸ `Text` ▸ `TextValue` ▸ `Value`
ValueTextSize	Schriftgröße der Werteanzeige.	`Double`
ValueTextColor	Schriftfarbe der Werteanzeige.	`Color`
IsGlasReflection	Aktivierung des Glaseffektes auf dem Element.	`Boolean`
GaugeOffsett	Absenkung des Rotationspunktes des gesamten Elements.	`Double`

Sankey-Diagramm

Das Sankey-Diagramm, WPF-Element steht Partnern von COPA-DATA zur Verfügung und ist für diese über das Partner Portal erhältlich.

Zur Modellierung eines Sankey-Diagrammes muss der Sankey Wizard verwendet werden. Der Wizard erstellt eine XML-Datei, die dann vom WPF-Element ausgewertet wird. Dazu muss die Eigenschaft **zSankeyName** mit dem Namen der XML-Datei versehen werden. Die XML-Datei muss in dem Ordner `Sonstiges` eines Projektes im Arbeitsbereich liegen. Diese wird vom Wizard dorthin gespeichert.

Im Folgenden wird ein Beispiel eines Sankey-Diagrammes in der Runtime dargestellt:



Folgende Einstellungen können im WPF-Konfigurationsfenster unter **COPADATA-ELEMENT** gemacht werden:

Property	Funktion	Wert
FontSize	Schriftgröße von den Texten.	`Integer`
zBackgroundColor	Hintergrundfarbe des Diagrammes.	`Color (String)`
zFontColor	Farbe der Texte.	`Color (String)`
zFontFamily	Schriftart aller Texte.	`Font (String)`
zLossDetectionActive	Autom. Verlusterkennung aktiviert/deaktiviert. Wenn `true`, dann werden Verluste bei einem Knotenpunkt automatisch als Flüsse dargestellt.	`Boolean`
zNoDataText	Text der angezeigt wird, wenn keine Werte zum Darstellen vorhanden sind und **zPrevireActive** `false` ist.	`String`
zNoValidXMLText	Text der angezeigt wird, wenn keine gültige XML-Datei mit dem eingegebenen Namen gefunden wurde und **zPreviewActive** false ist.	`String`
zNumberOfDecimalPlaces	Gibt an, wie viele Nachkommastellen angezeigt werden.	`Integer`
zPreviewActive	Anzeigen eines Preview aktiviert/deaktiviert. Das Preview wird dann angezeigt, wenn keine Daten zum Anzeigen vorhanden sind (es wird das modellierte Diagramm mit Default-Werten besetzt) oder die XML-Datei nicht gefunden wurde oder diese keine gültige Definition enthält (es wird ein Beispiel-Sankey-Diagramm angezeigt).	`Boolean`
zRefreshRate	Rate der Aktualisierungen des Diagrammes in ms.	`Integer`
zSankeyName	Name der XML-Datei mit der Modellierung des Diagrammes.	`String`
zShowRelativeValues	Anzeige der Werte in absoluten `false` oder relativen Werten `true`.	`Boolean`

Hinweis: Für die Darstellung des Sankey-Diagramms im zenon Web Client sind zusätzliche VSTA-Programmierungen notwendig. Details dazu erhalten Sie im Kapitel Anzeige von WPF-Elementen im zenon Web Client (auf Seite 164).

Temperaturanzeige - TemperatureIndicatorControl

Property	Funktion	Wert
CurrentValue	Aktueller Wert, der angezeigt werden soll.	Double
MinValue	Minimalwert der Skala.	Double
MaxValue	Maximalwert der Skala.	Double
MajorTicksCount	Anzahl der Hauptticks der Skala.	Integer
MinorTicksCount	Anzahl der Nebenticks der Skala.	Integer
TickNegativColor	Farbe des negativen Hauptticks (verlaufend zu TickPositivColor).	Color
TickPositivColor	Farbe des positiven Hauptticks (verlaufend zu TickNegativColor).	Color
MinorTickColor	Farbe der Nebenticks.	Color
ElementBorderBrush	Farbe des Elementrahmens.	Brush
ElementBackgroundBrush	Farbe des Elementhintergrunds.	Brush
ElementGlasReflection	Aktivierung des Glaseffektes auf dem Element.	Visibility
ElementFontFamily	Elementschriftart.	Font
IndicatorColor	Farbe der Indikatorfüllfarbe.	Color
IndicatorBorderColor	Farbe des Indikatorrahmens.	Color
MajorTickSize	Größe der Hauptticks der Skala.	Size
MinorTickSize	Größe der Nebenticks der Skala.	Size
ScaleLetteringDistance	Abstand der Skalenbeschriftung (vertikal), jeder x. Haupttick soll beschriftet werden.	Integer
IndicatorScaleDistance	Abstand zwischen Indikator und Skala (horizontal).	Double
ScaleFontSize	Schriftgröße der Skala.	Double
ScaleFontColor	Schriftfarbe der Skala.	Color
Unit	Einheit.	String
ElementStyle	Form/Art des Elements.	Enum: ▸ SmallFrame ▸ Unit ▸ None

Universalregler - UniversalReglerControl

Property	Funktion	Wert
CurrentValue	Aktueller Wert, der angezeigt werden soll.	Double
ElementFontFamily	Elementschriftart.	Font
MinValue	Minimalwert der Skala.	Double
MaxValue	Maximalwert der Skala.	Double
Radius		Double
ScaleRadius	Radius der Skala.	Double
ScaleStartAngle	Winkel, bei dem die Skala beginnt.	Double
ScaleLabelRotationMode	Ausrichtung der Skalenbeschriftung.	Enum: ▸ None ▸ Automatic ▸ SurroundIn ▸ SurroundOut
ScaleSweepAngle	Winkelbereich, der die Größe der Skala definiert.	Double
ScaleLabelFontSize	Schriftgröße der Skalenbeschriftung.	Double
ScaleLabelColor	Schriftfarbe der Skalenbeschriftung.	Color
ScaleLabelRadius	Radius, an dem die Skalenbeschriftung ausgerichtet ist.	Double
ScaleValuePrecision	Genauigkeit der Skalenbeschriftung.	Integer
ElementStyle	Darstellungsart des Elements	Enum: ▸ Knob ▸ Plate ▸ None
MajorTickColor	Farbe der Hauptticks der Skala.	Color
MinorTickColor	Farbe der Nebenticks der Skala.	Color
MajorTickSize	Größe der Hauptticks der Skala.	Size
MinorTickSize	Größe der Nebenticks der Skala.	Size
MajorTicksCount	Anzahl der Hauptticks der Skala.	Integer
MajorTicksShape	Form/Art der Hauptticks der Skala.	Enum: ▸ Rectangle ▸ Trapezoid ▸ Triangle

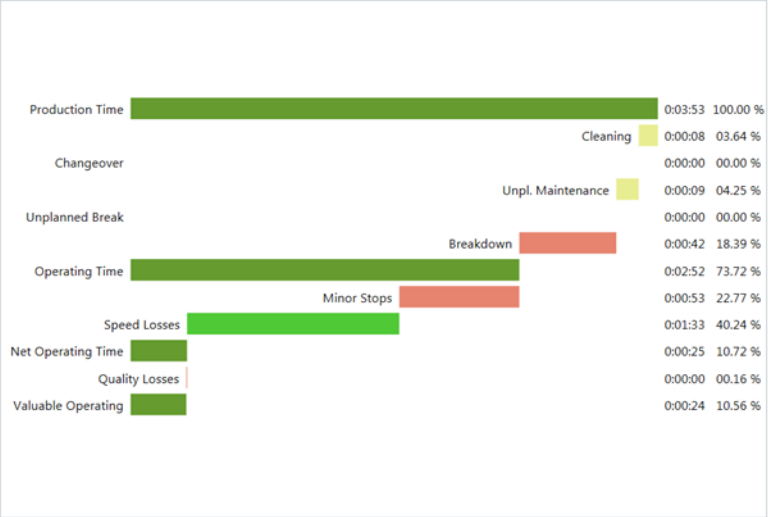
MinorTicksShape	Form/Art der Nebenticks der Skala.	Enum: ▸ Rectangle ▸ Trapezoid ▸ Triangle
MinorTicksCount	Anzahl der Nebenticks der Skala.	Integer
BackgroundBorderBrush	Farbe des Elementrahmens.	Brush
BackgroundBrush	Farbe des Elementhintergrunds.	Brush
PointerCapColorBrush	Farbe des Zeigerbefestigungspunkts.	Brush
GaugeMiddlePlate	Radius der Elementhintergrundmittelplatte.	Double
ValueFontSize	Schriftgröße der Werteanzeige.	Double
ValueFontColor	Schriftfarbe der Werteanzeige.	Color
IsGlasReflection	Aktivierung des Glaseffektes auf dem Element.	Boolean
KnobBrush	Farbe des Drehknopfes.	Brush
IndicatorBrush	Farbe des Indikators.	Brush
IndicatorBackgroundBrush	Hintergrundfarbe des unaktiven Indikators.	Brush
KnobSize	Durchmesser des Drehknopfes.	Double
KnobIndicatorSize	Indikatorgröße des Knopfes.	Size
ElementSize	Größe des Elements.	Size
VisibilityKnob	Aktivierung des Knopfes.	Boolean
ValuePosition	Position der Wertanzeige.	Double
ValueVisibility	Aktivierung der Werteanzeige.	Boolean

Wasserfall-Diagramm

Das Wasserfall-Diagramm, WPF-Element steht Partnern von COPA-DATA zur Verfügung und ist für diese über das Partner Portal erhältlich.

Zur Modellierung eines Wasserfall-Diagrammes muss der Meaning and Waterfall Chart Wizard verwendet werden. Mit diesem Wizard kann ein Wasserfall modelliert werden. Die Information wird direkt bei den betroffenen Variablen in der Eigenschaft **Analyzer --> Parameter for waterfall diagram** gespeichert.

Im Folgenden wird ein Beispiel eines Wasserfall-Diagrammes in der Runtime dargestellt:



Hinweis: Dieser Screenshot steht nur in englischer Sprache zur Verfügung.

Folgende Einstellungen können im WPF-Konfigurationsfenster unter **COPADATA-ELEMENT** gemacht werden:

Property	Funktion	Wert
zenonRefreshRate	Zeit zwischen den Aktualisierungen des Diagrammes in ms.	`Integer`
zenonWaterfallIdentifier	Name des Wasserfalldiagrammes.	`String`
zenonZSystemModel	Anlagengruppe der verwendeten Variablen.	`String`

Hinweis: Für die Darstellung des Wasserfall-Diagramms im zenon Web Client sind zusätzliche VSTA-Programmierungen notwendig. Details dazu erhalten Sie im Kapitel Anzeige von WPF-Elementen im zenon Web Client (auf Seite 164).

6.4.6 Anzeige von WPF-Elementen im zenon Web Client

Um die vorgefertigten WPF-Elemente **„Energieklassen-Diagramm"**, **„Sankey-Diagramm"** und **„Wasserfall-Diagramm"** auch für die Anzeige in einem zenon Web Client verwenden zu können, sind Anpassungen im Projekt notwendig:

- Projektierung in zenon Editor (auf Seite 165)
- VSTA-Code anpassen (auf Seite 165)

Projektierung im zenon Editor

Führen Sie im zenon Editor folgende Projektierungsschritte aus, um bestimmte WPF-Elemente auch im zenon Web Client anzeigen zu können:

WPF IN ZENON BILD PLATZIEREN:

- Platzieren Sie das Element WPF in einem zenon Bild.
- Vergeben Sie in der Eigenschaft **Elementname** einen eindeutigen Namen.
  Sie finden diese Eigenschaft in der Eigenschaftengruppe **Allgemein**.
  Hinweis: Wurde der Name für ein Element im Bild bereits in einem anderen Element vergeben, erscheint ein Warndialog.
- Verwenden Sie den hier vergebenen Elementnamen im VSTA-Code.

VSTA-Code (komplex)

Um den Programmiercode für die Darstellung von WPF-Elementen im zenon Web Client einzufügen, führen Sie die folgenden Schritte durch:

1. Wechseln Sie im zenon Editor in den Knoten **Programmierschnittstellen**.
2. Wählen Sie den Knoten **VSTA** und wählen Sie mit rechtem Mausklick den Kontextmenüeintrag **VSTA-Editor mit ProjectAddin öffnen...**
3. Der Dialog zum Anlegen eines VSTA-Projektes wird geöffnet.
4. Wählen Sie im Dialog **Neues VSTA-Projekt anlegen** den Eintrag C#.
5. Erstellen (kopieren) Sie den untenstehenden Code.
6. Tragen Sie den Namen des WPF-Elements im Code ein.

Hinweis: Achten Sie beim Öffnen des VSTA-Editors darauf, ob bereits Inhalte des folgenden Codes in der Projektierung vorhanden sind. Für die Anzeige des WPF-Elements im Web Client vergleichen Sie den bestehenden Code und nehmen Sie die notwendigen Ergänzungen vor. Bitte beachten Sie dazu die Kommentare im Muster-Code.

VSTA-CODE

```
//Als Member:
zenOn.IDynPictures zScreens = null;
string[] WPFElements ={"WPF_Control", "WPFWebclient_1", "WPFWebclient_2" }; //Namen der WPF-Bildelemente die in dem zenon Projekt vorkommen und die Zugriff auf die API brauchen (beliebig viele/wenige)
```

```
//In der Project-Active Funktion folgende drei Code-Zeilen hinzufügen:
void ThisProject_Active()
{
  zScreens = this.DynPictures();
  zScreens.Open += new zenOn.DDynPicturesEvents_OpenEventHandler(zScreens_Open);
  zScreens.Close += new zenOn.DDynPicturesEvents_CloseEventHandler(zScreens_Close);
}
//In der Project-InactiveFunktion folgende zwei Code-Zeilen hinzufügen:
void ThisProject_Inactive()
{
  zScreens.Open -= new zenOn.DDynPicturesEvents_OpenEventHandler(zScreens_Open);
  zScreens.Close -= new zenOn.DDynPicturesEvents_CloseEventHandler(zScreens_Close);

  //Final release and garbage collection of any API-Objects.
  FreeObjects();
}

//Zwei neue Event-Handler einfügen:
void zScreens_Open(zenOn.IDynPicture obDynPicture)
{
  foreach (string element in WPFElements)
  {
    if (obDynPicture.Elements().Item(element) != null)
    {
      obDynPicture.Elements().Item(element).set_WPFProperty("ELEMENT",
"zenonVariableLink", this.Variables().Item(0));
    }
  }
}
void zScreens_Close(zenOn.IDynPicture obDynPicture)
{
  foreach (string element in WPFElements)
  {
    if (obDynPicture.Elements().Item(element) != null)
    {
      zenOn.IElement zWPFElement= obDynPicture.Elements().Item(element);
      zWPFElement.set_WPFProperty("ELEMENT", "zenonTrigger", true);
      zWPFElement = null;
```

```
    }
  }
}
```

VSTA-Code (vereinfacht)

Wird nur ein WPF-Element in einem zenon Bild verwendet, kann alternativ folgender, schlankerer Code verwendet werden. Dazu müssen die Namen des WPF-Elementes, und des Bildes in dem das Element verwendet wird, eingetragen werden. Dieser Code wird dann empfohlen, wenn pro Projekt nur eines der vorgefertigten WPF-Elemente verwendet wird.

VSTA-CODE:

```
zenOn.IDynPicture zScreen = null;
string wpfElement = "WPF_Control"; //Name des WPF-Elements im Bild
string wpfPicture = "@Details_Overview_Online"; //Name des zenon Bildes

//In der Project-Active Funktion ergänzen:
void ThisProject_Active()
{
  zScreen = this.DynPictures().Item(wpfPicture);
  zScreen.Open += new zenOn.OpenEventHandler(zScreen_Open);
  zScreen.Close += new zenOn.CloseEventHandler(zScreen_Close);
}

//In der Project-Inctive Funktion ergänzen:
void ThisProject_Inactive()
{
  zScreen.Open -= new zenOn.OpenEventHandler(zScreen_Open);
  zScreen.Close -= new zenOn.CloseEventHandler(zScreen_Close);

  //Final release and garbage collection of any API-Objects.
  FreeObjects();
}

void zScreen_Open()
{
  if (zScreen.Elements().Item(wpfElement) != null)
  {
```

```
        zScreen.Elements().Item(wpfElement).set_WPFProperty("ELEMENT",
"zenonVariableLink", this.Variables().Item(0));
  }
}

void zScreen_Close()
{
  if (zScreen.Elements().Item(wpfElement) != null)
  {
        zenOn.IElement zWPFElement = zScreen.Elements().Item(wpfElement);
        zWPFElement.set_WPFProperty("ELEMENT", "zenonTrigger", true);
        zWPFElement = null;
  }
}
```

6.4.7 Beispiele: Integration von WPF in zenon

Wie XAML-Dateien erstellt und in zenon als WPF-Elemente eingebunden werden, sehen Sie an folgenden Beispielen:

- Button als WPF-XAML in zenon einbinden (auf Seite 174)
- Bargraf als WPF-XAML in zenon einbinden (auf Seite 168)
- DataGrid Control in zenon einbinden (auf Seite 181)

Bargraf als WPF-XAML in zenon einbinden

Aufbau des Beispiels:

- Erzeugung eines Bargrafs (auf Seite 89) in Adobe Illustrator und Umwandlung in WPF
- einbinden in zenon
- Verknüpfung mit Variablen
- anpassen des Bargraf-WPF-Elements

BARGRAF ERSTELLEN

Erzeugen Sie als ersten Schritt einen Bargraf wie im Kapitel Workflow mit Adobe Illustrator (auf Seite 89) beschrieben. Um die XAML-Datei in zenon verwenden zu können, fügen Sie diese im Projektbaum in den Ordner **Dateien/Grafiken** ein.

BARGRAF EINBINDEN

Hinweis: Für die folgende Beschreibung wird ein zenon Projekt mit folgenden Inhalten verwendet:

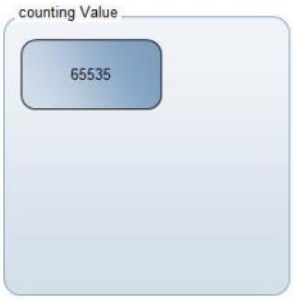
- ein leeres Bild als Startbild
- 4 Variablen vom Intern Treiber für
  - Skala `0`
  - Skala `Mitte`
  - Skala `hoch`
  - aktueller Wert
- eine Variable vom Mathematik Treiber für die Anzeige des aktuellen Werts (`255`)

Um den Bargraf einzubinden:

1. öffnen Sie das leere Bild
2. platzieren Sie ein **WPF-Element** (auf Seite 122) im Bild
3. wählen Sie im Eigenschaftenfenster **XAML-Datei**
4. wählen Sie die gewünschte XAML-Datei aus (z. B. **bargraph_vertical.xaml**) und schließen Sie den Dialog

BARGRAF ANPASSEN

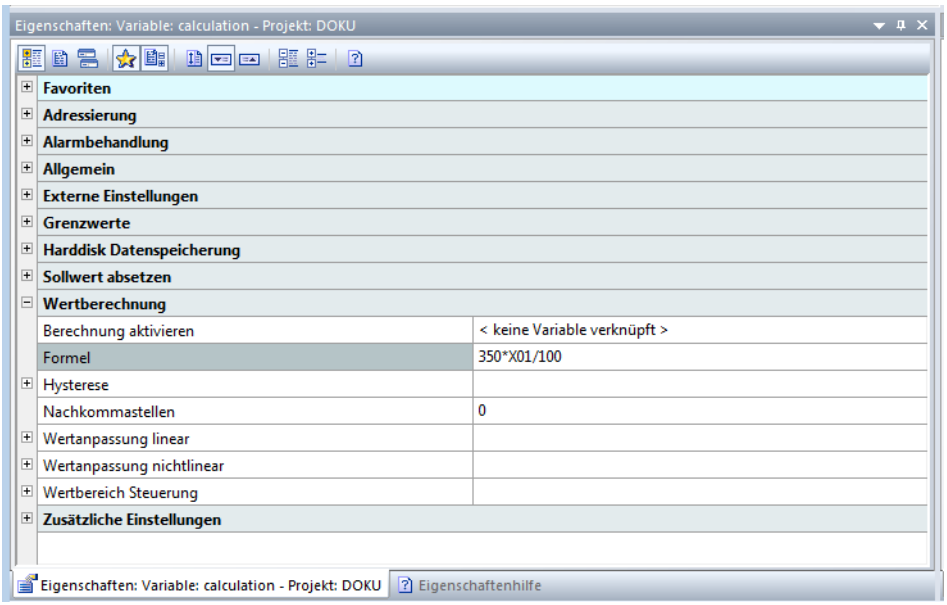
Vor der Konfiguration wird die Skala der XAML-Datei, wenn nötig, angepasst:



Dazu:

- legen Sie eine neue Mathematik-Variable an, die den neuen Wert in Bezug auf die Skalierung berechnet, zum Beispiel:
- Variable: 0-1000

- Mathematikvariable {im xaml-File erstellter Wert}*Variable/1000

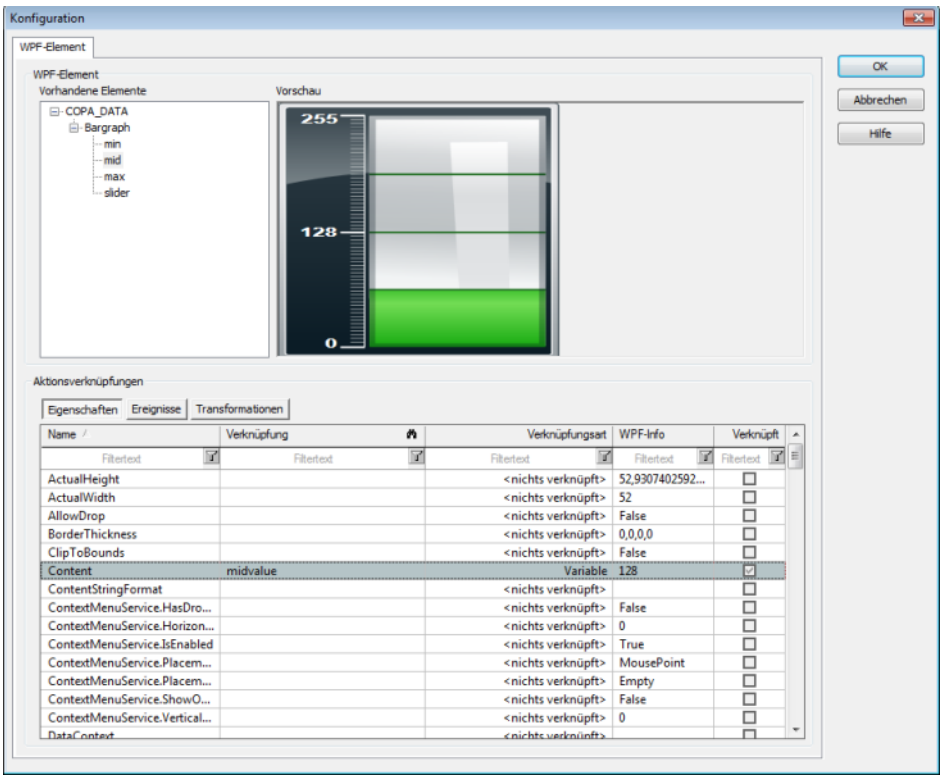


Dann wird die XAML-Datei konfiguriert.

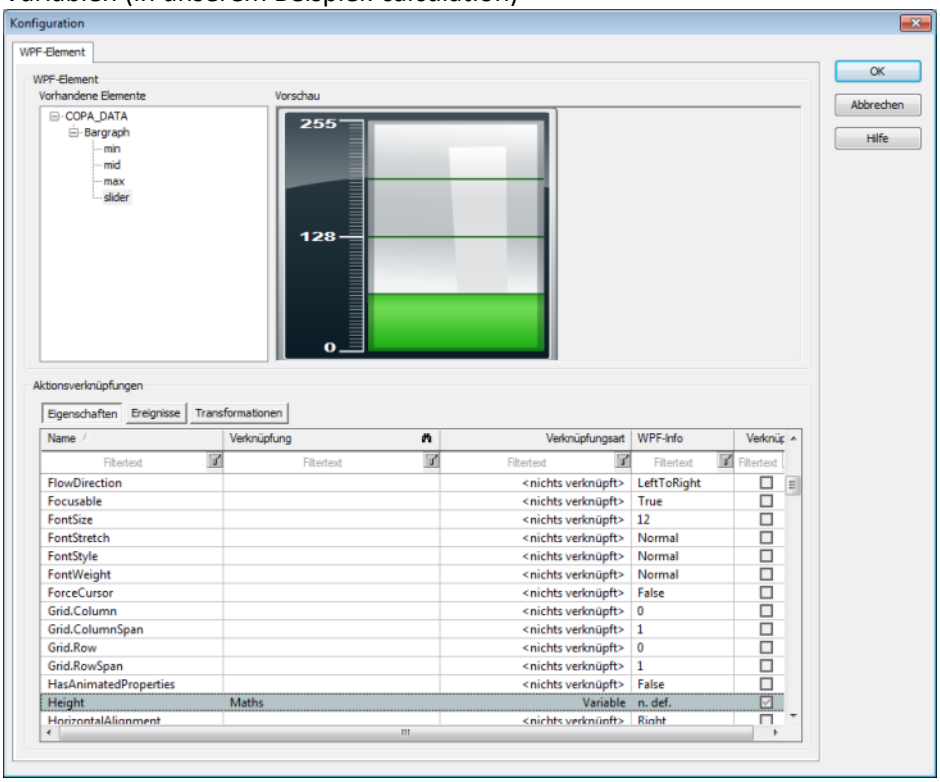
BARGRAF KONFIGURIEREN

1. klicken Sie auf das WPF-Element und wählen Sie die Eigenschaft **Konfiguration**
2. der Konfigurationsdialog zeigt eine Vorschau der ausgewählten XAML-Datei

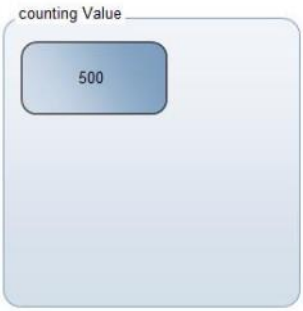
3. wählen Sie den Minimum-Wert, den Mittelwert und der Maximum-Wert aus und verknüpfen Sie jeweils in der Eigenschaft **Content** die entsprechende Variable



4. wählen Sie den **Slider** aus und verknüpfen Sie die Eigenschaft **Value** mit der Mathematik Variablen (in unserem Beispiel: calculation)



5. Überprüfen Sie die Projektierung in der Runtime:



Button als WPF-XAML in zenon einbinden

Aufbau des Beispiels:

- Erzeugung eines Buttons (auf Seite 85) in Microsoft Expression Blend
- einbinden in zenon
- Verknüpfung mit einer Variablen sowie einer Funktion
- anpassen des Buttons an die Größe des Elements
- Button erstellen

Erzeugen Sie als ersten Schritt einen Button wie im Kapitel Button als XAML-Datei mit Microsoft Expression Blend erstellen (auf Seite 85) beschrieben. Um die XAML-Datei in zenon verwenden zu können, fügen Sie diese im Projektbaum in den Ordner **Dateien**/**Grafiken** ein.

BUTTON EINBINDEN

Hinweis: Für die folgende Beschreibung wird ein zenon Projekt mit folgenden Inhalten verwendet:

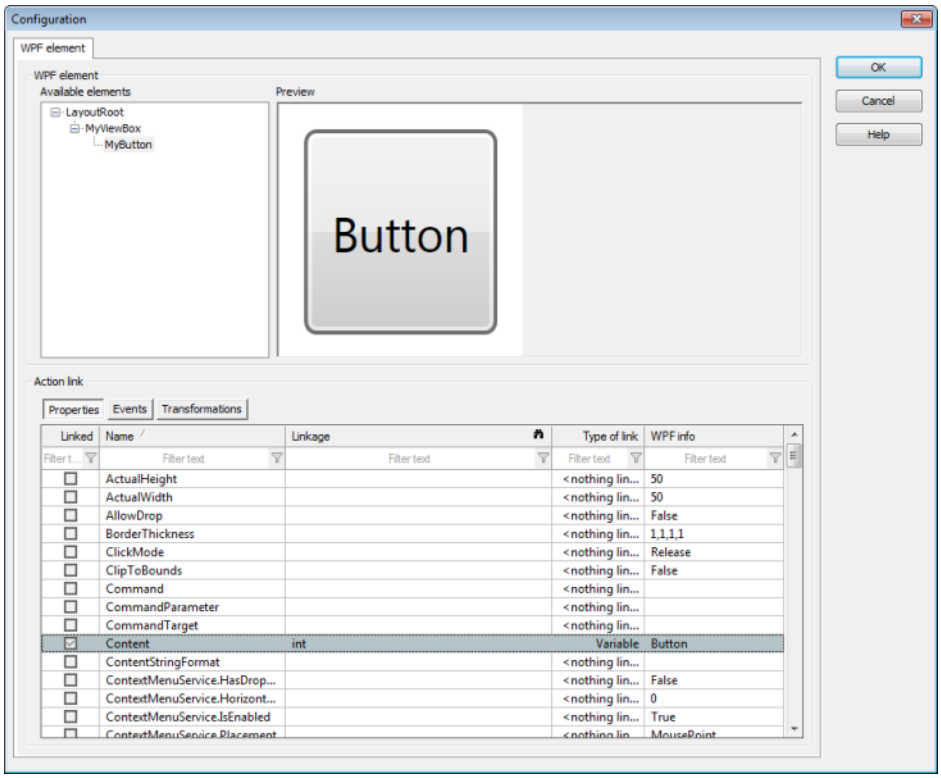
- ein leeres Bild als Startbild
- eine Intern-Variable **int** vom Typ **Int**
- eine Funktion **Funktion_0** vom Typ **Sollwert absetzen** mit:
  - Option **Direkt zu Hardware** aktiviert
  - Sollwert wurde auf 45

Um den Button einzubinden:

1. öffnen Sie das leere Bild
2. platzieren Sie ein **WPF-Element** (auf Seite 122) im Bild
3. wählen Sie im Eigenschaftenfenster **XAML-Datei**
4. wählen Sie die XAML-Datei aus (z. B.: **MyButton.xaml)** und schließen Sie den Dialog
5. wählen Sie die Eigenschaft **Konfiguration**

BUTTON KONFIGURIEREN

Der Konfigurationsdialog zeigt eine Vorschau der ausgewählten XAML-Datei. Im Baum werden alle in der XAML_Datei benannten Elemente aufgelistet:

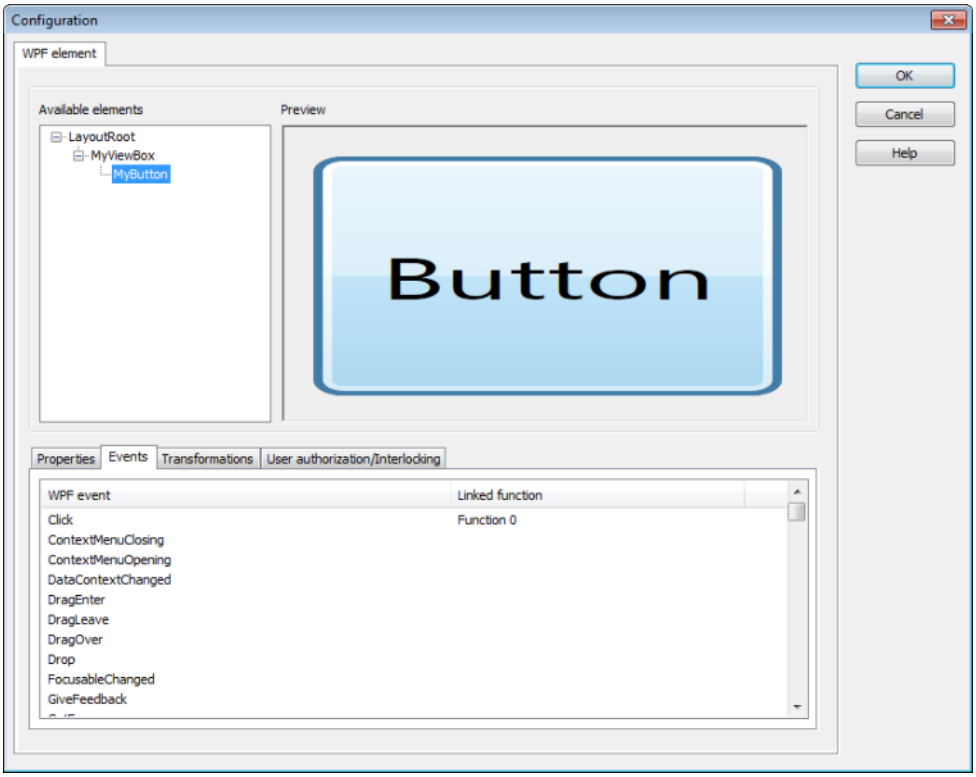


1. wählen Sie den WPF-Button, der sich in LayoutRoot->MyViewBox->MyButton befindet
2. suchen Sie in der Registerkarte **Eigenschaften** den Eintrag **Content**, dieser beinhaltet den Text des Buttons
3. klicken Sie in die Spalte V**erknüpfungsart**
4. wählen Sie in der Dropdownliste den Eintrag **Variable** aus
5. klicken Sie in die Spalte **Verknüpfung**
6. der Variablenauswahldialog wird geöffnet
7. wählen Sie die Variable `int`, um diese Variable mit der Eigenschaft **Content** zu verknüpfen

EREIGNISSE

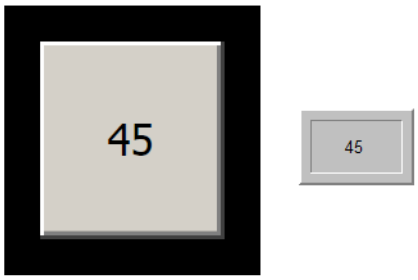
Um auch Ereignisse zuzuweisen:

1. selektieren Sie die Registerkarte Ereignisse



2. suchen Sie den Eintrag Click, dieses Ereignis wird von WPF-Element ausgelöst, sobald der Button geklickt wird
3. klicken Sie in die Spalte **Verknüpfungsart**
4. wählen Sie in der Dropdownliste den Eintrag **Funktion** aus
5. klicken Sie in die Spalte **Verknüpfung**
6. der **Funktionsauswahldialog** wird geöffnet
7. wählen Sie **Funktion_0** aus
8. bestätigen Sie die Änderungen mit **OK**
9. fügen Sie im Bild noch ein **Zahlenwert-Element** ein
10. verknüpfen Sie auch diesen **Zahlenwert** mit der Variablen `int`.
11. kompilieren Sie die Runtime-Dateien und starten Sie die Runtime

In der Runtime wird das **WPF-Element** dargestellt, der Text des Buttons ist 0. Sobald Sie auf den Button klicken, wird das **Click**-Ereignis getriggert und die Funktion **Sollwert setzen** ausgeführt. Der Wert 45 wird direkt auf die Hardware abgesetzt und sowohl **Zahlenwert** als auch **Button** zeigen den Wert 45 an.

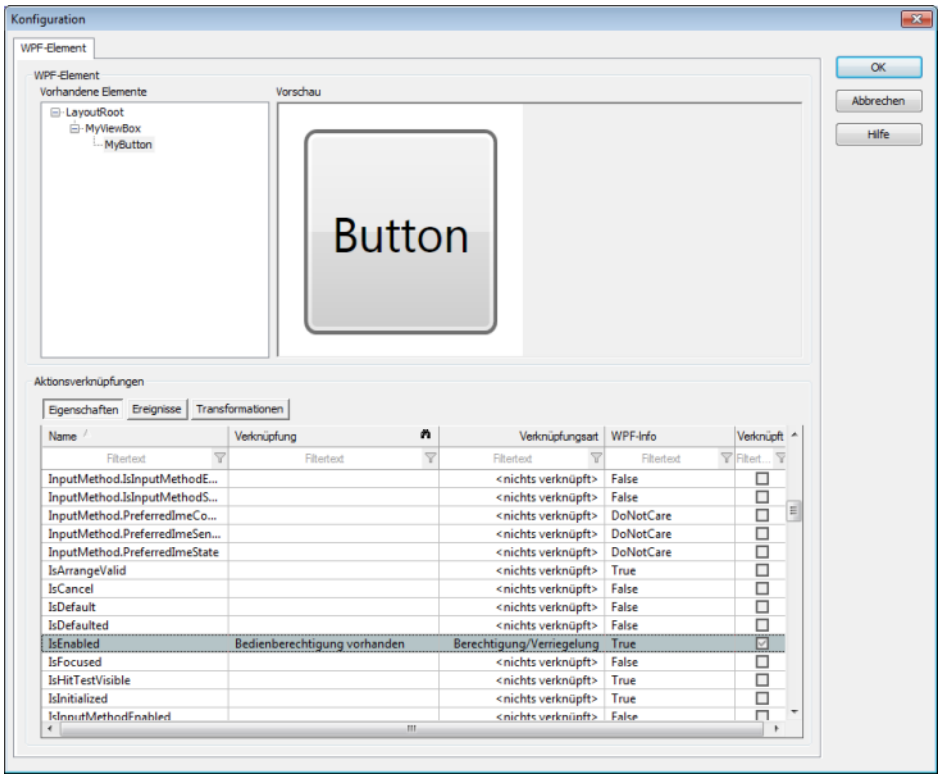


Setzen Sie über das Element **Zahlenwert** einen Sollwert von 30 ab, so wird auch dieser Wert vom **WPF-Element** übernommen.

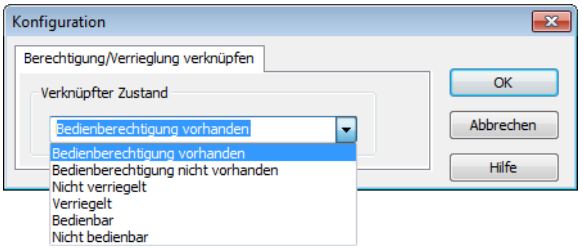
BEDIENBERECHTIGUNG

Ähnlich wie ein **Zahlenwert** kann ein **WPF**-Element je nach Bedienberechtigung gesperrt (Schlosssymbol) oder bedienbar geschaltet werden. Stellen Sie beim **WPF-Element** die Berechtigungsebene auf 1 und legen Sie einen Benutzer T**est** mit **Berechtigungsebene 1** an. Zusätzlich legen Sie die Funktionen **Login mit Dialog** und **Logout** an. Diese beiden Funktionen verknüpfen Sie mit 2 neuen Text-Buttons im Bild.

Im Konfigurationsdialog des **WPF-Elements** selektieren Sie den WPF-Button **MyButton** und wählen Sie die Registerkarte **Eigenschaften:**

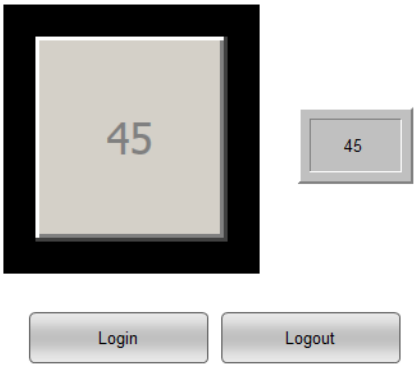


1. wählen Sie das Element **IsEnabled**
2. klicken Sie in die Spalte **Verknüpfungsart**
3. wählen Sie in der Dropdownliste den Eintrag **Bedienberechtigung/Verriegelung** aus
4. klicken Sie in die Spalte **Verknüpfung**
5. wählen Sie in der Dropdownliste die Option `Bedienberechtigung vorhanden` aus



6. schließen Sie den Dialog mit **OK**

Kompilieren Sie die Runtime-Dateien und achten Sie darauf, auch die Bedienberechtigungen zur Übertragung auszuwählen. Nach dem Starten der Runtime wird der WPF-Button im Bild deaktiviert angezeigt und kann nicht bedient werden. Wenn Sie sich jetzt mit dem Benutzer **Test** einloggen, wird der Button aktiviert und kann bedient werden. Sobald Sie sich ausloggen, wird der Button wieder gesperrt.



TRANSFORMATION

Um Transformationen verwenden zu können, muss die XAML-Datei noch angepasst werden:

1. wechseln Sie ins Programm **Expression Blend**
2. selektieren Sie **MyButton**, damit die Eigenschaften des Buttons im Eigenschaftenfenster sichtbar sind

3. selektieren Sie unter **Transform** bei **RenderTransform** die Option **Apply relative transform**

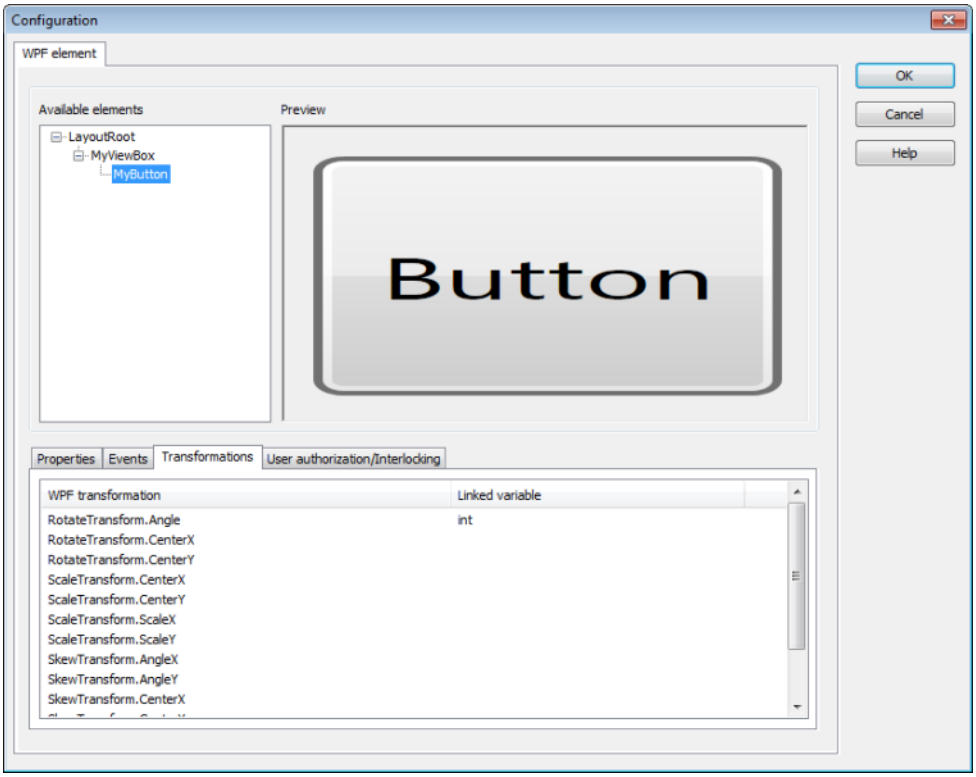
   Dadurch wird in der XAML-Datei ein Block eingefügt, der zur Laufzeit die Einstellungen zur Transformation speichert.

```
<Button.RenderTransform>
    <TransformGroup>
        <ScaleTransform ScaleX="1" ScaleY="1"/>
        <SkewTransform AngleX="0" AngleY="0"/>
        <RotateTransform Angle="0"/>
        <TranslateTransform X="0" Y="0"/>
    </TransformGroup>
</Button.RenderTransform>
```

4. speichern Sie die Datei und ersetzen Sie im zenon Projekt die alte Version mit dieser neuen Datei
5. öffnen Sie wieder den Konfigurationsdialog des **WPF-Elements**:

   a) wählen Sie den Button **MyButton** aus

b) wählen Sie die Registerkarte **Transformationen**



c) selektieren Sie das Element **RotateTransform.Angle**

d) klicken Sie in die Spalte **Verknüpfungsart**

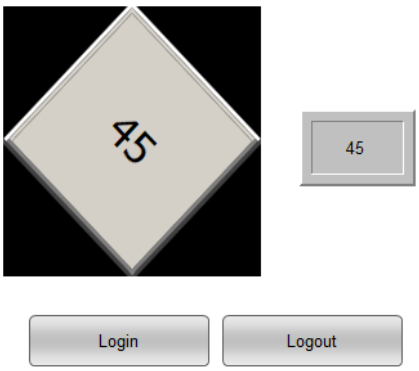
e) wählen Sie in der Dropdownliste den Eintrag **Transformationen** aus

f) klicken Sie in die Spalte **Verknüpfung**

g) der Variablenauswahldialog wird geöffnet

h) wählen Sie die Variable **int**, um diese Variable mit der Eigenschaft **RotateTransform.Angle** zu verknüpfen

Kompilieren Sie die Runtime-Dateien und starten Sie die Runtime. Loggen Sie sich als Benutzer **Test** ein und klicken Sie auf den Button. Der Button erhält den Wert `45` und das **WPF-Element** dreht sich um 45°.



DataGrid Control in zenon einbinden

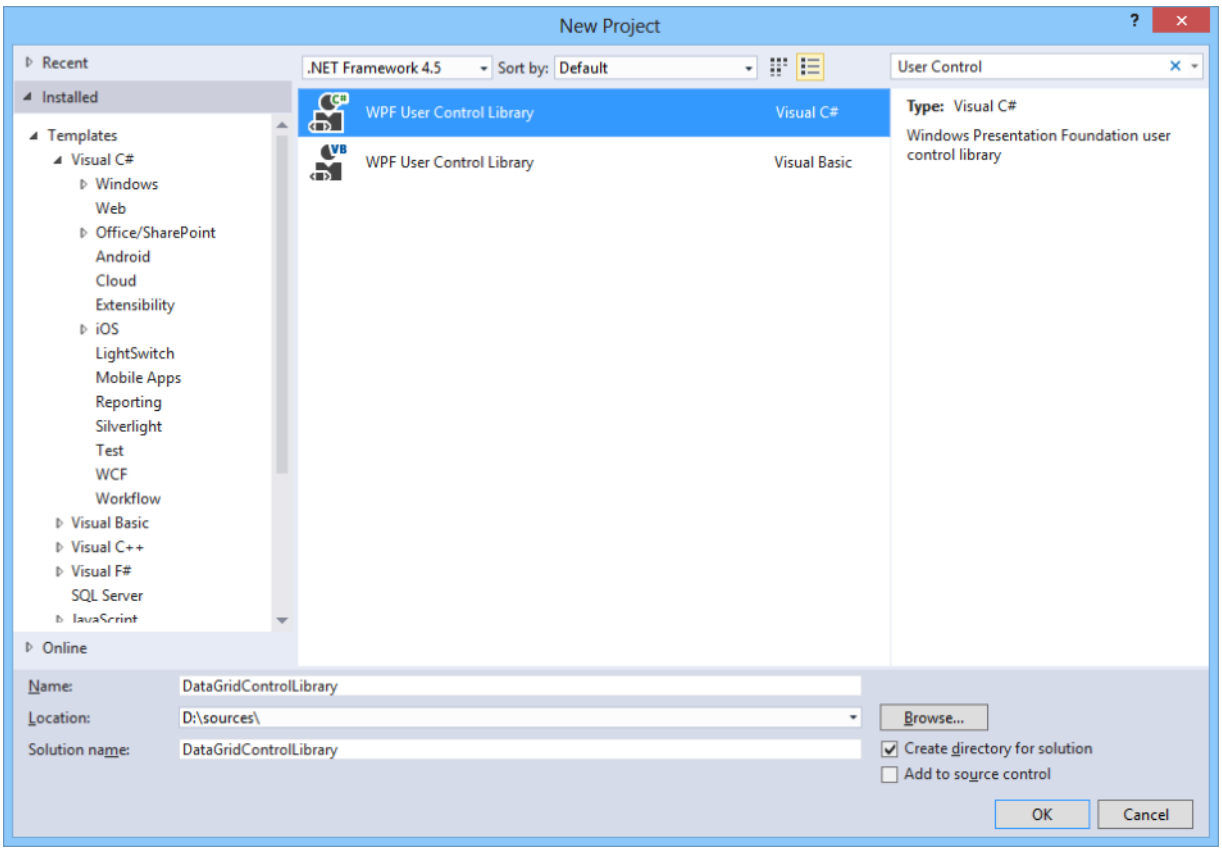
Um ein DataGrid Control für zenon zu erstellen, benötigen Sie:

- Visual Studio (in diesem Beispiel Visual Studio 2015)

WPF USER CONTROL ERSTELLEN

1. Erstellen Sie in Visual Studio eine neue **Solution** und darin ein **WPF User Control Library** Projekt in der .NET Framework Version 4 oder höher.

   Info: Erscheint die entsprechende Projektvorlage nicht in der Liste der verfügbaren Vorlagen, kann diese über die Suche (Feld rechts oben im Dialog) hinzugefügt werden.



   In unserem Beispiel erhält das Projekt den Namen **DataGridControlLibrary**.

2. Erstellen Sie eine neue Datenverbindung im **Server Explorer**.

   In unserem Beispiel wird die Datenbank `Northwind` verwendet, die von Microsoft als Beispieldatenbank zum freien Download zur Verfügung gestellt wird.

   Zum Anlegen der Datenbankverbindung:

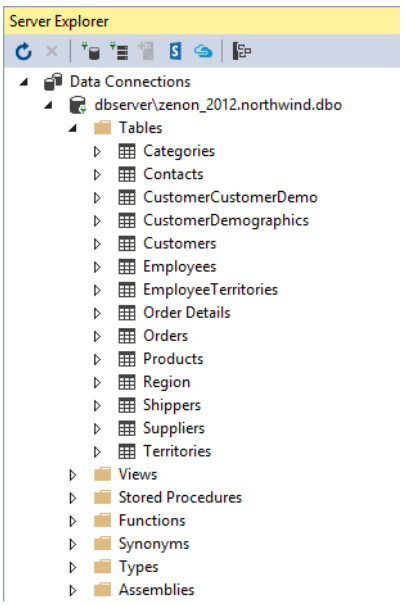
   a) Führen Sie einen Rechtsklick auf **Data Connections** aus.

b) Wählen Sie **Add connection...**.

c) wählen Sie `Microsoft SQL Server (SQLClient)` als **Data source**.

d) wählen Sie den entsprechenden Server- und Datenbanknamen aus.

Nach dem Hinzufügen der Verbindung, sollte das Server Explorer Fenster in etwa so aussehen:

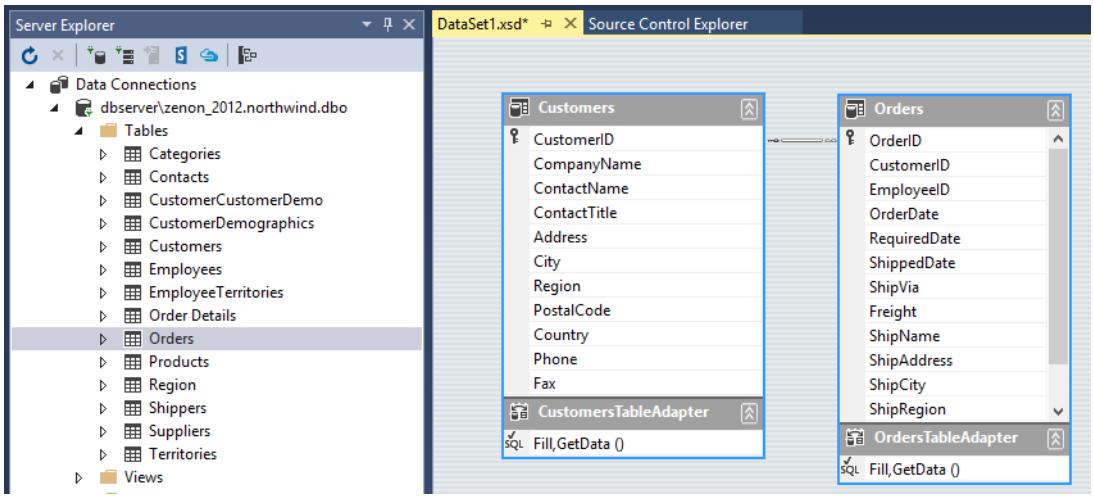


Im nächsten Schritt wird ein neues DataSet erstellt.

DATASET ERSTELLEN

1. Führen Sie einen Rechtsklick auf das Projekt aus.
2. Wählen Sie im Kontextmenü **Add – New Item...**.
3. Erstellen Sie ein neues **DataSet** mit dem Namen `DataSet1`.
4. Führen Sie einen Doppelklick auf das DataSet aus um es im Designer zu öffnen.

5. Ziehen Sie die Tabellen die Sie benötigen (in Diesem Beispiel `Customers` und `Orders`) , in das DataSet Design-Fenster.



Im nächsten Schritt wird die XAML-Datei modifiziert.

KONFIGURATION DER XAML-DATEI

1. Falls noch nicht vorhanden, fügen Sie den **Namespace** als Referenz auf die Klasse in die XAML Datei ein:

```
<UserControl x:Class="DataGridControlLibrary.UserControl1"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:DataGridControlLibrary"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
```

2. Definieren Sie die Ressourcen und das DataGrid, das im WPF verwendet werden soll:

```
<UserControl.Resources>
    <local:DataSet1 x:Key="DataSet1"/>
    <CollectionViewSource x:Key="CustomersViewSource" Source="{Binding Path=Customers,
    Source={StaticResource DataSet1}}"/>
</UserControl.Resources>
<Grid DataContext="{StaticResource CustomersViewSource}">
    <DataGrid Name="DataGrid1"  DisplayMemberPath="CompanyName"
    ItemsSource="{Binding}" SelectedValuePath="CustomerID"
    HorizontalAlignment="Stretch" VerticalAlignment="Stretch"/>
</Grid>
```

3. Öffnen Sie die Code-Behind Datei (**UserControl1.xaml.cs**) und fügen Sie im Konstruktor folgende Zeilen ein:

```
public UserControl1()
{
    InitializeComponent();
    DataSet1 ds = ((DataSet1)(FindResource("DataSet1")));
    DataSet1TableAdapters.CustomersTableAdapter ta = new
    DataSet1TableAdapters.CustomersTableAdapter();
    ta.Fill(ds.Customers);
    CollectionViewSource CustomersViewSource =
    ((CollectionViewSource)(this.FindResource("CustomersViewSource")));
    CustomersViewSource.View.MoveCurrentToFirst();
}
```

Dabei wird:

- Das DataSet geholt
- Ein neuer TableAdapter erstellt
- Das DataSet befüllt
- Die Informationen dem DataGrid Control zur Verfügung gestellt

Jetzt kann die Solution gebaut werden.

BUILD

Bauen Sie jetzt die Solution. Im Output-Ordner des Projekts wird nun die entsprechende DLL (**DataGridControlLibrary.dll**) erzeugt.

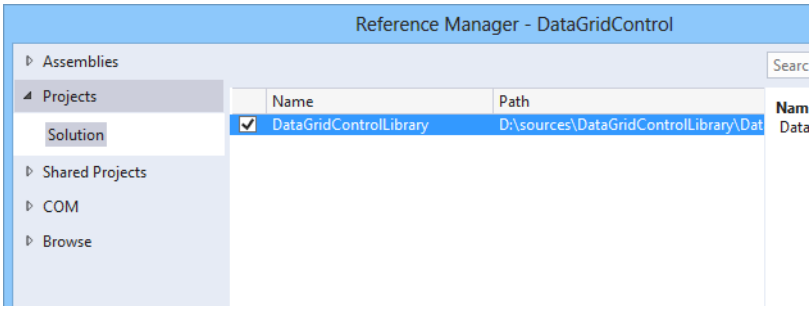
Sie verfügen jetzt über die DLL, die die benötigte Funktionalität zur Verfügung stellt.

Allerdings kann zenon nur XAML-Dateien darstellen, die nicht zur Code-Behind Datei verlinkt werden können, weshalb eine zusätzliche XAML Datei benötigt wird, die die eben gebaute DLL referenziert.

Dazu:

1. Erstellen Sie ein weiteres Projekt, wieder als **WPF User Control Library**
2. In unserem Beispiel wurde es **DataGridControl** benannt.

3. Fügen Sie in diesem neuen Projekt eine Referenz auf das eben gebaute Projekt ein.



4. Die XAML-Datei (**UserControl1.xaml**) sieht folgendermaßen aus:

```
<UserControl x:Class="DataGridControl.UserControl1"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:DataGridControl"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
  <Grid>

  </Grid>
</UserControl>
```

5. Da alle nötigen Inhalte in der zuvor erstellten DLL enthalten sind und kein Code-Behind benötigt wird, löschen Sie die folgenden Zeilen:

   x:Class="**DataGridControl.UserControl1**"

   xmlns:local="**clr-namespace:DataGridControl**"

6. Löschen Sie zusätzlich (für die Positionierung) folgende Zeilen:

   mc:Ignorable="**d**"

   d:DesignHeight="**300**" d:DesignWidth="**300**"

7. Löschen Sie die Code-Behind Datei (**UserControl1.xaml.cs**) in diesem Projekt.

8. Definieren Sie, was in der XAML-Datei angezeigt werden soll.

   Dazu modifizieren Sie die XAML Datei folgendermaßen:

   <UserControl xmlns="**http://schemas.microsoft.com/winfx/2006/xaml/presentation**"

   xmlns:x="**http://schemas.microsoft.com/winfx/2006/xaml**"

   xmlns:mc="**http://schemas.openxmlformats.org/markup-compatibility/2006**"

   xmlns:d="**http://schemas.microsoft.com/expression/blend/2008**"

   xmlns:dataGridLibrary="**clr-namespace:DataGridControlLibrary;assembly=DataGridControlLibrary**"

   <Grid x:Name="**Grid1**">

   <dataGridLibrary:UserControl1 Name="**DataGridControl**" HorizontalAlignment="**Left**" VerticalAlignment="**Top**"/>

   </Grid>

</UserControl>

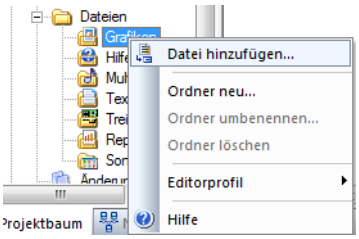
Die Zeile `xmlns:dataGridLibrary="clr-namespace:DataGridControlLibrary;assembly=DataGridControlLibrary"` definiert den Namespace **dataGridLibrary** und legt fest, dass dieser die zuvor gebaute Assembly benutzen soll.

9. Vergeben Sie einen Namen für den Grid.
10. Fügen Sie das Control **dataGridLibrary:UserControl1** aus unserer Bibliothek hinzu und vergeben Sie einen Namen, da zenon nur Objekte modifizieren kann, die einen Namen haben.
11. Bauen Sie die Solution.

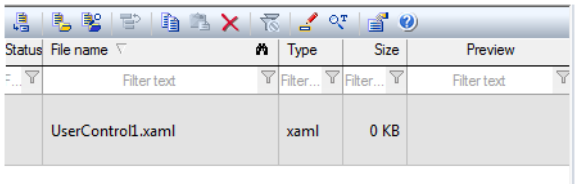
Im nächsten Schritt wird erklärt wie die DLL sowie die XAML-Datei in zenon eingefügt werden.

SCHRITTE IN ZENON

1. Öffnen Sie den zenon Editor.
2. Navigieren Sie zum Knoten `Dateien -> Grafiken`.
3. Wählen Sie im Kontextmenü **Datei hinzufügen...**.



4. Wählen Sie die XAML-Datei am Speicherort (**UserControl1.xaml** aus dem Projekt **DataGridControl**) aus und fügen Sie diese ein:

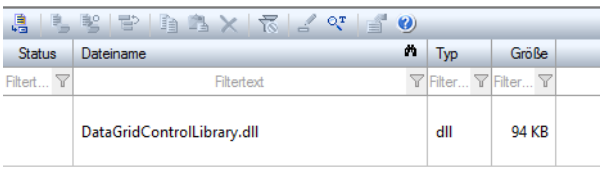


5. Fügen Sie die DLL mit der Funktionalität für die XAML-Datei hinzu.

   Dazu:

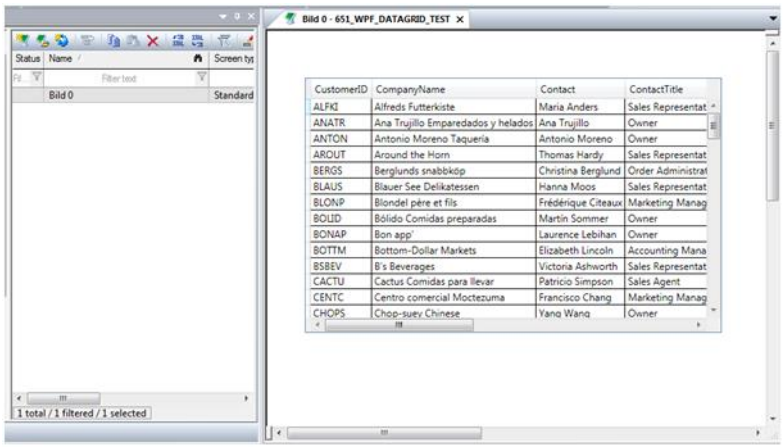
   a) Wählen Sie im Kontextmenü des Knotens Datei -> Sonstige **Datei hinzufügen...**.

b) Wählen Sie die Datei **DataGridControlLibrary.dll** des ersten Projekts (**DataGridControlLibrary**) aus.



6. Legen Sie ein zenon Bild an.
7. Fügen Sie ein WPF-Element ein und wählen Sie die zuvor eingebundene XAML-Datei aus.

   Sie sollten jetzt im zenon Editor folgendes sehen:



8. Starten Sie die zenon Runtime um auch dort das Control zu testen.

> **Achtung**
>
> *Assemblies werden nach dem Laden erst beim Beenden der Applikation wieder entladen. Das bedeutet:*
>
> *Wenn eine WPF-Datei mit einer referenzierten Assembly in zenon dargestellt wird, dann wird diese Assembly geladen und befindet sich bis zum Beenden von zenon im Speicher, selbst wenn das Bild wieder geschlossen wurde. Möchten Sie eine Assembly aus dem Ordner* `Dateien/Sonstige` *entfernen, muss zuvor der Editor neu gestartet werden, damit die Assembly entladen wird.*

6.4.8 Fehlerbehandlung

EINTRÄGE IN LOG-DATEIEN

Eintrag	Level	Bedeutung
Xaml file found in %s with different name, using default!	Warnung	Der Name der Sammeldatei und der Name der darin enthaltenen XAML-Datei stimmt nicht überein. Um interne Konflikte zu vermeiden, wird im Arbeitsordner die Datei mit dem Name der Sammeldatei und der Endung **.xaml** verwendet.
no preview image found in %s	Warnung	Die Sammeldatei enthält keine gültige Vorschaugrafik (**preview.png** oder **[Namen der XAML Datei].png**). Daher kann keine Vorschau angezeigt werden.
Xaml file in %s not found or not unique!	Fehler	Die Sammeldatei enthält keine XAML-Datei oder mehrere Dateien mit der Endung **.xam**l. sie kann nicht verwendet werden.
Could not remove old assembly %s	Warnung	Im Arbeitsordner befindet sich ein Assembly, das durch eine neuere Version ersetzt werden soll, aber nicht gelöscht werden kann.
Could not copy new assembly %s	Fehler	Für ein Assembly im Arbeitsordner ist eine neue Version verfügbar, aber sie kann nicht dorthin kopiert werden. Möglicher Grund: z. B. ist das alte Assembly noch geladen. Die alte Version wird weiter verwendet, die neue Version kann nicht benutzt werden,
file exception in %s	Fehler	Beim Zugriff auf eine Sammeldatei ist ein Dateifehler aufgetreten.
Generic exception in %s	Fehler	Beim Zugriff auf eine Sammeldatei ist ein allgemeiner Fehler aufgetreten.