



COPADATA
do it your way

zenon manual

Controls

v.7.50





©2016 Ing. Punzenberger COPA-DATA GmbH

All rights reserved.

Distribution and/or reproduction of this document or parts thereof in any form are permitted solely with the written permission of the company COPA-DATA. Technical data is only used for product description and are not guaranteed qualities in the legal sense. Subject to change, technical or otherwise.

Table of contents

1. Welcome to COPA-DATA help	5
2. Controls	5
3. General	6
3.1 Access zenon API	6
3.2 Methods	8
3.2.1 CanUseVariables	8
3.2.2 MaxVariables	9
3.2.3 VariableTypes	9
3.2.4 zenonExit	10
3.2.5 zenonExitEd	10
3.2.6 zenonInit	10
3.2.7 zenonInitEd	10
4. ActiveX	10
4.1 Develop ActiveX elements	11
4.1.1 Methods	11
4.2 Example LatchedSwitch (C++)	14
4.2.1 Interface	14
4.2.2 Control	15
4.2.3 Methods	18
4.2.4 Operation and display	21
4.2.5 zenon Interface	23
4.3 Example CD_SliderCtrl (C++)	24
4.3.1 Interface	24
4.3.2 Control	24
4.3.3 Methods	27
4.3.4 Operation and display	30
4.3.5 zenon Interface	31
4.4 Example :NET control as ActiveX (C#)	31
4.4.1 Create Windows Form Control	32
4.4.2 Change .NET User Control to dual control	35

4.4.3	Work via VBA with ActiveX in the Editor	39
4.4.4	Connect zenon variables with the .NET user control	40
5.	.NET user controls.....	44
5.1	Different use .NET Control in Control Container or ActiveX	44
5.2	Example .NET control container.....	45
5.2.1	General	45
5.2.2	Create .NET user control	47
5.2.3	add a CD_DotNetControlContainer and a .NET User Control	55
5.2.4	Accessing the user control via VSTA or VBA	60
5.3	Example :NET control as ActiveX (C#)	63
5.3.1	Create Windows Form Control.....	63
5.3.2	Change .NET User Control to dual control	66
5.3.3	Work via VBA with ActiveX in the Editor	70
5.3.4	Connect zenon variables with the .NET user control	71
6.	WPF element.....	75
6.1	Basics.....	75
6.1.1	WPF in process visualization	76
6.1.2	Transfer of values from zenon to WPF	77
6.1.3	Referenced assemblies	78
6.1.4	Workflows	79
6.2	Guidelines for designers.....	81
6.2.1	Workflow with Microsoft Expression Blend	81
6.2.2	Workflow with Adobe Illustrator.....	85
6.3	Engineering in zenon.....	93
6.3.1	CDWPF files (collective files)	93
6.3.2	create WPF element	94
6.3.3	Configuration of the linking.....	95
6.3.4	Validity of XAML Files	107
6.3.5	Pre-built elements	109
6.3.6	Display of WPF elements in the zenon web client	135
6.3.7	Examples: Integration of WPF in zenon	139
6.3.8	Error handling.....	158

1. Welcome to COPA-DATA help

GENERAL HELP

If you cannot find any information you require in this help chapter or can think of anything that you would like added, please send an email to documentation@copadata.com (<mailto:documentation@copadata.com>).

PROJECT SUPPORT

You can receive support for any real project you may have from our Support Team, who you can contact via email at support@copadata.com (<mailto:support@copadata.com>).

LICENSES AND MODULES

If you find that you need other modules or licenses, our staff will be happy to help you. Email sales@copadata.com (<mailto:sales@copadata.com>).

2. Controls

In zenon you can integrate own controls. For this following is available:

- ▶ .NET user controls (on page 44) (For implementing in zenon see also .NET controls in manual Screens.)
- ▶ ActiveX (on page 10) (For implementing in zenon see also ActiveX in manual Screens.)
- ▶ WPF (on page 75)



Information

You can find information about how to use the zenon programming interfaces (PCE, VBA, VSTA) in manual Programming Interfaces.



License information

Part of the standard license of the Editor and Runtime.



Attention

Note that errors in applications such as ActiveX, PCE, VBA, VSTA, WPF and external applications that access zenon via the API can also influence the stability of Runtime.

3. General

Controls for zenon can be implemented via ActiveX, .NET and WPF. Via VBA/VSTA you can access the zenon API.

3.1 Access zenon API

Under zenon you can enhance an ActiveX control with special functions in order to access the zenon API.

ACCESS THE ZENON API

- ▶ In **Project References**, select **Add References...** the **zenon RT object library**
- ▶ add the enhanced functions to the class code of the control

ENHANCED ZENON ACTIVEX FUNCTIONS

```
// Is called during the initializing of the control in the zenon Runtime.  
public bool zenon>Init(zenon.Element dispElement) ...  
  
// Is called during the destruction of the control in the zenon Runtime.
```

```

public bool zenonExit()
// Supports the control variable linking
public short CanUseVariables()...
// Com control supports data types.
public short VariableTypes()...
// Maximum number of variables which can be linked to the control.
public short MaxVariables()...

```

EXAMPLE

The COM object of a zenon variable is temporarily saved in a Member in order to access it later in the Paint Event of the control.

```

zenon.Variable m_cVal = null;

public bool zenon>Init(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; // the variables are supported
}

public short VariableTypes()
{
    return short.MaxValue; // all data types are supported
}

public short MaxVariables()
{
    return 1; // as maximum one variable should be linked to the control
}

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 Methods

ActiveX and .NET controls which use zenon variables need certain methods.

3.2.1 CanUseVariables

Prototype: `short CanUseVariables();`

This method either returns 1 or 0

Value	Description
1:	<p>The control can use zenon variables.</p> <p>For the dynamic element (via button Variable) you can only state zenon variables with the type stated via method VariableTypes (on page 9) in the number stated by method MaxVariables (on page 9).</p>
0:	<p>The control cannot use zenon variables or does not have the method.</p> <p>You can state variables with all types without restricting the number. In the Runtime however they only can be used with VBA.</p>

3.2.2 MaxVariables

Prototype: `short MaxVariables()` ;

Here the number of variables is defined, that can be selected from the variable list.

If 1 is returned, multi-select is disabled in the variable list. A warning is displayed when several variables are selected anyway.

3.2.3 VariableTypes

Prototype: `short VariableTypes()` ;

The value returned by this method is used as a mask for the usable variable types in the variable list. The value is an **AND** relation from the following values (defined in `zenon32/dy_type.h`):

Value 1	Value 2	Equivalent
WORD	0x0001	Position 0
BYTE	0x0002	Position 1
BIT	0x0004	Position 2
DWORD	0x0008	Position 3
FLOAT	0x0010	Position 4
DFLOAT	0x0020	Position 5
STRING	0x0040	Position 6
IN_OUTPUT	0x8000	Position 15

3.2.4 zenonExit

Prototype: `boolean zenonExit();`

This method is called by the zenon Runtime when the ActiveX control is closed.

Here all dispatch pointers on variables should be released.

3.2.5 zenonExitEd

Equals zenonExit (on page 10) and is executed in closing the ActiveX in the Editor.

Therewith you can also react to changes in the ActiveX e.g. values changes in Editor.

Info: Currently only available for ActiveX.

3.2.6 zenonInit

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

With this method (in the Runtime) the ActiveX control gets a pointer to the dispatch interface of the dynamic element. With this pointer zenon variables linked to the dynamic element can be accessed.

You define the sorting order of the handed over variables in the configuration of the ActiveX element with the help of buttons **Down** or **Up**.

The **Element Input** dialog appears after double-clicking the ActiveX element or after selecting property **ActiveX settings** in the element properties in node **Representation**.

3.2.7 zenonInitEd

Equals zenonInit (on page 10) and is executed on opening the ActiveX (double click the ActiveX) in the Editor.

Info: Currently only available for ActiveX.

4. ActiveX

With ActiveX the functionality of the zenon Runtime and Editor can be enhanced autonomously.

In this manual you can find:

- ▶ Develop ActiveX elements (on page 11)
- ▶ Example LatchedSwitch (C++) (on page 14)
- ▶ Example CD_SliderCtrl (C++) (on page 24)
- ▶ Example :NET control as ActiveX (C#) (on page 31)

You can find information about the dynamic element ActiveX in manual Screens in chapter ActiveX.

ACTIVEX FOR WINDOWS CE

If an ActiveX Control should run under Windows CE, the **apartment** model must be set to `Threading`. If it is set to `Free`, the control will not run in zenon Runtime.

4.1 Develop ActiveX elements

The dynamic element ActiveX in zenon can forward variables to the ActiveX control without using VBA to operate the control.

The control now defines by itself, how many zenon variables it can use and of what type they may be. The properties of the control can be established by means of dynamic elements.

To do this, the interface (dispatch interface) of the control must support a range of certain methods (on page 11).

4.1.1 Methods

Each ActiveX control which can use zenon variables must contain the following methods:

- ▶ `CanUseVariables` (on page 8)
- ▶ `MaxVariables` (on page 9)
- ▶ `VariableTypes` (on page 9)
- ▶ `zenonExit` (on page 10)
- ▶ `zenonExitEd` (on page 10)
- ▶ `zenonInit` (on page 10)
- ▶ `zenonInitEd` (on page 10)

It does not matter, which dispatch ID the methods have in the interface. On calling the methods zenon receives the correct ID from the interface.

CanUseVariables

Prototype: `short CanUseVariables() ;`

This method either returns 1 or 0

Value	Description
1:	<p>The control can use zenon variables.</p> <p>For the dynamic element (via button Variable) you can only state zenon variables with the type stated via method <code>VariableTypes</code> (on page 9) in the number stated by method <code>MaxVariables</code> (on page 9).</p>
0:	<p>The control cannot use zenon variables or does not have the method.</p> <p>You can state variables with all types without restricting the number. In the Runtime however they only can be used with VBA.</p>

MaxVariables

Prototype: `short MaxVariables() ;`

Here the number of variables is defined, that can be selected from the variable list.

If 1 is returned, multi-select is disabled in the variable list. A warning is displayed when several variables are selected anyway.

VariableTypes

Prototype: `short VariableTypes() ;`

The value returned by this method is used as a mask for the usable variable types in the variable list. The value is an **AND** relation from the following values (defined in `zenon32/dy_type.h`):

Value 1	Value 2	Equivalent
WORD	0x0001	Position 0
BYTE	0x0002	Position 1
BIT	0x0004	Position 2
DWORD	0x0008	Position 3
FLOAT	0x0010	Position 4
DFLOAT	0x0020	Position 5
STRING	0x0040	Position 6
IN_OUTPUT	0x8000	Position 15

zenonExit

Prototype: `boolean zenonExit();`

This method is called by the zenon Runtime when the ActiveX control is closed.

Here all dispatch pointers on variables should be released.

zenonExitEd

Equals zenonExit (on page 10) and is executed in closing the ActiveX in the Editor.

Therewith you can also react to changes in the ActiveX e.g. values changes in Editor.

Info: Currently only available for ActiveX.

zenonInit

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

With this method (in the Runtime) the ActiveX control gets a pointer to the dispatch interface of the dynamic element. With this pointer zenon variables linked to the dynamic element can be accessed.

You define the sorting order of the handed over variables in the configuration of the ActiveX element with the help of buttons **Down** or **Up**.

The **Element Input** dialog appears after double-clicking the ActiveX element or after selecting property **ActiveX settings** in the element properties in node **Representation**.

zenonInitEd

Equals zenonInit (on page 10) and is executed on opening the ActiveX (double click the ActiveX) in the Editor.

Info: Currently only available for ActiveX.

4.2 Example LatchedSwitch (C++)

The following example describes an ActiveX control, that realizes a latched switch with two bit variables. The first variable represents the switch, the second variable the lock. The value of the switching variable of the ActiveX control can only be changed, if the locking variable has the value 0.

The status of the element is displayed with four bitmaps which can be selected in the properties dialog of the control in the zenon Editor.

4.2.1 Interface

The control LatchedSwitch has the following 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();
};
```

The properties **SwitchOn** to **LatchedOff** contain the bitmaps for displaying the four different states of the control. The bitmaps themselves are stored in objects of the class **CScreenHolder**. The property **SollwertDirekt** defines if the input of set values is done via a dialog or directly by clicking the control.

4.2.2 Control

Implementing the control is done with the class **CLatchedSwitchCtrl**. As members this class has the **CScreenHolder** objects for the storage of the bitmaps. Additionally three dispatch drivers for the dynamic element and the variables are generated:

```
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
CScreenHolder m_SwitchOn;
CScreenHolder m_SwitchOff;
CScreenHolder m_LatchedOn;
CScreenHolder 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 Methods

The following methods are used:

- ▶ CanUseVariables (on page 18)
- ▶ VariableTypes (on page 18)
- ▶ MaxVariables (on page 18)
- ▶ zenonInit (on page 19)
- ▶ zenonExit (on page 20)

CanUseVariables

This method returns 1 so zenon variables can be used.

```
short CLatchedSwitchCtrl::CanUseVariables()  
{  
  
    return 1;  
}
```

VariableTypes

The control only can work with bit variables, so 0x0004 is returned.

```
short CLatchedSwitchCtrl::VariableTypes()  
{  
  
    return 0x0004;    // Only bit variables  
}
```

MaxVariables

Two variables can be used. Therefore 2 is returned.

```
short CLatchedSwitchCtrl::MaxVariables()  
{  
  
    return 2; // 2 variables  
}
```

zenonInit

This method gets the `Dispatchdriver` of the variables via the `Dispatchpointer` of the dynamic element. With this `Pointer` the variable values are read and written when clicking and drawing the control.

```
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;
}
```



Information

Element.m_lpDispatch->AddRef();

Objects that are not used are automatically deleted from the memory. This must be carried out by the programming. The programmer determines whether an object - based on a reference counter - can be removed.

COM uses the `IUnknown` methods `AddRef` and `Release` to administer the number of references of interfaces to an object.

The general rule for calling up these methods are:

- ▶ `AddRef` must always be called up on the interface if the client receives an interface pointer.
- ▶ A `Release` must always be called up if the client ends the use of the interface pointer.

With a simple implementation, a counter variable in the object is increased with an `AddRef` call. Each call of a `Release` reduces this counter in the object. If this counter is at ZERO again, the interface can be removed from the memory.

A reference counter can also be implemented so that each reference to the object (and not to an individual interface) is counted.

In this case, each `AddRef` and each `Release` substitute call up a central implementation to the object. A `Release` then unlocks the complete object if the reference counter has reached zero.

zenonExit

This method releases the dispatch driver.

```

BOOL CLatchedSwitchCtrl::zenonExit()
{

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

```

4.2.4 Operation and display

Write set value

A value can be set by clicking the control with the left mouse button.

If **m_iSollwertDirekt** is 0, a dialog for the selection of the set value is opened, otherwise the current value of the switching variable is inverted.

If the locking variable has the value 1, only a `MessageBeep` is executed. No value can be set via the control.

```
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)    // Locked!!!

        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);
}
CObjectControl::OnLButtonDown(nFlags, point);
}

```

Drawing

On drawing the control the values of the variables are read via their dispatch drivers, and accordingly one of the four defined graphics is displayed. When the value of a variable changes, the control is updated by the **OnDraw** routine.

```

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 exists?
{

vRes = m_dSwitchVar.GetValue();

```

```

bVal1 = (BOOL)VariantToDouble(&vRes);
}
if (m_dLatchVar)      // Variable exists?
{

vRes = m_dLatchVar.GetValue();
bVal1 = (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

Classes deduced from COleDispatchDriver have to be created for the element and the variables, so that the dispatch interface of zenon can be used to set values. The easiest way to create these classes is the Class Wizard of the development environment (button **Add Class**, select **From a type library**, select zenrt32.tlb).

For our control these are the classes **IElement** and **IVariable**. They are defined in zenrt32.h and zenrt32.cpp.

4.3 Example CD_SliderCtrl (C++)

The following example describes an ActiveX control which equals the Windows **SliderCtrl**. This component can be linked with a zenon variable. The user can change the value of a variable with this slider. If the value of the variable is changed with some other dynamic element, the slider is updated.

4.3.1 Interface

The control **CD_SliderCtrl** has the following dispatch interface:

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

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

  methods: /*** method of the control (for 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

Implementing the control is done with the class **CD_SliderCtrlCtrl**. This class has a standard Windows **CSliderCtrl** as a member, with which the control is subclassed. The interfaces **IVariable** and **IElement** contain zenon interfaces which had to be integrated. These are deduced from **COleDispatchDriver**.


```

class CCD_SliderCtrlCtrl : public COleControl
{

    DECLARE_DYNCREATE(CCD_SliderCtrlCtrl)
private: /*** member variables

    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();
    /*** methods for the conversion from 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

```

```

/** methods for the functionality of the SliderCtrl
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   HScroll(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 Methods

The following methods are used:

- ▶ CanUseVariables (on page 27)
- ▶ VariableTypes (on page 28)
- ▶ MaxVariables (on page 28)
- ▶ zenonInit (on page 28)
- ▶ zenonExit (on page 29)

CanUseVariables

This method returns 1 so zenon variables can be used.

```
short CCD_SliderCtrlCtrl::CanUseVariables()  
{  
  
    return 1;  
}
```

VariableTypes

The control can work with word, byte, doubleword and float variables. You will find a list of the possible data types in the general description (on page 9) of this method.

```
short CCD_SliderCtrlCtrl::VariableTypes()
{

return 0x0001 | // Word

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

MaxVariables

Only one variable can be linked to this control.

```
short CCD_SliderCtrlCtrl::MaxVariables()
{

return 1; // 1 variables
}
```

zenonInit

The parameter **dispElement** contains the interface for the dynamic element. With this element the linked zenon variable determined. If it is valid, the area of the **SlideCtrl** is set. Additionally the settings for the display (number of ticks, ...) are set. If no variable is linked, the display range is set to 0 to 0. Thus the SliderCtrl cannot be changed. The variable **m_bInitialized** defines that values can be set from now on.

```
BOOL CCD_SliderCtrlCtrl::zenonInit(LPDISPATCH dispElement)
{
/** Determine the variable using the zenon element

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

short nIndex = 0;
```

```

m_interfaceVariable = IVariable
(m_interfaceElement.ItemVariable(ColeVariant(nIndex)));
}

/** Initialize the area of the Slider-Ctrl
if (m_interfaceVariable) {

/** Define range
m_nRangeStart = (long) VariantToDouble(&m_interfaceVariable.GetRangeMin());
m_nRangeEnd = (long) VariantToDouble(&m_interfaceVariable.GetRangeMax());
m_wndSliderCtrl.SetRange(m_nRangeStart,m_nRangeEnd,TRUE);
/** Define sub ticks
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 this method the zenon interfaces are released again.

```

BOOL CCD_SliderCtrlCtrl::zenonExit()
{

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

```

4.3.4 Operation and display

Drawing

With **DoSuperclassPaint** the SliderCtrl is drawn (as is a subclassed control). If at the moment of drawing the slider is moved, the variable **m_bInitialized** gets the value `FALSE`. This makes sure that the value can be changed. Normally the value of the variable is read and displayed with the method **SetPos** of the SliderCtrl.

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

    /*** update view
    DoSuperclassPaint(pdc, rcBounds);
    if (m_interfaceVariable && m_bInitialized) {

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

Write set value

In the method **LButtonDown** the variable **m_bInitialized** is set to `FALSE`, and in the event **LbuttonUp** it is set to `TRUE` again. This makes sure that the value can be changed. Otherwise the routine **OnDraw** would be executed and the old value would be displayed.

```
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);
}
```

A value is sent to the hardware, when the slider is moved. In the methods **Hscroll** or **Vscroll** the value is sent to the hardware (depending if it is a horizontal or a vertical slider).

```
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:{

    /*** Set value without dialog ?
    int nValue =          m_wndSliderCtrl.GetPos();
    COleVariant cValue((short) nValue,VT_I2);
    m_interfaceVariable.SetValue(cValue);
    }
    }
}
```

4.3.5 zenon Interface

Classes deduced from COleDispatchDriver have to be created for the element and the variables, so that the dispatch interface of zenon can be used to set values. The easiest way to create these classes is the Class Wizard of the development environment (button **Add Class**, select **From a type library**, select zenrt32.tlb).

For our control these are the classes **IElement** and **IVariable**. They are defined in zenrt32.h and zenrt32.cpp.

4.4 Example :NET control as ActiveX (C#)

The following example describes a .NET control which is executed as ActiveX control in zenon.

The creation and integration is carried out in four steps:

1. Create Windows Form Control (on page 32)

2. Change .NET User Control to dual control (on page 35)
3. Work via VBA with ActiveX in the Editor (on page 39)
4. Connect zenon variables with the .NET user control (on page 40)



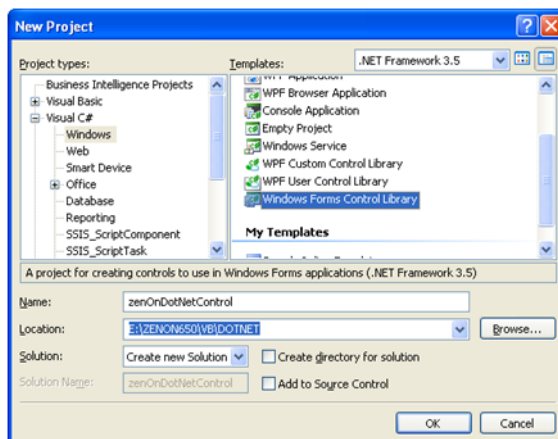
Information

The screenshots for this theme are only available in English.

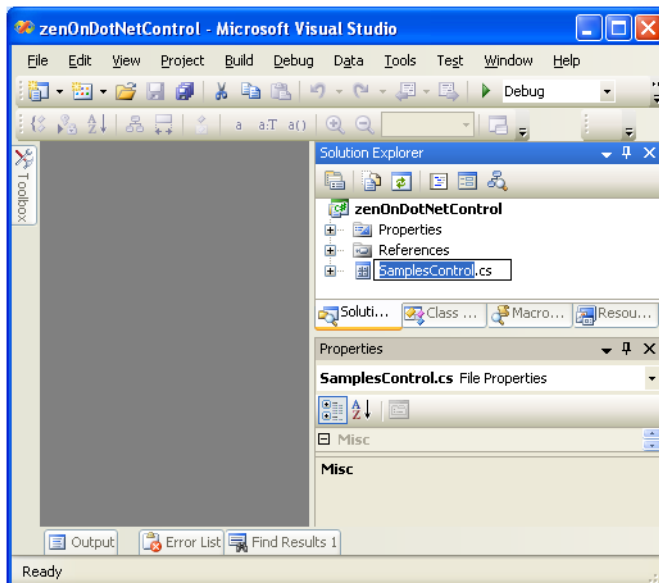
4.4.1 Create Windows Form Control

To create a Windows Form Control:

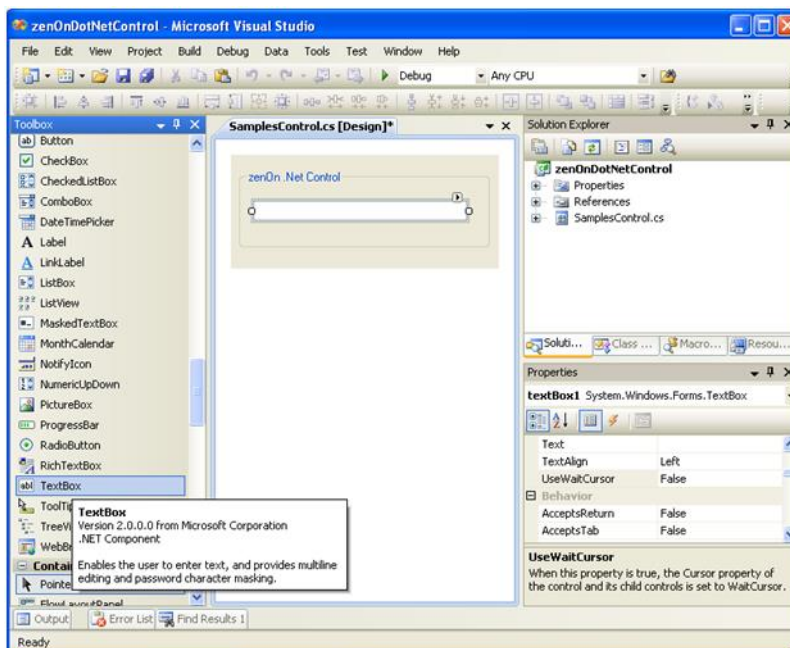
1. Start Visual Studio 2008 and create a new Windows **Form Control Library** project:



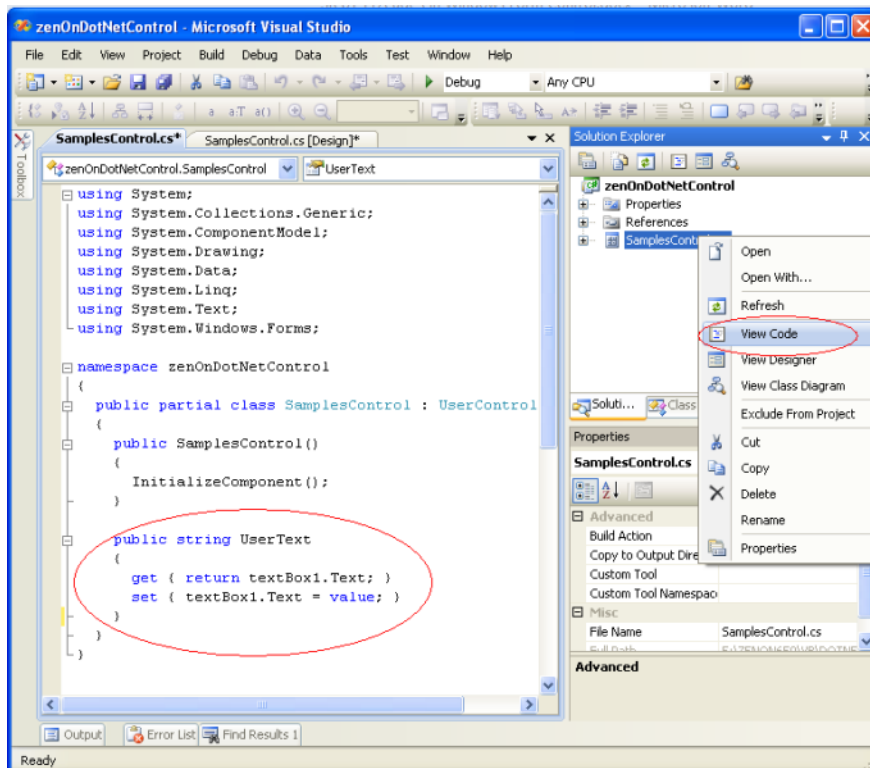
2. Rename the default control to the desired control name.
In our example: **SampesControl.cs**.



3. Open the Control Designer and add the desired control; in our case a text box:

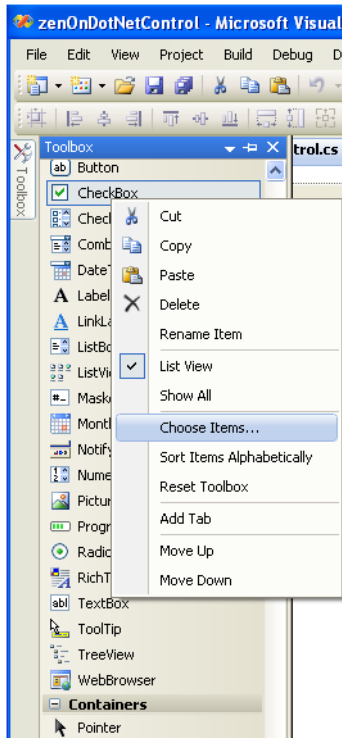


4. Normally controls have properties. Open the Code Designer via **View Code** and ass the desired properties which should be available externally.
In our example: Externally visible property „**UserText**“ with **get** and **set** access which contains the text of the text box:



5. Compile the project.
The Windows Forms Control can now be used in other Windows Forms projects.

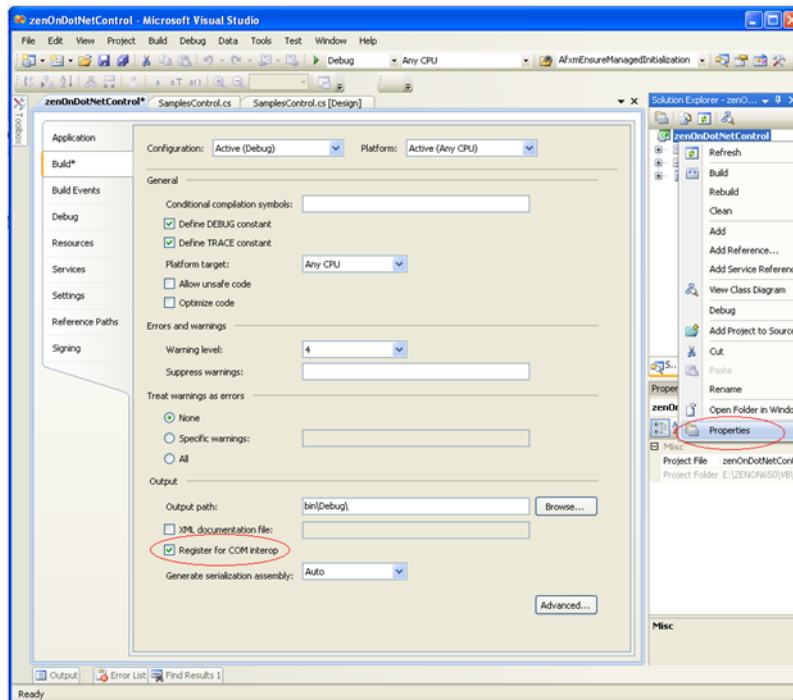
Important: The control must be inserted manually in the control tool box via **Choose Items**.



4.4.2 Change .NET User Control to dual control

To change the .NET in a dual control, you must first activate the COM interface for ActiveX.

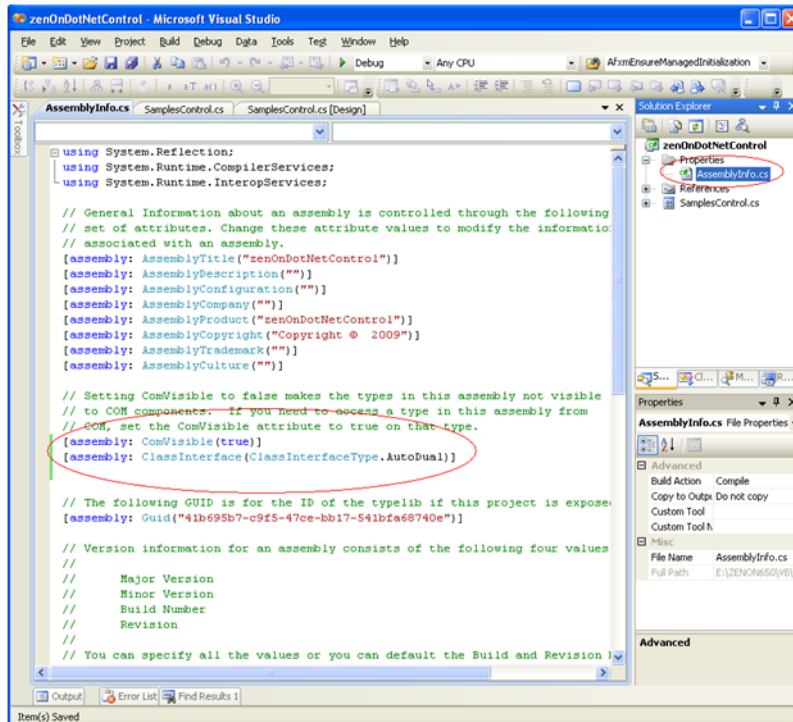
1. Open the project and activate property **Register for COM interop** in the **Build** settings:



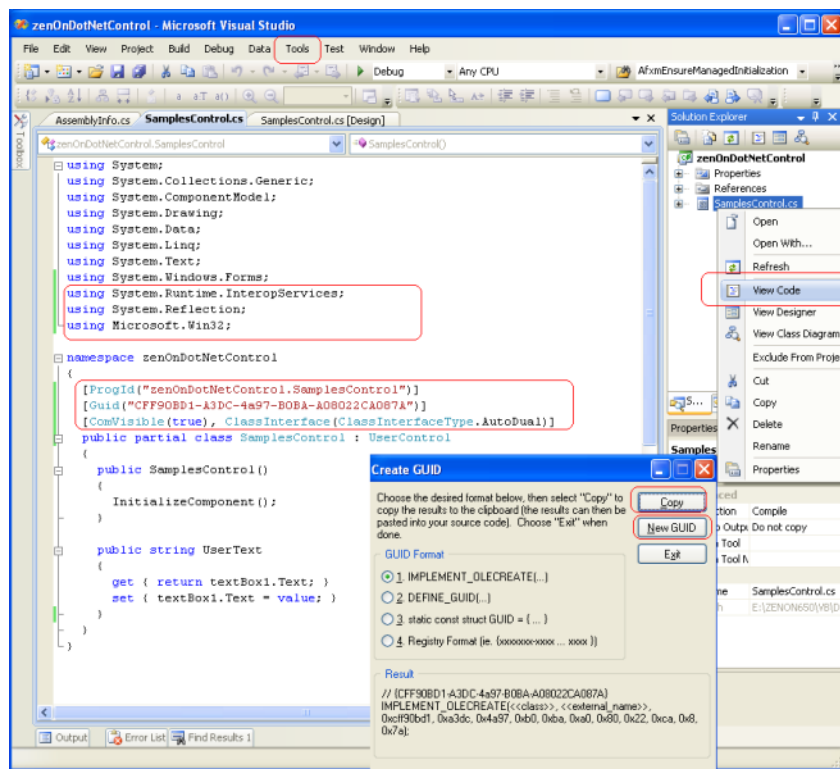
2. Open the file **AssemblyInfo.cs** and
 - set attribute **ComVisible** to **true**
 - add attribute **ClassInterface**

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

[assembly: ClassInterface(ClassInterfaceType.AutoDual)]

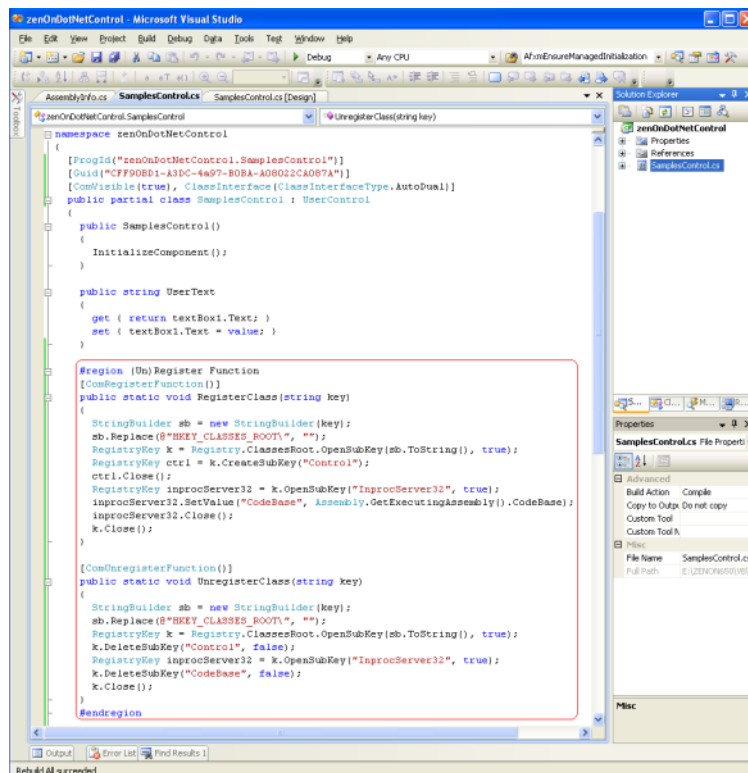


3. Open the code designer via **View Code** and add the necessary ActiveX attributes and **using** entries. Via menu **Tools/Create GUID** create a new GUID for the GUID attribute:



4. For the control to be selectable as Active X user interface control, you must add the functions to the following control classes:

- RegisterClass
- UnregisterClass



After that you can register the control in the registry.

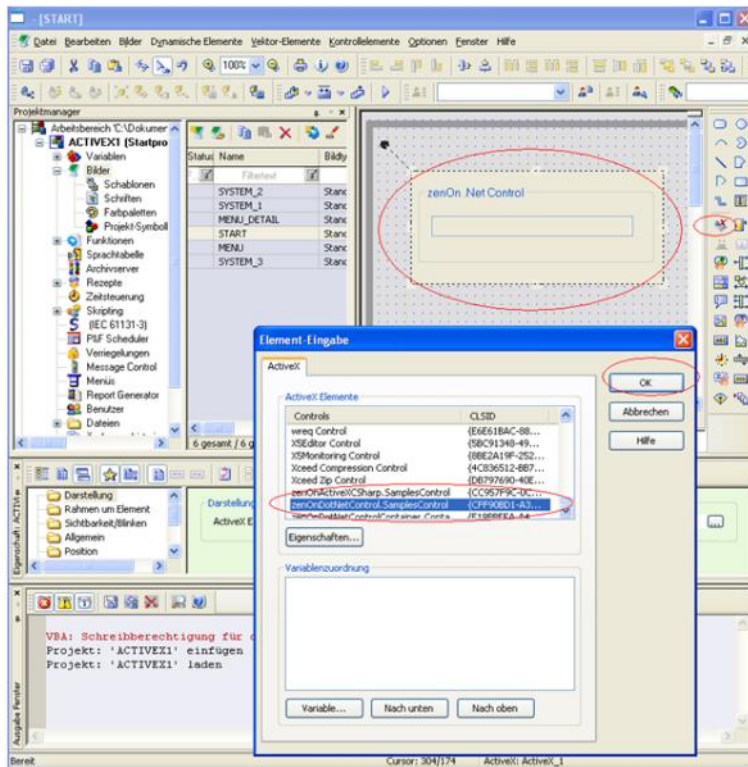
5. Compile the project again.

The Windows Form Control is now ActiveX-able and was registered automatically during the rebuild. An additional typelib file **zenOnDotNetControl.tlb** was created in the output directory.

6. To use the control on another computer:

- a) copy the DLL file and the TLB file to the target computer
- b) register the files via the command line:
%windir%\Microsoft.NET\Framework\v2.0.50727/regasm.exe zenOnDotNetControl.dll /tlb:zenOnDotNetControl.tlb

7. Add the extended Windows Form Control as ActiveX control to the zenon Editor:



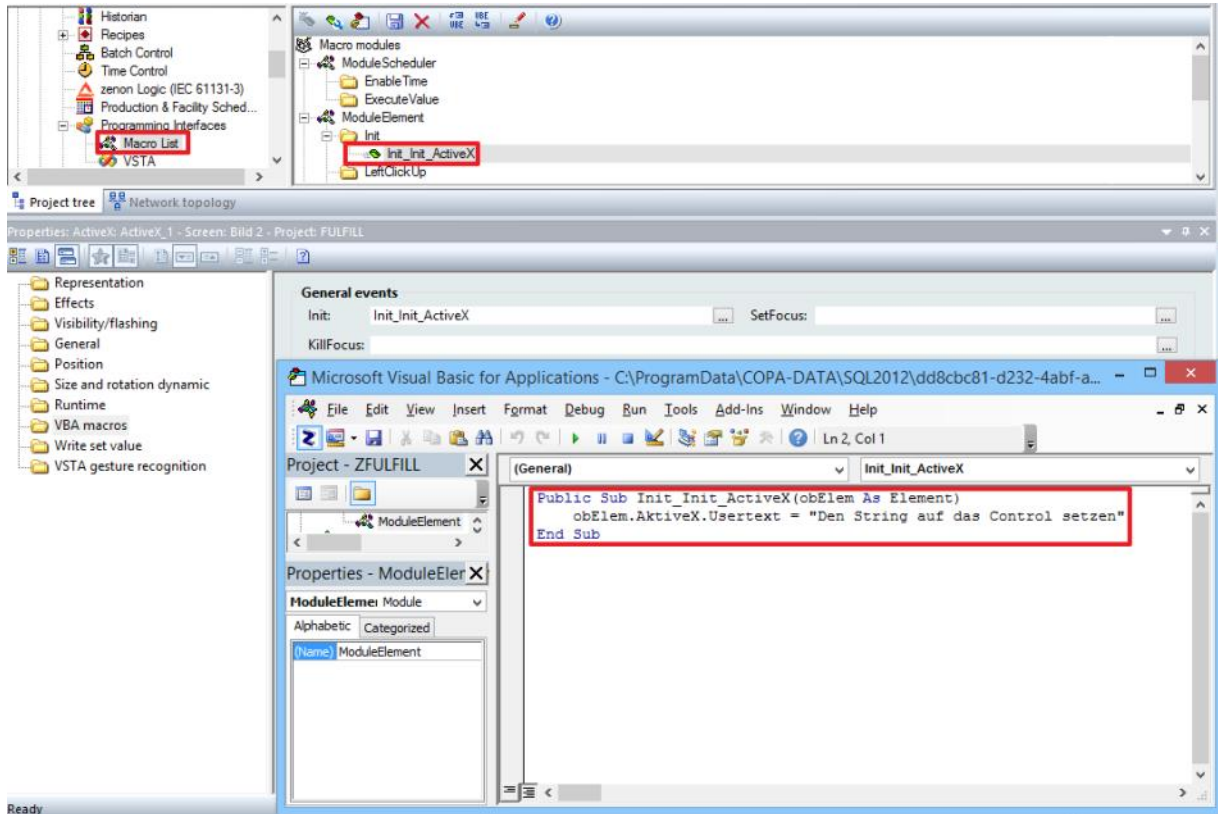
4.4.3 Work via VBA with ActiveX in the Editor

To access the properties of the control in the zenon Editor:

1. In the zenon Editor in node **Programming interfaces/VBA macros** create a new **Init** macro with the name **Init_ActiveX**.

In this macro you can access all external properties via **obElem.ActiveX**.

2. Assign his macro to the ActiveX control via properties **VBA macros/Init** of the ActiveX element.



EXAMPLE INIT MACRO

```
Public Sub Init_ActiveX(obElem As Element)
    obElem.ActiveX.UserText = "Set the string to the control"
End Sub
```

4.4.4 Connect zenon variables with the .NET user control

In zenon you have the possibility to enhance an ActiveX control with special functions in order to access the zenon API.

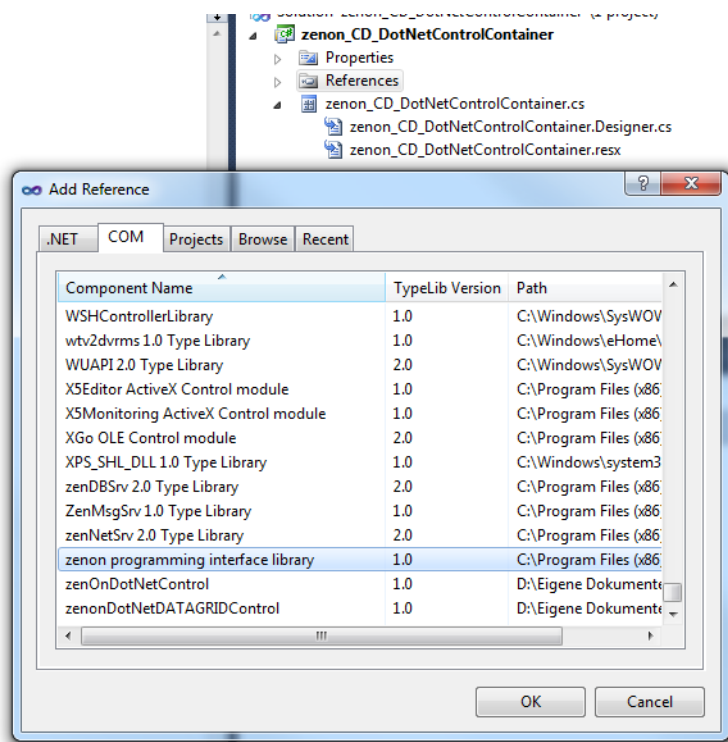
NECESSARY METHODS

- ▶ public bool zenOnInit (on page 42) (Is called up during control initializing in the zenon Runtime.)
- ▶ public bool zenOnInitED (on page 42) (Is used in the Editor.)
- ▶ public bool zenOnExit() (on page 43) (Is called up during control destruction in the zenon Runtime.)

- ▶ public bool zenOnExitED() (on page 43) (Is used in the Editor.)
- ▶ public short CanUseVariables() (on page 43) (Supports linking variables.)
- ▶ public short VariableTypes() (on page 43) (Supported data types by the control)
- ▶ public MaxVariables() (on page 44)(Maximum number of variables which can be linked to the control.)

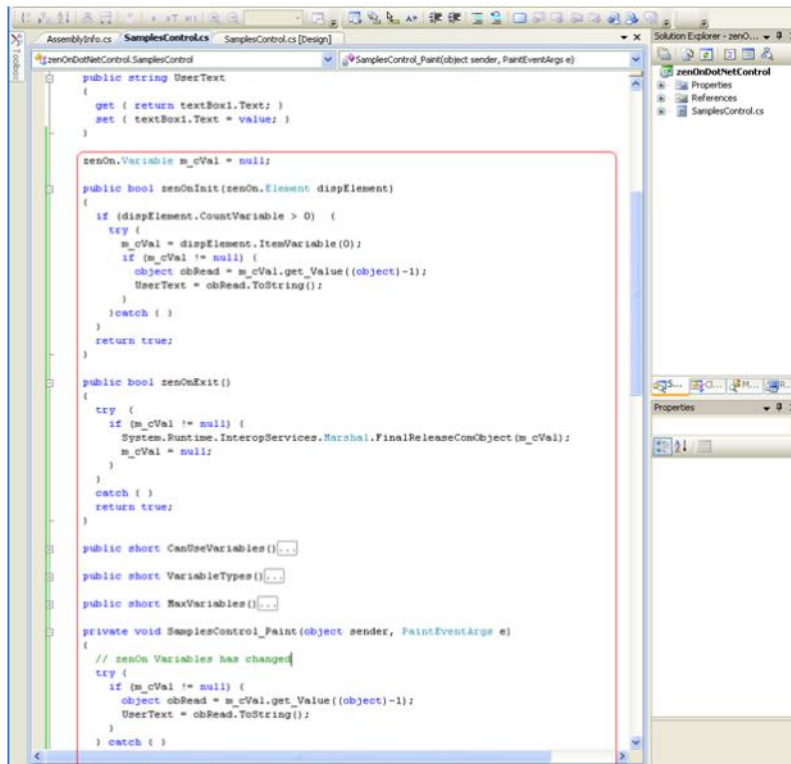
ADD REFERENCE

1. Select in Microsoft Visual Studio under **Add References** the zenon Runtime object library in order to be able to access the zenon API in the control.



2. Add the enhanced functions in the class code of the control in order to access the whole zenon API.

In our example the COM object of a zenon variable is temporarily saved in a **Member** in order to access it later in the **Paint** event of the control.



public bool zenOnInit(zenOn.Element dispElement)

With this method (in the Runtime) the ActiveX control gets a pointer to the dispatch interface of the dynamic element. With this pointer zenon variables linked to the dynamic element can be accessed.

You can configure the sequence of the sent variables in the Enter Element dialog with the buttons **down** or **up**. The dialog "element input" opens if:

- ▶ you double click the ActiveX element or
- ▶ select **Properties** in the context menu or
- ▶ select the **ActiveX settings** property in the **Representation** node of the property window

public bool zenOnInitED(zenOn.Element dispElement)

Equals public bool zenOnInit (on page 42) and is executed when opening the ActiveX in the Editor (double click on ActiveX).

public bool zenOnExit()

This method is called by the zenon Runtime when the ActiveX control is closed. Here all dispatch pointers on variables should be released.

public bool zenOnExitED()

Equals public bool zenOnExit() (on page 43) and is executed in closing the ActiveX in the Editor. With this you can react to changes, e.g. value changes, in the Editor.

public short CanUseVariables()

This method returns 1 if the control can use zenon variables and 0 if it cannot.

- ▶ 1: For the dynamic element (via button **Variable**) you can only state zenon variables with the type stated via method **VariableTypes** in the number stated by method **MaxVariables**.
- ▶ 0: If **CanUseVariables** returns 0 or the control does not have this method, any number of variables of all types can be defined without limitations. In the Runtime however they only can be used with VBA.

public short VariableTypes()

The value returned by this method is used as a mask for the usable variable types in the variable list. The value is an **AND** relation from the following values (defined in **zenon32/dy_type.h**):

Parameters	Value	Description
WORD	0x0001	corresponds to position 0
BYTE	0x0002	corresponds to position 1
BIT	0x0004	corresponds to position 2
DWORD	0x0008	corresponds to position 3
FLOAT	0x0010	corresponds to position 4
DFLOAT	0x0020	corresponds to position 5
STRING	0x0040	corresponds to position 6
IN_OUTPUT	0x8000	corresponds to position 15

public MaxVariables()

Here the number of variables is defined, that can be selected from the variable list:

1: Multi-select is disabled in the variable list. A warning is displayed when several variables are selected anyway.

5. .NET user controls

With .NET control the functionality of the zenon Runtime and Editor can be enhanced autonomously.

In this manual you can find:

- ▶ Difference between control container and ActiveX (on page 44)
- ▶ Example .NET control container (on page 45)
- ▶ Example :NET control as ActiveX (C#) (on page 31)

You can find information about .NET controls in ActiveX in manual Screens in chapter .NET controls.

5.1 Different use .NET Control in Control Container or ActiveX

A .NET user control can:

- ▶ be integrated directly in the zenon ActiveX element via the CD_DotNetControlContainer control
- ▶ be used as ActiveX control and be integrated directly in the zenon ActiveX element

Above all the differences between container control and ActiveX control are:

CD_DotNetControlContainer control	ActiveX control
▶ Does not have to be registered at the computer.	▶ Must be registered as Active X at the computer (regsvr32).
▶ For changes at the controller only the DLL must be changed.	▶ For changes at the controller the TLB must be registered again.
▶ Access via VBA and VSTA only possible via the CD_DotNetControlContainer method.	▶ Easy access via VBA and VSTA.

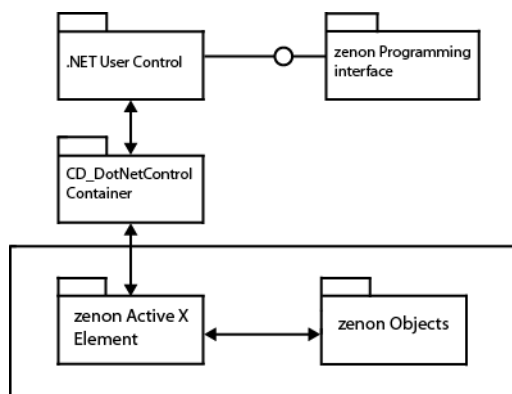
5.2 Example .NET control container

In this tutorial you get to know how to create a simple .NET user control in Visual Studio 2010 (programming language **C#**) and how to integrate it with the help of the zenon **CD_DotNetControlContainer** control as ActiveX in a zenon ActiveX element.

5.2.1 General

The CD_DotNetControlContainer therefore acts as a wrapper between the user control and the zenon ActiveX element. All methods used in the following example and all public methods and properties are passed on via the CD_DotNetControlContainer from the user control to the ActiveX and can be used by zenon; also in VBA and VSTA.

If there is a reference to the zenon programming interface in the user control, you can directly access >CD_PRODUCTNAME< objects.



In the following example we will:

- ▶ create .NET user control (on page 47)
- ▶ add a CD_DotNetControlContainer and a .NET User Control (on page 55)
- ▶ enable the access to the user control via VSTA (VBA) (on page 60)

PATH FOR DLL IN EDITOR AND RUNTIME

The path to **.Net DLL** that is selected in the Editor is also used in Runtime. It is set as absolute and cannot be changed.

Ensure that the same path is used on all computers in the zenon network for Editor and Runtime.

Hint: Select an absolute path, for example: C : \Controls. Enter the path as fixed in

Remote-Transport and in the **.NET Control Container**. Use **Remote-Transport** to harmonize this path with all computers.

public bool zenOnInit(zenOn.Element dispElement)

With this method (in the Runtime) the ActiveX control gets a pointer to the dispatch interface of the dynamic element. With this pointer zenon variables linked to the dynamic element can be accessed.

You can configure the sequence of the sent variables in the Enter Element dialog with the buttons **down** or **up**. The dialog "element input" opens if:

- ▶ you double click the ActiveX element or
- ▶ select **Properties** in the context menu or
- ▶ select the **ActiveX settings** property in the **Representation** node of the property window

public bool zenOnExit()

This method is called by the zenon Runtime when the ActiveX control is closed. Here all dispatch pointers on variables should be released.

public short CanUseVariables()

This method returns 1 if the control can use zenon variables and 0 if it cannot.

- ▶ 1: For the dynamic element (via button **Variable**) you can only state zenon variables with the type stated via method **VariableTypes** in the number stated by method **MaxVariables**.
- ▶ 0: If **CanUseVariables** returns 0 or the control does not have this method, any number of variables of all types can be defined without limitations. In the Runtime however they only can be used with VBA.

public short VariableTypes()

The value returned by this method is used as a mask for the usable variable types in the variable list. The value is an **AND** relation from the following values (defined in **zenon32/dy_type.h**):

Parameters	Value	Description
WORD	0x0001	corresponds to position 0
BYTE	0x0002	corresponds to position 1
BIT	0x0004	corresponds to position 2
DWORD	0x0008	corresponds to position 3
FLOAT	0x0010	corresponds to position 4
DFLOAT	0x0020	corresponds to position 5
STRING	0x0040	corresponds to position 6
IN_OUTPUT	0x8000	corresponds to position 15

public MaxVariables()

Here the number of variables is defined, that can be selected from the variable list:

1: Multi-select is disabled in the variable list. A warning is displayed when several variables are selected anyway.

5.2.2 Create .NET user control

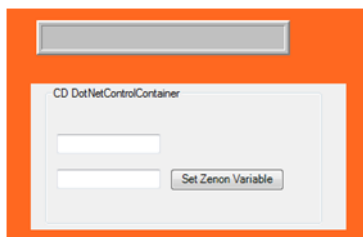
The user control is a simple control which can set a new value via an input field (text box). After clicking the button, the value is written to the desired zenon variable.

An additional function should automatically detect the change of value of the variable in zenon and display the new value automatically in the control.



Information

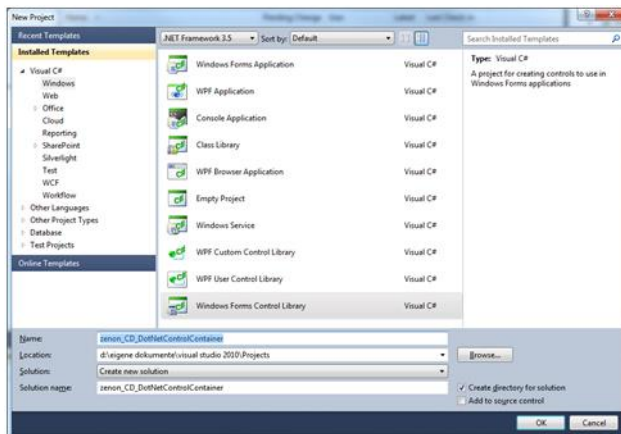
The screenshots for this theme are only available in English.



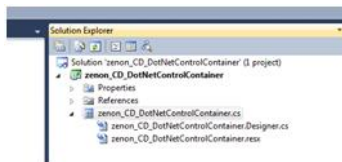
WORK STEPS

1. First you create a new project in VS and use project type „**Windows Forms Control Library**“

Important: Set framework to 3.5!



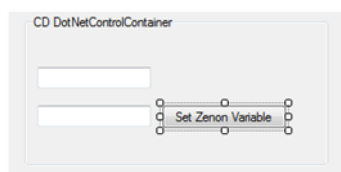
2. After that rename the CS file from "**UserControl**" to "**zenon_CD_DotNetControlContainer.cs**". The files **Designer.cs** and the **.resx** are renamed automatically.



3. In the next step you create the user control. For this use two text boxes one each for the input and the output and a button for writing new values to the zenon variable.

Name:

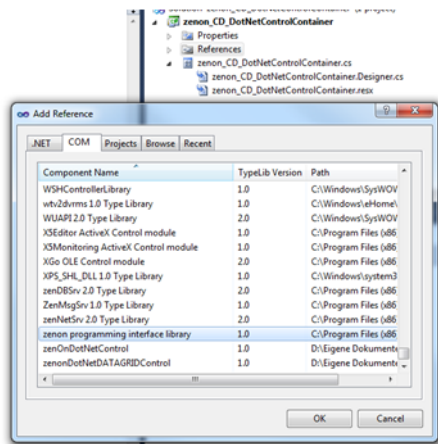
- the first text box "**txtGetZenonVariable**"
- the second text box "**txtSetZenonVariable**"
- the button "**btnSetZenonVariable**"



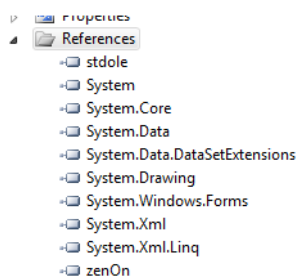
4. In order to access zenon objects you need a reference to the <CD_PRODUCNAME> Programming Interface. To do this:

- click on node "**References**" in the Solution Explorer
- open the context menu
- select **Add References...**
- switch to tab **COM**

- select **zenon programming interface library**



After that the "zenOn" reference should be visible in the reference list.



5. In the next step create a global variable of type `zenon.variable` in the code of the **zenon_CD_DotNetControlContainer.cs**:

```

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. This variable is initialized via public method **zenOnInit**:

```

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              txtSetZenonVariable.Text = m_cVal.get_Value(0).ToString();
42          }
43          catch { }
44      }
45      return true;
46  }

```

and enabled via public method **zenOnExit**:

```

47  /// <summary>
48  /// This public Method will be called 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 the following methods we define whether <CD_PRODUTCNAME> variables and data types are used and how many variables may be handed over:

```

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. In the next step define in the **Click-Event** of button **btnSetZenonVariable** that when you click the button the value of text box **txtSetZenonVariable** is written to the zenon variable and then the content of the text box is deleted.

```

98  /// <summary>
99  /// This will be triggered 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. To react to a value change of the variable, you need the **Paint Event** of the control. The **Paint Event** is also triggered if the value of the initialized zenon variable changes and it can therefore be used to update values. As variables which are referenced in the zenon ActiveX element are

automatically advised, you can generally refrain from using the **zenon.OnlineVariable** container in the control.

```

113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
    /// <summary>
    /// This will be triggered 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;
        }
    }

```

THE CODE AT A GLANCE

Here is the whole code as review:

```

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 triggered 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 triggered 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;

        }

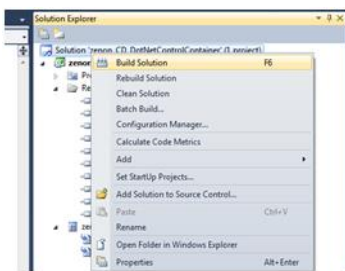
    }

}

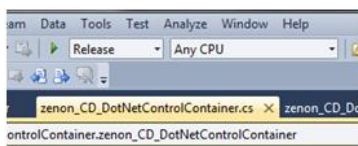
```

CREATE RELEASE

At last create a Release in order to integrate the completed DLL in zenon or in the **CD_DotNetControlContainer**.



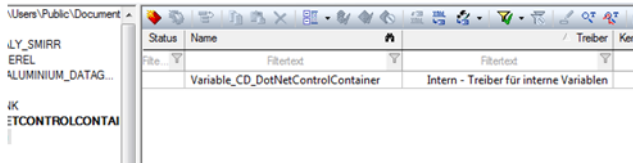
For this it is necessary that you switch from **Debug** to **Release** in the settings.



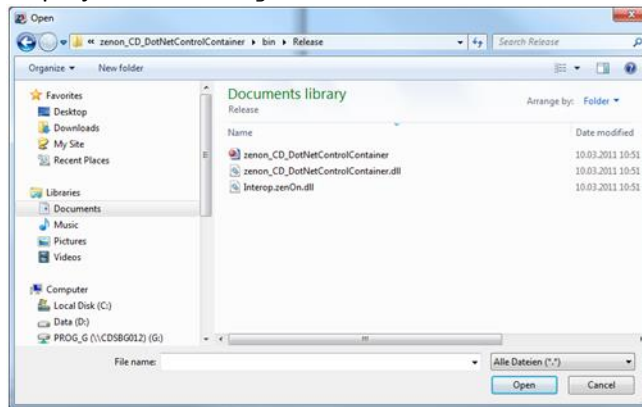
5.2.3 add a CD_DotNetControlContainer and a .NET User Control

To prepare the zenon project and to add the **CD_DotNetControlContainer** and the **.NET User Control**, carry out the following steps:

1. Create an internal variable of type `String` and set the string length to 30.

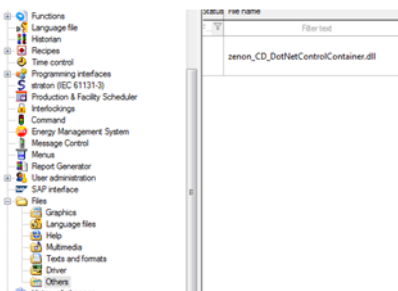


2. In the zenon project node `Project/Files/Others` add the DLL of the created .NET user



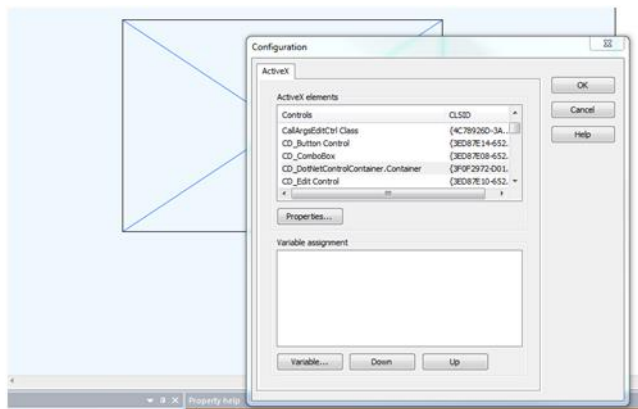
controls.

The DLL is located in the Visual Studio Project folder under `bin\Release\zenon_CD_DotNetControlContainer.dll`.



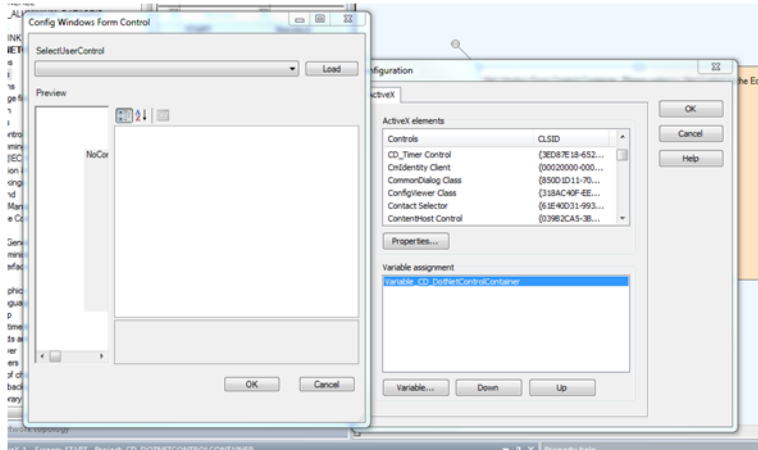
3. In the project select the ActiveX element and drag it in a zenon screen.

- The dialog **Configuration** is opened
- Select the **CD_DotNetControlContainer.Container** control.



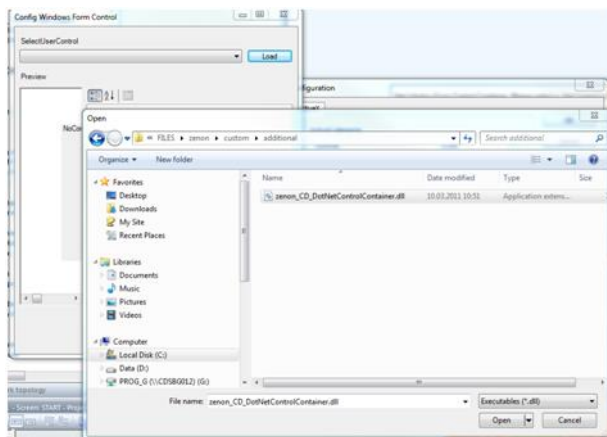
4. To embed the .NET user control in the **CD_DotNetControlContainer** control:

- Click on button **Properties**
- A new dialog is opened



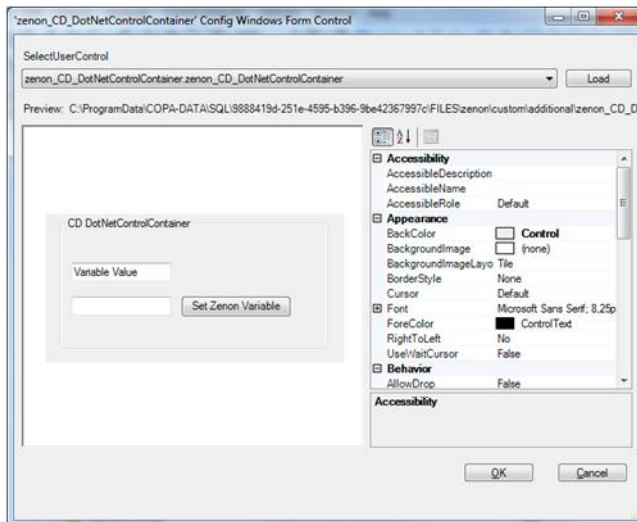
- Click on button **Load** in order to select the path of the project folder, for example:
C:\ProgramData\COPA-DATA\SQL\9888419d-251e-4595-b396-9be42367997c\FILES\zenon\custom\additional\zenon_CD_DotNetControlContainer.dll

By adding the DLL to folder `additional`, the control is automatically transferred when copying or loading the Runtime files to another computer. With this the link is lost.

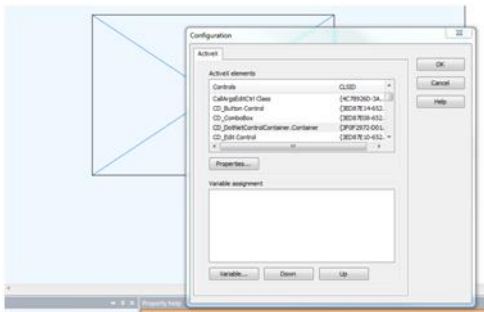


Now the .NET user control should be displayed.

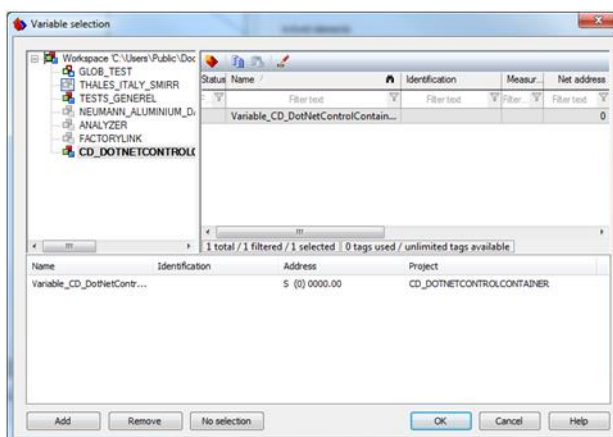
Confirm the dialog by clicking on **OK**.



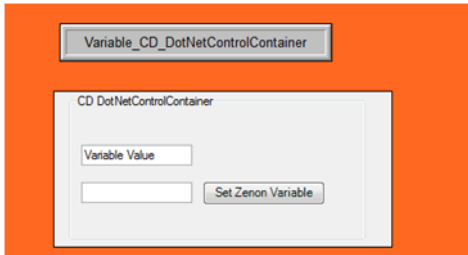
5. In the last step link a variable with the control via button **Variables**.



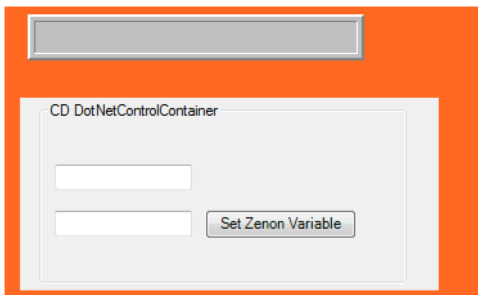
The variable selected first is automatically linked with our globally defined variable (.NET UserControl) via **public** method **zenonInit**. The linking with the control is carried out after the Runtime start.



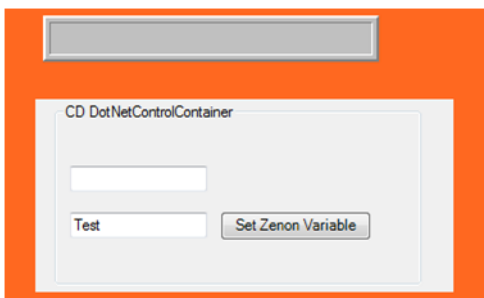
Then link the internal variable with a text element.



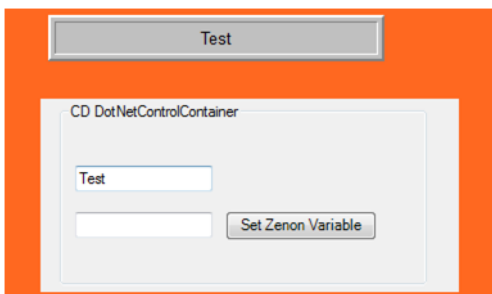
6. After the Runtime start the control is initially empty.



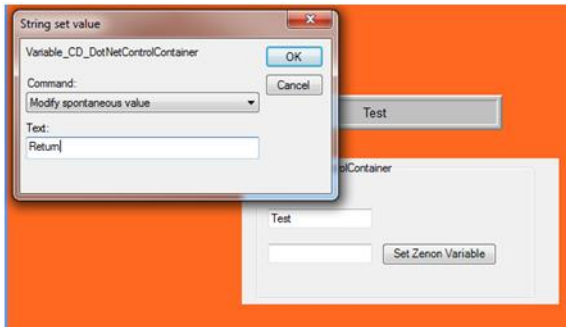
If you enter a value in the second text box and then confirm it with button **Set zenon variable**, the value is written to the zenon variable. (The **btnSetZenonVariable_Click** event is carried out.)



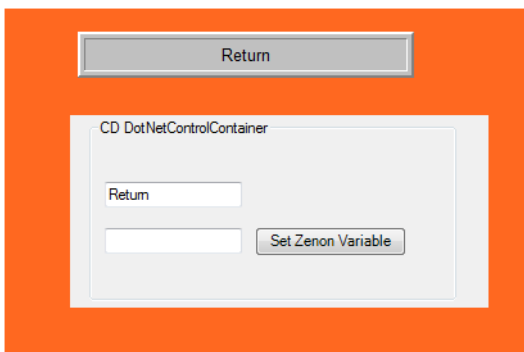
This is also displayed in the zenon text element.



If the value is directly changed in the zenon text element,



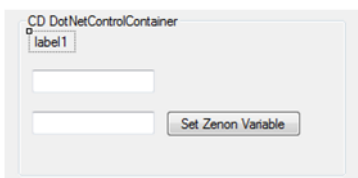
the value is directly written in the first text box via the **Paint** event of the .NET control.



5.2.4 Accessing the user control via VSTA or VBA

This examples shows the access via VSTA. The procedure is the same as with VBA.

1. Enhance the control with a label (**label**) and name it **lblzenonInfo**. In this label the value of another zenon variable should be displayed. The new value should be set via a VSTA macro.



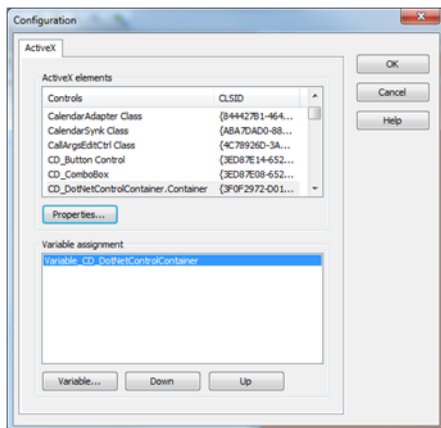
2. Enhance the code by a property (**Information**) and add the properties **get** and **set** to the property. They allow you to read and write the text of the label.

```

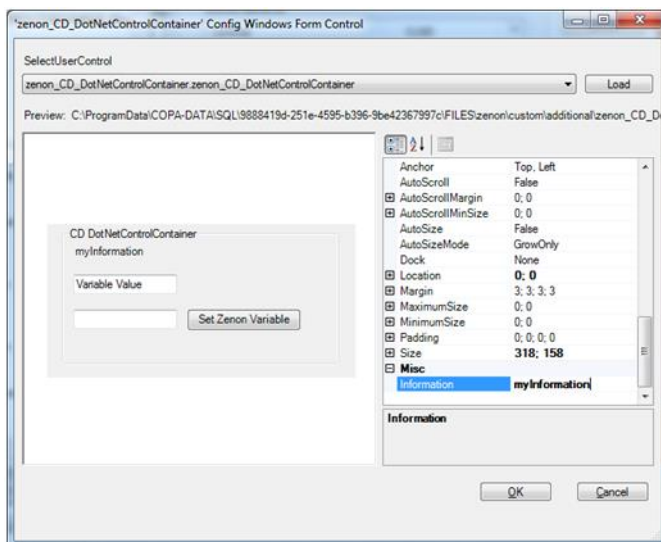
13 public partial class zenon_CD_DotNetControlContainer : UserControl
14 {
15     //This will be needed to get the zenon Variable Container
16     zenOn.Variable _cVal = null;
17
18     public zenon_CD_DotNetControlContainer()
19     {
20         InitializeComponent();
21     }
22
23     public string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

```

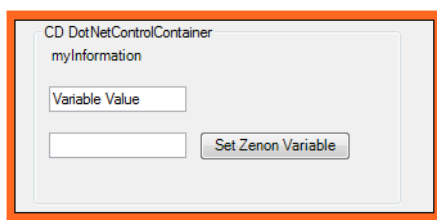
3. Create a new release for our user control and copy it to folder `additional` of the zenon project.
Do not forget: Close the zenon Editor before you do this!
Delete the old DLL and restart the zenon Editor. If the DLL is still in the folder, just delete it a second time. Now you can import the changed DLL. The **CD_DotNetContainerControl** and the ActiveX are updated automatically.
4. In the zenon Editor click on the ActiveX and open the property window.



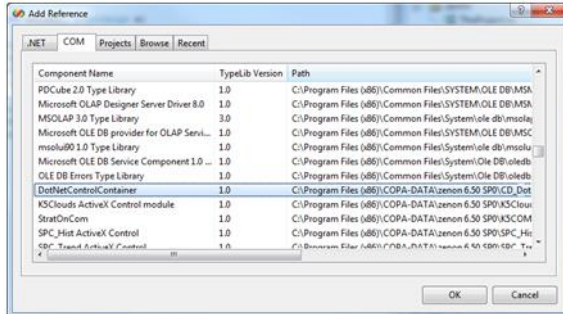
Now you can see the new property **Information** in the selection window of the control and you can also set a value.



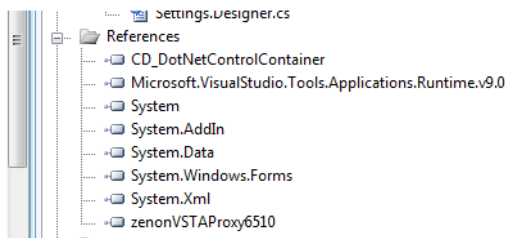
This value is also set in the control ("myInformation")



5. In order to be able to work with the **CD_DotNetControlContainer** in VSTA or VBA, you first need the reference to the control. After VSTA has been opened for the project (**ProjectAddin**), you must add the reference of the **CD_DotNetControlContainer**.



In addition you must also add the Assembly **System.Windows.Forms**.



6. With the following code you can set the value of our property **Information** anew.

```
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 ActiveX Element
        // with a cast
        CD_DotNetControlContainer.Container zActiveX = (CD_DotNetControlContainer.Container)zElement.ActiveX();

        //With using SetExternalUserControlProperty and the name of the Property "Information" we can set
        // a new Value "New Information" to the Property
        if (zActiveX.GetExternalUserControlProperty("Information").Equals("myInformation"))
        {
            zActiveX.SetExternalUserControlProperty("Information", "New Information");
        }
        else
        {
            zActiveX.SetExternalUserControlProperty("Information", "myInformation");
        }
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.Print("ERROR : " + ex.Message + " " + ex.Source);
    }
}
```

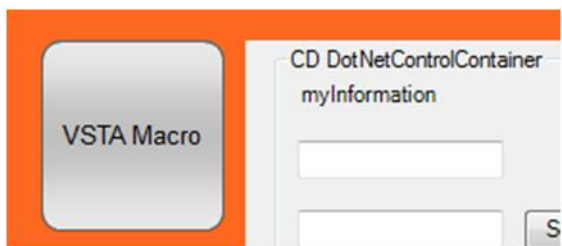
7. Finally:

- create a new zenon function **Execute VSTA macro**
- link the function to a button

In the Runtime the label is changed from **myInformation** to **New Information** by clicking on the button.



And back when you click the button again.



5.3 Example :NET control as ActiveX (C#)

The following example describes a .NET control which is executed as ActiveX control in zenon.

The creation and integration is carried out in four steps:

1. Create Windows Form Control (on page 32)
2. Change .NET User Control to dual control (on page 35)
3. Work via VBA with ActiveX in the Editor (on page 39)
4. Connect zenon variables with the .NET user control (on page 40)



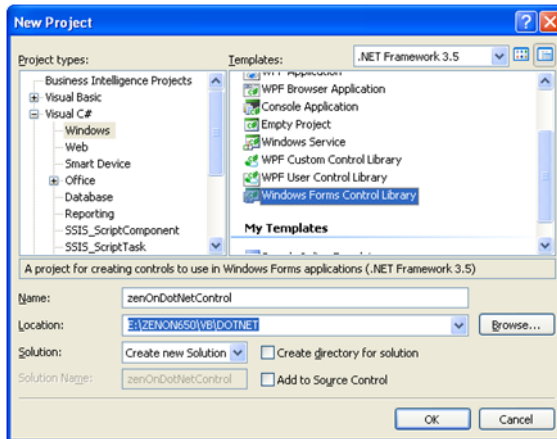
Information

The screenshots for this theme are only available in English.

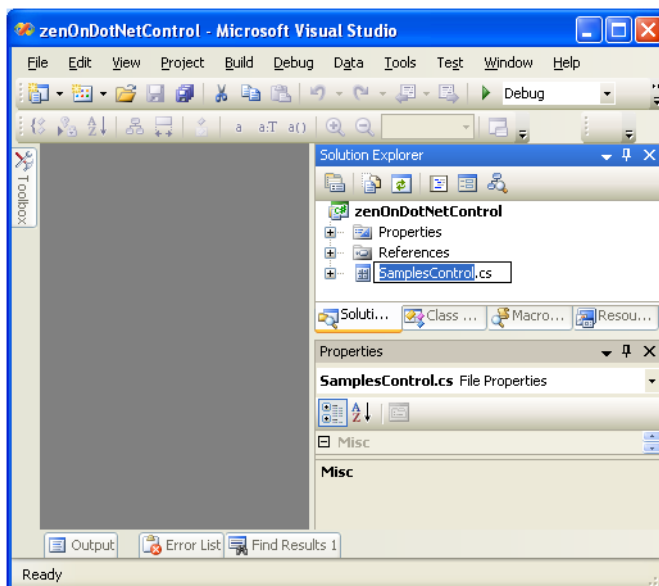
5.3.1 Create Windows Form Control

To create a Windows Form Control:

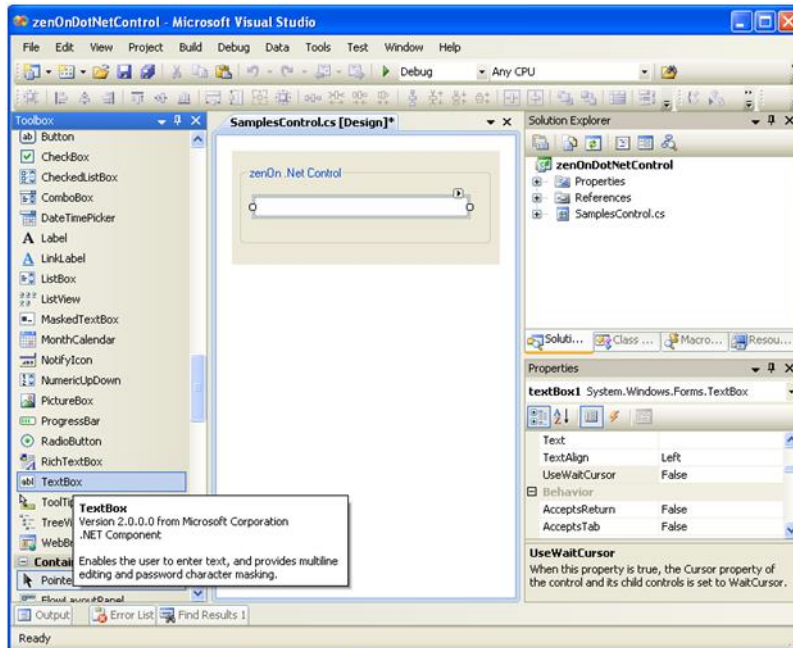
1. Start Visual Studio 2008 and create a new Windows **Form Control Library** project:



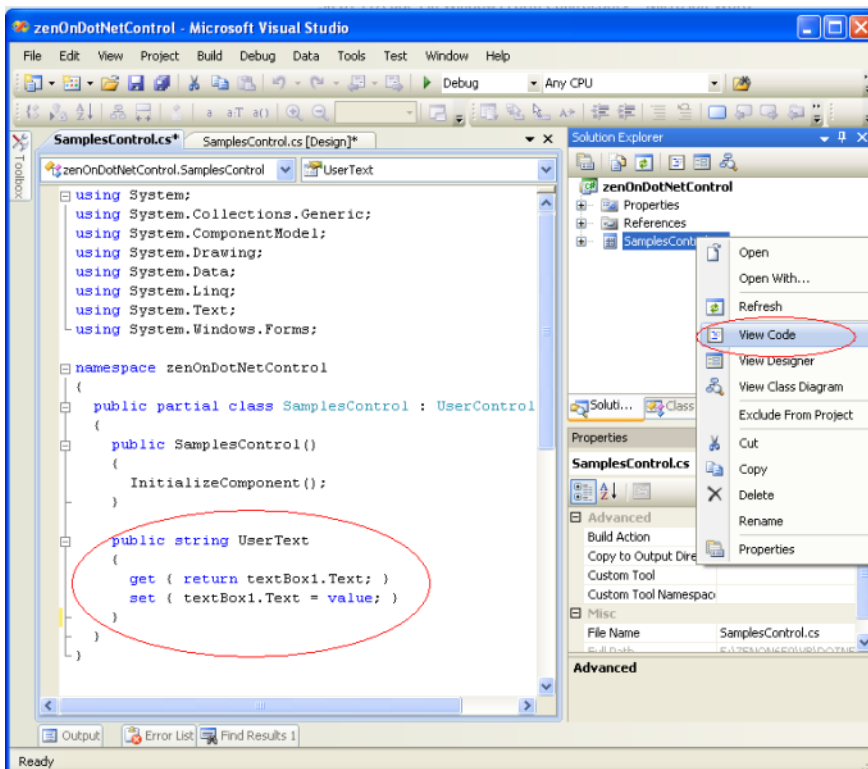
2. Rename the default control to the desired control name.
In our example: **SampesControl.cs**.



3. Open the Control Designer and add the desired control; in our case a text box:



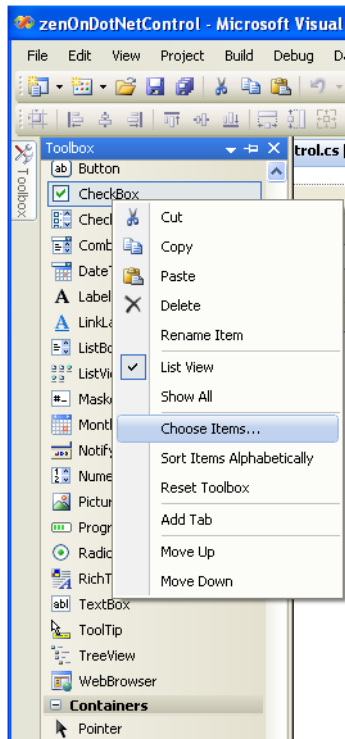
4. Normally controls have properties. Open the Code Designer via **View Code** and add the desired properties which should be available externally.
In our example: Externally visible property „**UserText**“ with **get** and **set** access which contains the text of the text box:



5. Compile the project.

The Windows Forms Control can now be used in other Windows Forms projects.

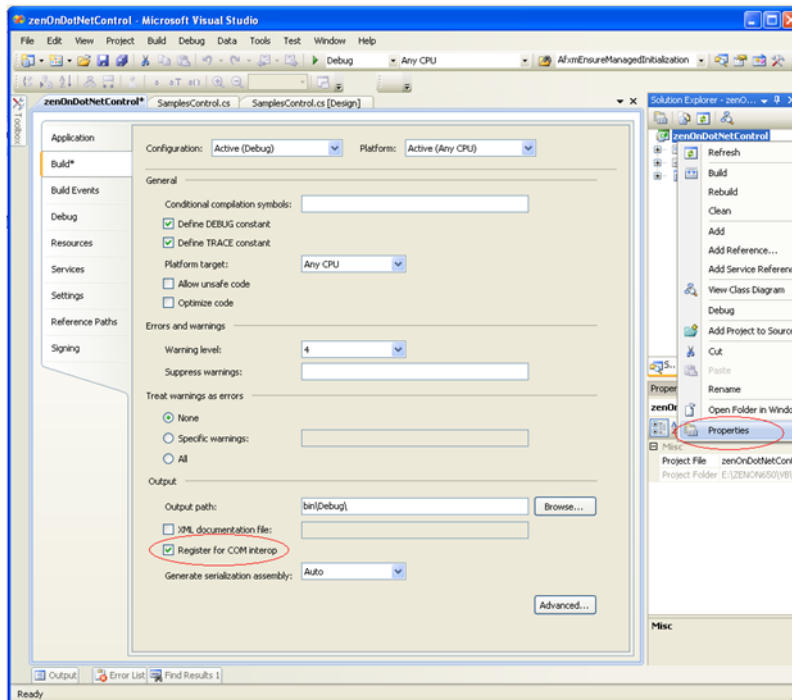
Important: The control must be inserted manually in the control tool box via **Choose Items**.



5.3.2 Change .NET User Control to dual control

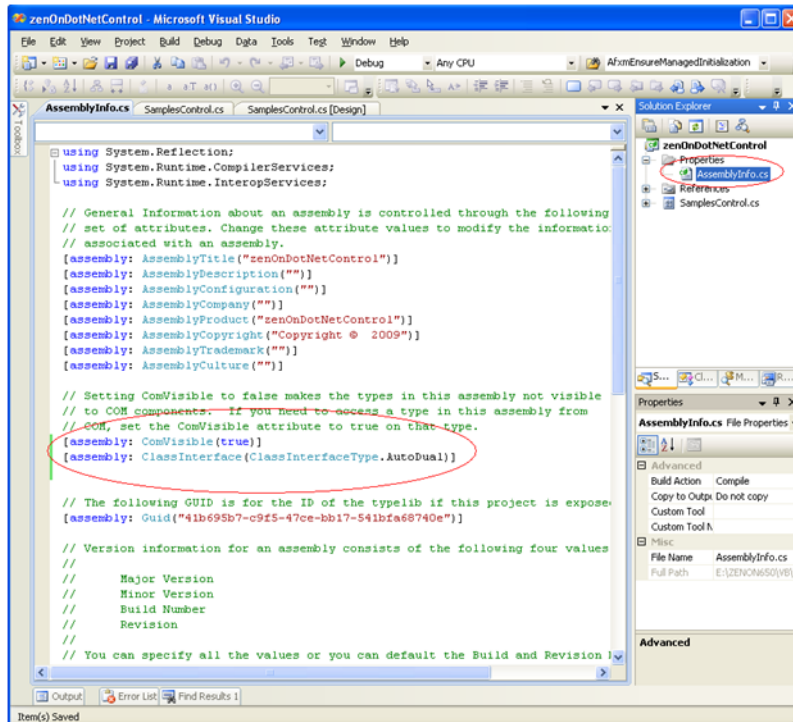
To change the .NET in a dual control, you must first activate the COM interface for ActiveX.

1. Open the project and activate property **Register for COM interop** in the **Build** settings:

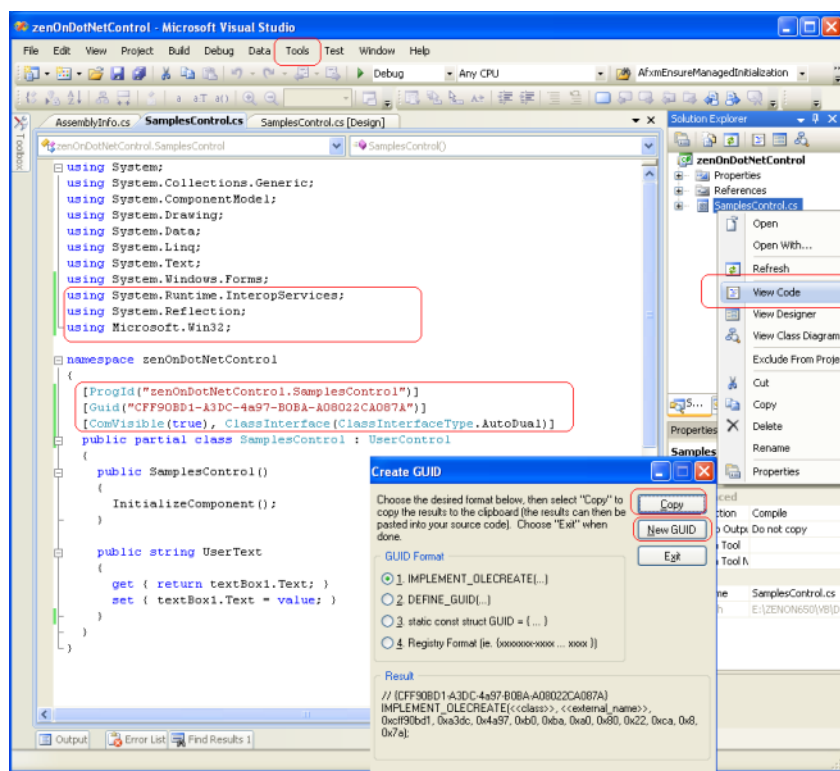


2. Open the file **AssemblyInfo.cs** and
 - set attribute **ComVisible** to `true`
 - add attribute **ClassInterface**
- [assembly: ComVisible(`true`)]

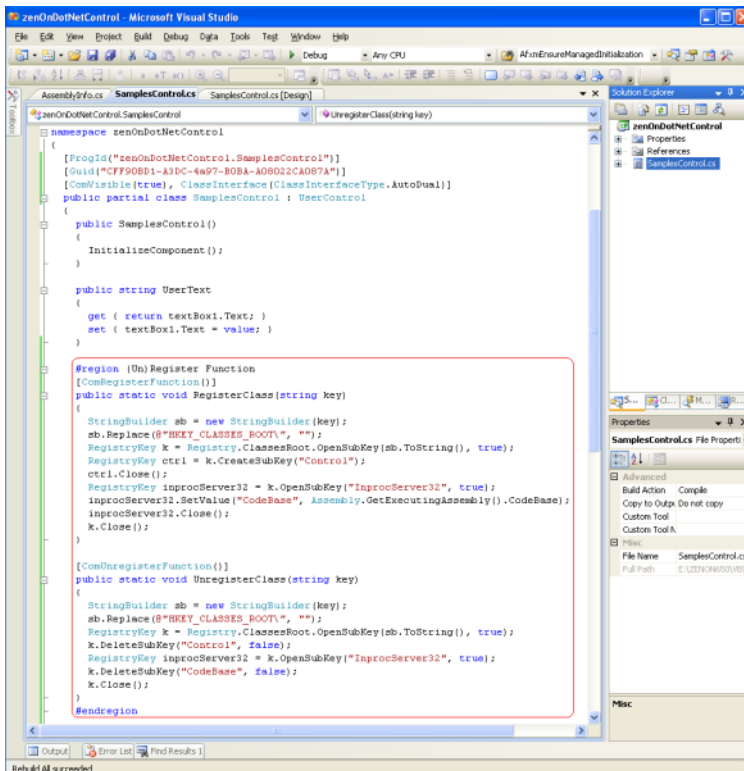
[assembly: ClassInterface(ClassInterfaceType.AutoDual)]



3. Open the code designer via **View Code** and add the necessary ActiveX attributes and **using** entries. Via menu **Tools/Create GUID** create a new GUID for the GUID attribute:



4. For the control to be selectable as Active X user interface control, you must add the functions to the following control classes:
 - RegisterClass
 - UnregisterClass



After that you can register the control in the registry.

5. Compile the project again.

The Windows Form Control is now ActiveX-able and was registered automatically during the rebuild. An additional typelib file **zenOnDotNetControl.tlb** was created in the output directory.

6. To use the control on another computer:

a) copy the DLL file and the TLB file to the target computer

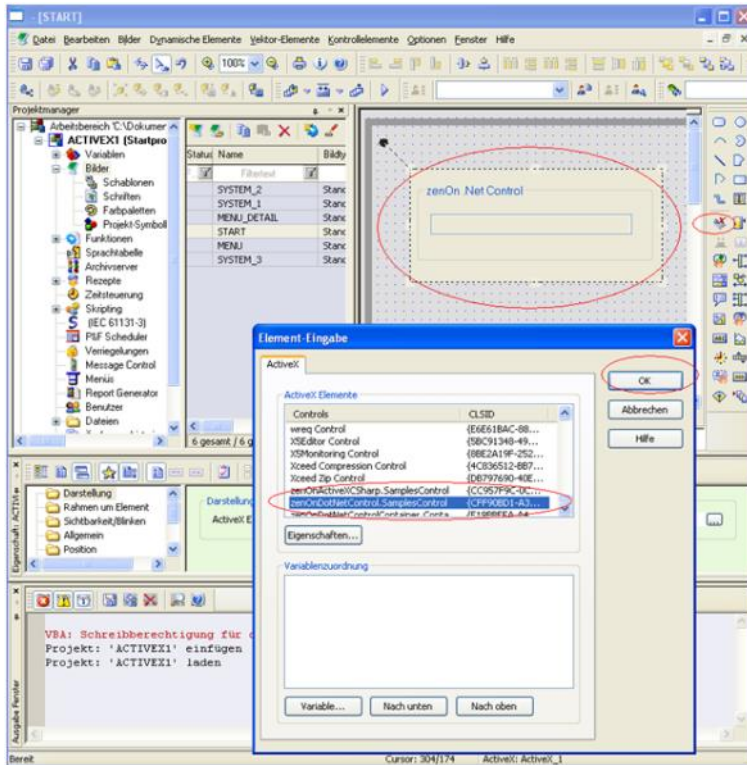
b) register the files via the command line:

```

%windir%\Microsoft.NET\Framework\v2.0.50727/regasm.exe zenOnDotNetControl.dll
/tlb:zenOnDotNetControl.tlb

```

7. Add the extended Windows Form Control as ActiveX control to the zenon Editor:



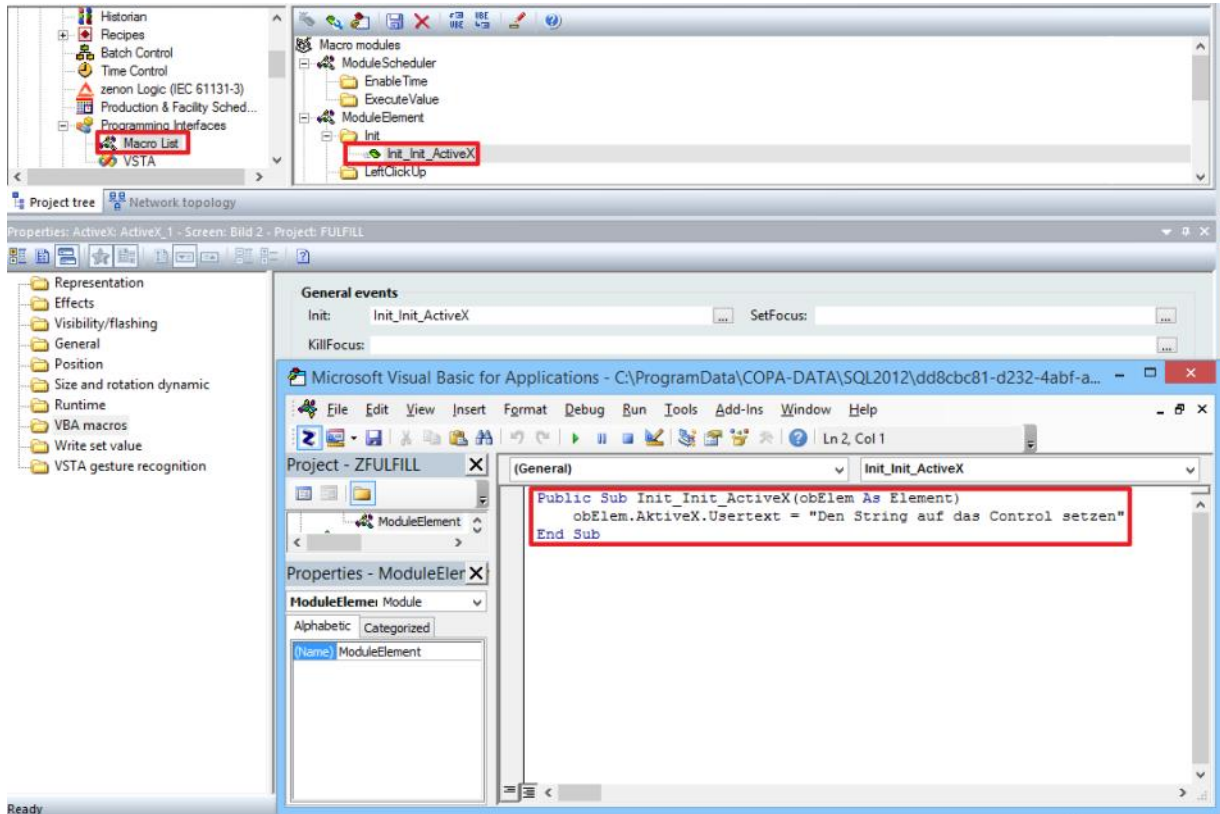
5.3.3 Work via VBA with ActiveX in the Editor

To access the properties of the control in the zenon Editor:

1. In the zenon Editor in node **Programming interfaces/VBA macros** create a new **Init** macro with the name **Init_ActiveX**.

In this macro you can access all external properties via **obElem.ActiveX**.

2. Assign his macro to the ActiveX control via properties **VBA macros/Init** of the ActiveX element.



EXAMPLE INIT MACRO

```
Public Sub Init_ActiveX(obElem As Element)
    obElem.ActiveX.UserText = "Set the string to the control"
End Sub
```

5.3.4 Connect zenon variables with the .NET user control

In zenon you have the possibility to enhance an ActiveX control with special functions in order to access the zenon API.

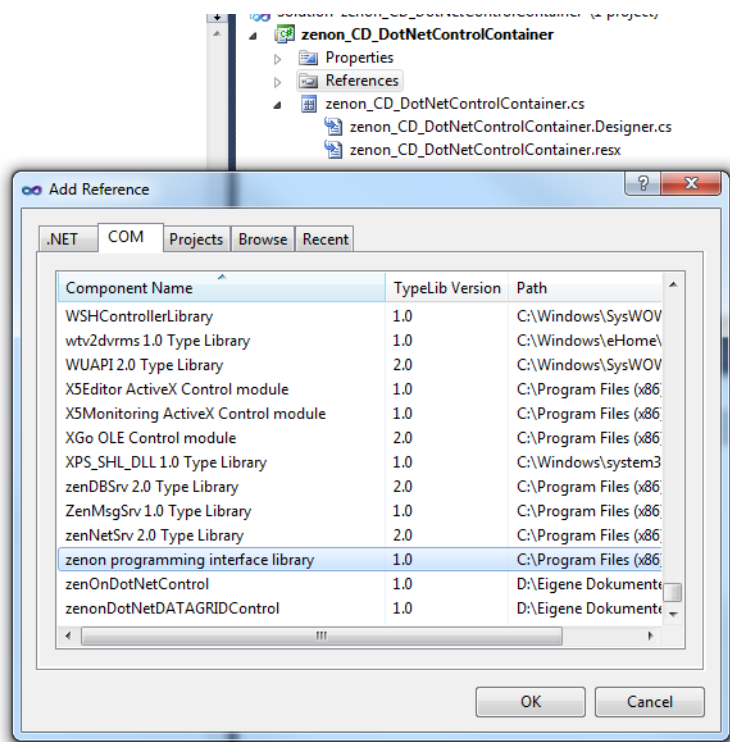
NECESSARY METHODS

- ▶ public bool zenOnInit (on page 42) (Is called up during control initializing in the zenon Runtime.)
- ▶ public bool zenOnInitED (on page 42) (Is used in the Editor.)
- ▶ public bool zenOnExit() (on page 43) (Is called up during control destruction in the zenon Runtime.)

- ▶ public bool zenOnExitED() (on page 43) (Is used in the Editor.)
- ▶ public short CanUseVariables() (on page 43) (Supports linking variables.)
- ▶ public short VariableTypes() (on page 43) (Supported data types by the control)
- ▶ public MaxVariables() (on page 44)(Maximum number of variables which can be linked to the control.)

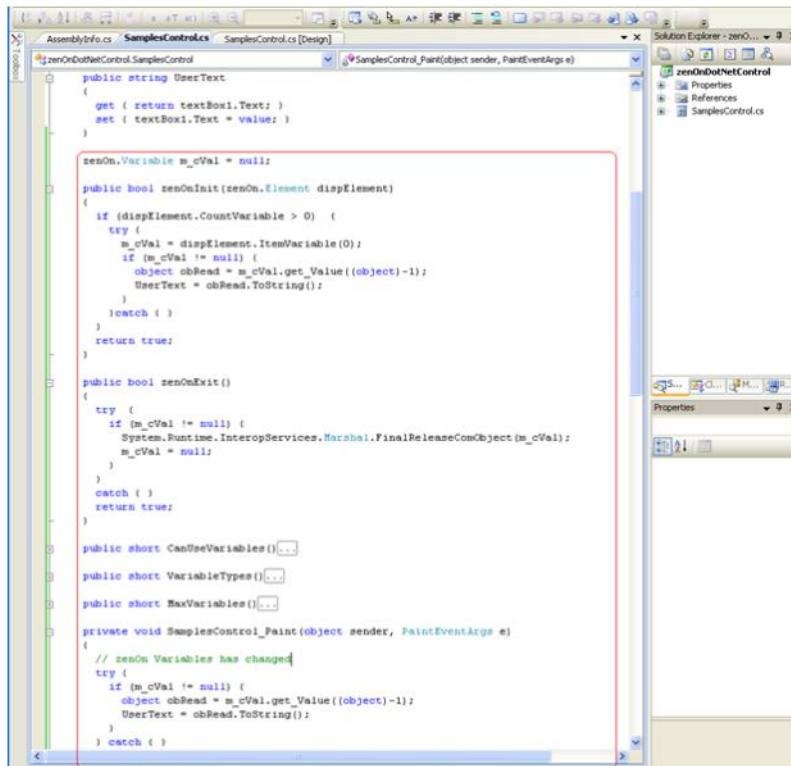
ADD REFERENCE

1. Select in Microsoft Visual Studio under **Add References** the zenon Runtime object library in order to be able to access the zenon API in the control.



2. Add the enhanced functions in the class code of the control in order to access the whole zenon API.

In our example the COM object of a zenon variable is temporarily saved in a **Member** in order to access it later in the **Paint** event of the control.



public bool zenOnInit(zenOn.Element dispElement)

With this method (in the Runtime) the ActiveX control gets a pointer to the dispatch interface of the dynamic element. With this pointer zenon variables linked to the dynamic element can be accessed.

You can configure the sequence of the sent variables in the Enter Element dialog with the buttons **down** or **up**. The dialog "element input" opens if:

- ▶ you double click the ActiveX element or
- ▶ select **Properties** in the context menu or
- ▶ select the **ActiveX settings** property in the **Representation** node of the property window

public bool zenOnInitED(zenOn.Element dispElement)

Equals public bool zenOnInit (on page 42) and is executed when opening the ActiveX in the Editor (double click on ActiveX).

public bool zenOnExit()

This method is called by the zenon Runtime when the ActiveX control is closed. Here all dispatch pointers on variables should be released.

public bool zenOnExitED()

Equals public bool zenOnExit() (on page 43) and is executed in closing the ActiveX in the Editor. With this you can react to changes, e.g. value changes, in the Editor.

public short CanUseVariables()

This method returns 1 if the control can use zenon variables and 0 if it cannot.

- ▶ 1: For the dynamic element (via button **Variable**) you can only state zenon variables with the type stated via method **VariableTypes** in the number stated by method **MaxVariables**.
- ▶ 0: If **CanUseVariables** returns 0 or the control does not have this method, any number of variables of all types can be defined without limitations. In the Runtime however they only can be used with VBA.

public short VariableTypes()

The value returned by this method is used as a mask for the usable variable types in the variable list. The value is an **AND** relation from the following values (defined in **zenon32/dy_type.h**):

Parameters	Value	Description
WORD	0x0001	corresponds to position 0
BYTE	0x0002	corresponds to position 1
BIT	0x0004	corresponds to position 2
DWORD	0x0008	corresponds to position 3
FLOAT	0x0010	corresponds to position 4
DFLOAT	0x0020	corresponds to position 5
STRING	0x0040	corresponds to position 6
IN_OUTPUT	0x8000	corresponds to position 15

public MaxVariables()

Here the number of variables is defined, that can be selected from the variable list:

1: Multi-select is disabled in the variable list. A warning is displayed when several variables are selected anyway.

6. WPF element

With the **WPF** dynamic element, valid WPF/XAML files in zenon can be integrated and displayed.



Information

All brand and product names in this documentation are trademarks or registered trademarks of the respective title holder.

6.1 Basics

XAML

XAML stands for **Extensible Application Markup Language**. The XML-based descriptive text developed by Microsoft defines graphic elements, animations, transformations, displays of color gradients etc. in Silverlight and WPF user interfaces. The use of XAML makes it possible to strictly separate design and programming. The designer prepares, for example, the graphical user interface and creates basic animations that are then used by the developers/project planners who create the application logic.

WPF

WPF stands for Windows Presentation Foundation and describes a graphics framework that is part of the Windows .NET framework:

- ▶ WPF provides a comprehensive model for the programmer.
- ▶ XAML describes, based on XML, the interface hierarchy as a markup language. Depending on the construction of the XAML file, there is the possibility to link properties, events and transformations of WPF elements with variables and functions of CD_PRODUCTNAME<.

- The framework unites the different areas of presentation such as user interface, drawing, graphics, audio, video, documents and typography.

For execution in zenon, Microsoft .NET framework version 3.5 or higher is required.

6.1.1 WPF in process visualization

XAML makes different design possibilities possible for zenon. Display elements and dynamic elements can be adapted graphically regardless of the project planning. For example, laborious illustrations are first created by designers and then imported into zenon as an XAML file and linked to the desired logic. There are many possibilities for using this, for example:

DYNAMIC ELEMENTS IN ANALOG-LOOK



Graphics no longer need to be drawn in zenon, but can be imported directly as an XAML file. This makes it possible to use complex, elaborately illustrated elements in process visualization. Reflections, shading, 3D effects etc. are supported as graphics. The elements that are adapted to the respective industry environment make intuitive operation possible, along the lines of the operating elements of the machine.

INTRICATE ILLUSTRATIONS FOR INTUITIVE OPERATION



The integration of XAML-based display elements improves the graphics of projects and makes it very easy to display processes clearly. Elements optimized for usability make operation easier. A clear display of data makes it easier to receive complex content. The flexible options for adapting individual elements makes it easier to use for the operator. It is therefore possible for the project planners to determine display values, scales and units on their own.

CLEAR PRESENTATION OF DATA AND SUMMARIES



Grouped display elements make it possible to clearly display the most important process data, so that the equipment operator is always informed of the current process workflow. Graphical evaluations, display values and sliders can be grouped into an element and make quick and uncomplicated control possible.

INDUSTRY-SPECIFIC DISPLAYS



Elements such as thermometers, scales or bar graphs are part of the basic elements of process visualization. It is possible, using XAML, to adapt these to the respective industry. Thus equipment operators can find the established and usual elements that they already know from the machines in process visualization at the terminal.

ADAPTATION TO CORPORATE DESIGN



Illustrations can be adapted to the respective style requirements of the company, in order to achieve a consistent appearance through to the individual process screen. For example, the standard operation elements from zenon can be used, which can then be adapted to color worlds, house fonts and illustration styles of the corporate design.

6.1.2 Transfer of values from zenon to WPF

zenon always works internally with `Double` or `String` data types. These are sent to the WPF element. The WPF element is embedded in a .NET container. It usually needs to be converted so that the data type can be used. This conversion can automatically be carried out by .NET.

The values are sent in accordance with the following rules:

- ▶ If the .NET type (**`System.Object`**) for zenon is not evident, the value is sent as it is to .NET. .NET must take care of the display or conversion itself.
- ▶ If the .NET type is a Boolean type (**`System.Boolean`**), then zenon writes according to the .NET convention 0 or -1.
- ▶ If the .NET type is known, a check is carried out to see if .NET can convert the value. The converter from .NET is used for this.
 - Yes: The value is sent.
 - No: The value is sent nevertheless. If .NET reacts with an error message, the value of zenon is converted into a string and sent again.

INTERLOCKING

If a value change is forwarded to the WPF content, an interlocking ensures that no Property changes can be forwarded to zenon. This interlocking works at Property level.



Example

The following are present in the WPF:

- ▶ **Property1 (Element1)**
- ▶ **Property1 (Element2)**

If **Property1 (Element1)** of zenon is updated, the change notifications from this Property and for this element are blocked. However, direct linking of **Property1 (Element1)** to **Property1 (Element2)** leads to an updating of the linking of **Property1 (Element2)**.

6.1.3 Referenced assemblies

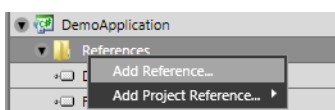
It is not just standard objects (rectangles, graphics, etc.) or effects (color gradients, animations, etc.) that can be displayed using the **WPF elements**, but also customized user controls (with logic in the code behind), which are referenced as assemblies.

For example, a user control that looks like a tachometer and provides special properties and optical effects can be created, such as a "Value" property, which causes the pointer of the tachometer to move and/or the corresponding value to be displayed in a label.

The workflow for this:

- ▶ The appearance of a user control is labeled with standard objects, which are offered by WPF.
- ▶ The properties and interactions are programmed.
- ▶ The whole package is compiled and present in the form of a .NET assembly.

This assembly can also be used for WPF projects. To do this, it must be referenced (linked) in the WPF editor (for example: Microsoft Expression Blend). To do this, select the assembly in the zenon file selection dialog:



From this point in time, the WPF user controls of the assembly in the tool box can be selected under **Custom user controls** and used in the WPF project.

See also, in relation to this, the following chapter: Guidelines for developers.

USED REFERENCED ASSEMBLIES IN ZENON

To use an assembly in zenon, this must be provided as a file.

Collective files in **.cdwmpf** format administer these independently; no further configuration is necessary.

Assemblies must be added to the `Files` folder for `.xaml` files:

- ▶ Click on `Files` on the project tree
- ▶ Select `Other`
- ▶ Select **Add file...** in the context menu
- ▶ The configuration dialog opens
- ▶ Insert the desired assembly

When displaying a WPF file in the **WPF element** (Editor and Runtime), the assemblies from this folder are loaded. It is thus also ensured that when the Runtime files are transferred using **Remote Transport**, all referenced assemblies are present on the target computer.

A collective file (**.cdwmpf**) can exist alongside an XAML file with the same name. All assemblies (*.dll) from all collective files and the `Other` folder are copied to the work folder. Only the highest file version is used if there are several assemblies with the same name.



Attention

Assemblies are only removed after loading when the application is ended. This means:

If a WPF file with a referenced assembly in zenon is displayed, then this assembly is loaded in the memory until zenon is ended, even if the screen is closed again. If you would like to remove an assembly from the `Files/Other` folder, the Editor must first be restarted, so that the assembly is removed.

MULTI-PROJECT ADMINISTRATION

With multi-project administration, the same assembly must be used in all projects. If an assembly is replaced by another version in a project, it must also be replaced in all other projects that are loaded in the Editor or in Runtime.

6.1.4 Workflows

The WPF/XAML technology makes new workflows in process visualization possible. The separation of design and functionality ensures a clear distinction of roles between the project engineer and designers; design tasks can be easily fulfilled by using pre-existing designs, which no longer need to be modified by the project engineer.

The following people are involved in the workflow to create WPF elements in zenon:

- ▶ Designer

- illustrates elements
- takes care of the graphics for MS Expression Design
- ▶ MS Expression Blend operator
 - Animates elements
 - Creates variables for the animation of WPF elements in zenon, which project engineer can access
- ▶ Project engineer
 - Integrates elements into zenon:
 - stores logic and functionality

We make a distinction:

- ▶ Workflow with Microsoft Expression Blend (on page 80)
- ▶ Workflow with Adobe Illustrator (on page 80)

Workflow with Microsoft Expression Blend

When using Microsoft Expression Blend, a WPF element is created in four stages:

1. Illustration of elements in **MS Expression Blend** (on page 81)
2. Open element in **MS Expression Design** and export as WPF
3. Animation in **MS Expression Blend** (on page 81)
4. Integration into zenon (on page 145)

You can find an example for creating a WPF elements with Microsoft Expression Blend in the Create button as XAML file with Microsoft Expression Blend (on page 81) chapter.

Workflow with Adobe Illustrator

Based on traditional design processes with **Adobe Illustrator** the following workflow is available:

1. Illustration of elements in **Adobe Illustrator** (on page 86)
2. Import of **.ai** files and preparation in **MS Expression Design** (on page 87)
3. WPF export from **MS Expression Design** (on page 87)
4. Animation in **MS Expression Blend** (on page 89)
5. Integration into zenon (on page 139)

You can find an example for creation in the Workflow with Adobe Illustrator (on page 85) chapter.

6.2 Guidelines for designers

This section informs you how to correctly create WPF files in Microsoft Expression Blend and Adobe Illustrator. The tutorials on Creating a button element (on page 81) and a bar graph element (on page 85) show you how fully functional WPF files for zenon can be created from pre-existing graphics in a few steps.

The following tools were used for this:

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



Information

If referenced objects (assemblies) are used in WPF, note the instructions in the Referenced objects (on page 78) chapter.

6.2.1 Workflow with Microsoft Expression Blend

With Microsoft Expression Blend, a WPF element:

- ▶ is illustrated
- ▶ is converted into WPF format using **MS Expression Design**
- ▶ animated

The following example shows the illustration and conversion of a button element into an XAML file.

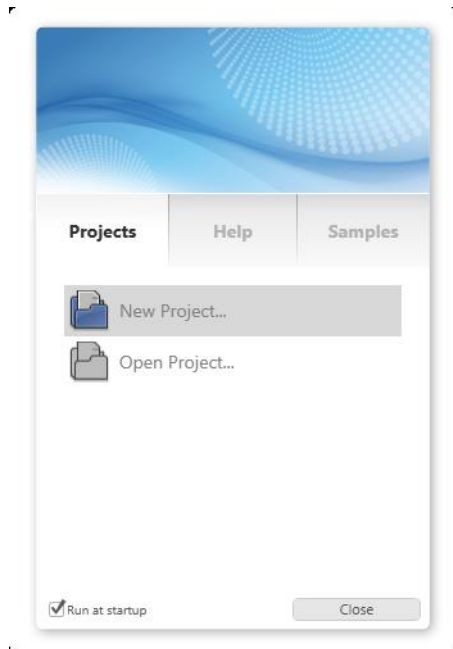
Note: A test version of "Microsoft Expression Blend" can be downloaded from the Microsoft website.

Create button as an XAML file with Microsoft Expression Blend

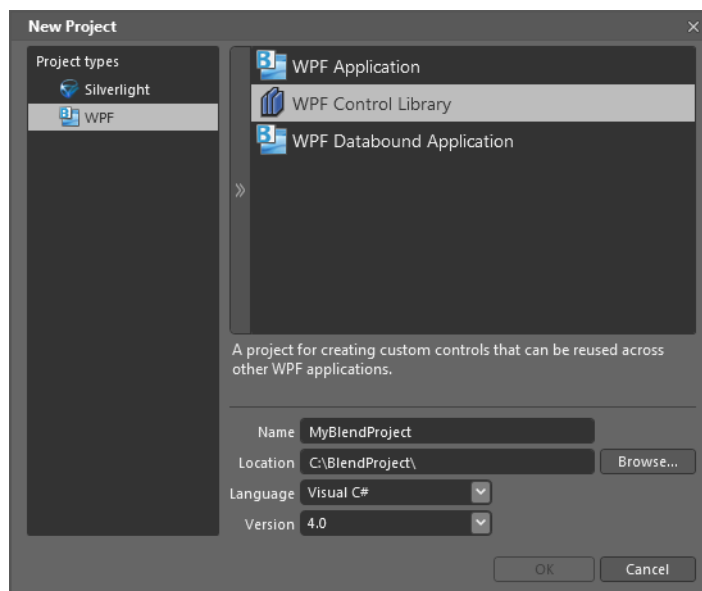
CREATE BUTTON

1. Start Expression Blend

2. select the **New Project** option



3. Select **WPF** as project type
4. give it a path and name of your choice (MyBlendProject, for example)



The **Language** and **Version** settings can be ignored, because no functionality is to be programmed.

5. After the dialog has been confirmed with **OK**, Microsoft Blend creates a new project with the chosen settings. Expression Blend adds an empty XAML file which already contains a class reference.
6. Delete the CS file that belongs to the XAML file using the context menu.

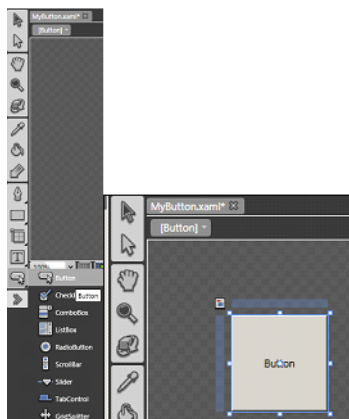
7. Rename the XAML file **MainControl.xaml** to **MyButton.xaml**.
8. The development size of the file is set at 640 x 480 pixels as standard and must still be changed:
 - a) switch to **XAML** view
 - b) correct the size to 100 x 100 pixels
 - c) Delete the class reference `x:Class="MyBlendProject.MyButton"`

```

MyButton.xaml x
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>

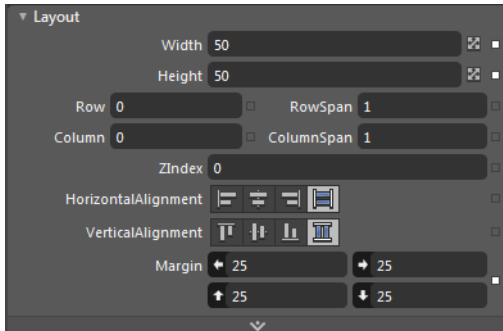
```

9. switch to **Design** view

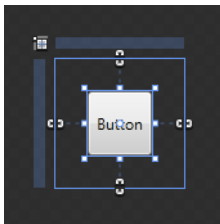


10. add a button via the toolbar
11. define the properties
 - Width: 50
 - Height: 50

- Margins: 25



The button is therefore at the center of the control.



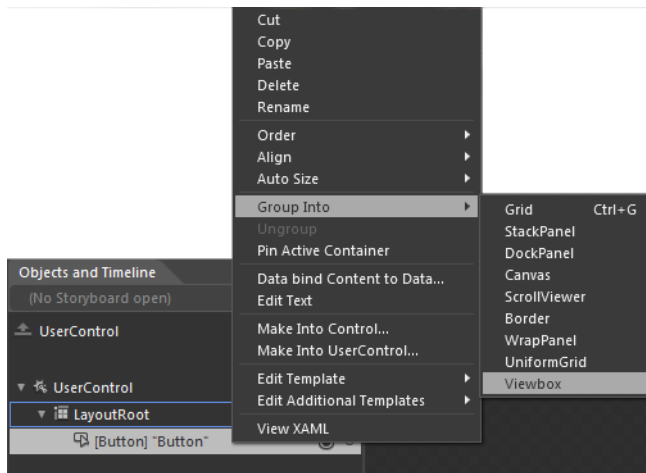
12. Save the changes and open the file in Internet Explorer to check it. You will see that the button is displayed in a size of 50 x 50 pixels.

MAKE BUTTON SCALABLE

If you integrate this status into zenon, the button will always have the exact size of 50 x 50 pixels. Because the button can be implemented as a scalable button, switch to Expression Blend again:

1. select the button in the tree view
2. select the Group Into->Viewbox button in the context menu
3. the button is inserted into a **Viewbox**
4. Define the properties of the viewbox
 - Width: Auto
 - Height: Auto

5. save the file



6. If you now open the file in Internet Explorer, the button is automatically scaled when the IE window size is changed. This file will now also automatically adapt to changes in the size of the **WPF element** in zenon.

CHANGE NAME

Before you can integrate the file into zenon, you must give the **WPF element** a name. The **WPF elements** are not named in Expression Blend as standard, and are labeled with square brackets and their type. zenon content is assigned to WPF content via the name of the **WPF elements**:

- ▶ in tree view, change the name
 - of the button on **MyButton**
 - of the ViewBox to **MyViewBox**

This button can now be integrated in zenon (on page 145) as an XAML file.

6.2.2 Workflow with Adobe Illustrator

When **Adobe Illustrator** is used, a WPF element:

- ▶ is illustrated in **Adobe Illustrator**
- ▶ is converted into a WPF in **MS Expression Design**
- ▶ is animated in **MS Expression Blend**

The following example shows the illustration and conversion of a bar graph element into an XAML file.

Bar graph illustration

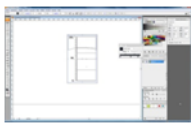
A bar graph is created in Adobe Illustrator.

1. AI: Starting element for bar graph



Illustrated in Adobe Illustrator CS3.

2. AI: Path view of bar graph in Adobe Illustrator



- All effects must be converted (**Object -> Convert appearance**)
- All lines are transformed into paths (**Object -> Path -> Contour line**)
- Do not use filters such as shading, blurring etc.

NOTES ON COMPATIBILITY

Illustrations that were created with Adobe Illustrator are in principle suitable for WPF export. However, not all Illustrator effects can become corresponding effects in Expression Design/Blend. Note:

Effect	Description
Clipping masks	<p>Clipping masks created in Adobe Illustrator are not correctly interpreted by Expression Design. These are usually shown in Blend as areas of black color.</p> <p>We recommend creating illustrations without clipping masks.</p>
Filters and effects	<p>Not all Adobe Illustrator filters are transferred into Expression Design accordingly: Thus blurring filters, shading filters and corner effects from Illustrator do not work in Expression Design.</p> <p>Solution:</p> <ul style="list-style-type: none"> ▶ Most effects can be converted so that they can be read correctly by Expression Design using the Object -> Convert appearance command in Adobe Illustrator. ▶ Corner effects from Adobe Illustrator are correctly interpreted by MS Design if they are converted to AI in paths.
Text fields	<p>To be able to link text fields with code, these must be created separately in Expression Blend. "Labels" are required for dynamic texts; simple "text fields" are sufficient for static information.</p> <p>There is no possibility to create text labels in MS Design. These must be directly created in MS Blend.</p>
Transparencies and group transparencies	<p>There can be difficulties in Adobe Illustrator with the correct interpretation of transparency settings, in particular from group transparency settings.</p> <p>However MS Expression Blend and MS Expression Design do offer the possibility to create new transparency settings.</p>
Multiply levels	<p>These level settings in Adobe Illustrator are not always correctly displayed by MS Expression Blend.</p> <p>However, there is the possibility to "Multiply levels" directly in Expression Design.</p>
Indicating instruments and standard positions	<p>To prepare the graphics optimally for animation, the indicator and slider must always be set to the starting position, usually 0 or 12:00 o'clock.</p> <p>Thus the position parameters for rotations etc. are also correct in Blend and an animation can be implemented without conversion of position data.</p>

WPF export

WPF files are required for animation in Microsoft Expression Blend. We recommend Microsoft Expression Design for this export, because it provides good results and most Illustrator effects are correctly interpreted.

Note: There is a free plug-in for the direct export of WPF files from Adobe Illustrator available on the internet. This plug-in provides a quick, uncomplicated way of exporting from Illustrator, however it is less suited to the current application because it lead to graphical losses. Even color deviations from the original document are possible.

Files in **.ai** format can regularly be imported into Expression Design; the paths are retained in the process.

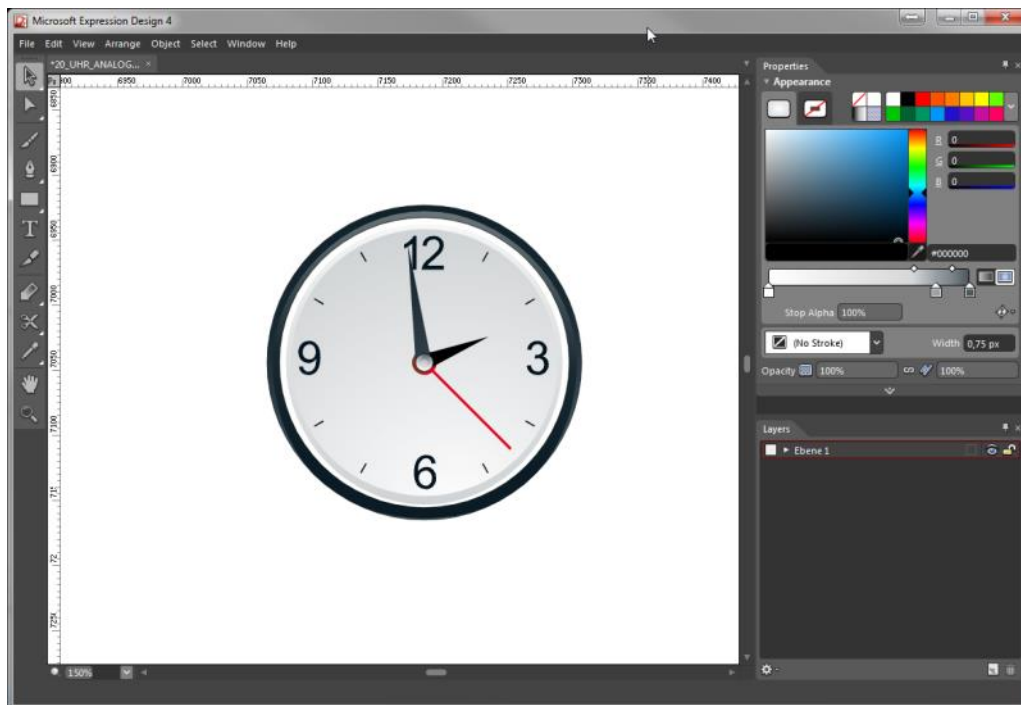
Attention: Some common Illustrator effects cannot be displayed by Expression Design correctly however (see Illustration (on page 86) chapter).

We export the pre-created bar graph element in 5 stages:

1. ED: Import

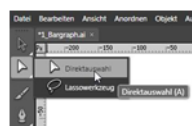
- Import the prepared Illustrator file (on page 86) in **Microsoft Expression Design** via File -> Import

2. ED: Optimization



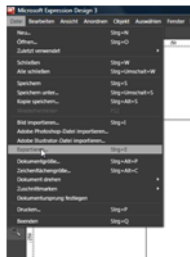
- If the starting file is not correctly displayed in MS Expression Design, it can still be subsequently edited and optimized here

3. ED: Select



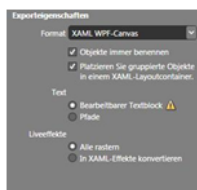
- Highlight the element for WPF export with the **direct selection** arrow in MS Expression Design; in this case it is the whole clock

4. ED: Start export



- Start the export via File -> Export
- the dialog for configuring the export settings opens

5. ED: Export settings



- Enter the following export settings:
 - Format:** XAML Silverlight 4 / WPF Canvas
 Always name objects: Activate with tick
 Place the grouped object in an XAML layout container: Activate with tick
 - Text:** Editable text block
 - Line effects:** Rasterize all

The exported file has **.xaml** file suffix. It is prepared and animated (on page 89) in MS Expression Blend in the next stage.

Animation in Blend

With MS Expression Blend:

- ▶ static XAML files from MS Expression Design are animated
- ▶ Variables for controlling effects that can be addressed by zenon are created

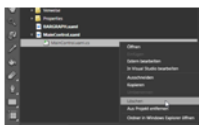
In thirteen steps, we go from a static XAML to an animated element, that can be embedded in zenon:

1. EB:create project



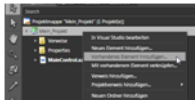
- a) Open Microsoft Expression Blend
- b) Create a new project
- c) Select the **Project type** of WPF- >WPF Control Library
- d) Give it a name (in our tutorial: **My_Project**)
- e) Select a location where it is to be saved
- f) Select a language (in our tutorial: C#)
- g) Select Framework Version 3.5

2. EB: delete MainControl.xaml.cs



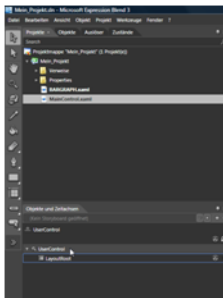
- a) Navigate to **MainControl.xaml.cs**
- b) Delete this file using the **Delete** command in the context menu

3. EB: Open exported XAML file



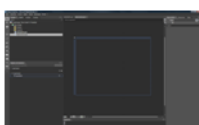
- a) Open the context menu for **My_Project** (right mouse button)
- b) Select **Add existing element...**
- c) Select the XAML file exported from Microsoft Expression Design, in order to open this in Microsoft Expression Blend

4. EB: Open MainControl.xaml



- a) Open the automatically created **MainControl.xaml**
- b) In the **Objects and Time axes** area, navigate to the **UserControl** entry

5. EB: Adapt XAML code



- a) Click on **UserControl** with the right mouse button
- b) Select **Display XAML** in the contextual menu.
- c) Delete lines 7 and 9 in the XAML code:

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

6. EB: check XAML code



- The XAML code should now look like this:

```
<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">

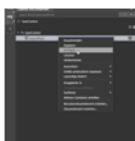
    <Grid x:Name="LayoutRoot"/>
</UserControl>
```

7. EB: Copy elements



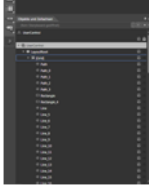
- a) Open the XAML file imported from Expression Design
- b) Mark all elements
- c) Select **Delete** in the context menu
- d) Change back to the automatically created XAML file

8. EB: Insert element



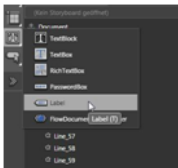
- a) Click on **Layout Root** with the right mouse button
- b) Select **Insert**

9. EB: Adapt layout type



- Click on Layout root -> Change layout type -> Viewbox with the right mouse button
- The structure should now look like this: **UserControl** -> **LayoutRoot** -> **Grid** -> **Elements**
- Give a name for **LayoutRoot** and **Grid** by double-clicking on the names

10. EB: Texts and values



- Dynamic and static texts are labeled with text fields
- Values (numbers) are issued with **Labels**

11. EB: Insert labels



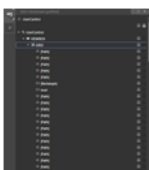
- Labels replace numbers that are to be subsequently linked using INT variables (must be carried out for all number elements)

12. EB: Set property



- To display 100%, set the bar graph element's **MaxHeight** property to 341 (the maximum height of the indicator element is 340)

13. EB: prepare for use in zenon



- Delete all name labels (names may only be given for elements that are to be addressed via zenon)

- b) Save the XAML file with any desired name
- c) Integrate the XAML file into zenon (on page 139)

A tip for checking: If the XAML file is displayed with no problems in Microsoft Internet Explorer and the window size of Internet Explorer adapts to it, it will also be correctly used in zenon.

6.3 Engineering in zenon

In order to be able to use WPF user controls in zenon, version 3.5 (or higher, depending on the .NET framework version used in the user control) of the Microsoft framework must be used on both the Editor computer and the Runtime computer.

CONDITIONS FOR THE WPF DISPLAY IN ZENON

The dynamization is currently available for simple variable types (numerical data types as well as string). Arrays and structures cannot be dynamized.

Therefore the following WPF functions can be implemented in zenon:

- ▶ Element properties that correspond to simple data types, such as `SString`, `Int`, `Double`, `Bool` etc.
- ▶ Element properties of the "Object" type, which can be set with simple data types
- ▶ Element events can be used with functions; the parameters of the events are not however available in and cannot be evaluated in zenon
- ▶ Element transformation, for which a **RenderTransform** is present for the element in the XAML file

Attention: if the content is outside of the area of the WPF element during transformation, this is not labeled

Notes on dBase: No shade can be displayed in zenon for WPF elements.



Attention

*If the Runtime files were created for a project for a version before 6.50, existing **WPF elements** are not included into Runtime screens.*

6.3.1 CDWPF files (collective files)

A CDWPF file (with the suffix ***.cdwpf**) is an renamed ZIP file that contains the following components:

- ▶ XAML file (to reference the user control assembly)
- ▶ DLL file (the actual WPF user control, optional)
- ▶ Preview graphics (for preview, optional)

Rules for the use of collective files:

- ▶ The files (XAML, DLL, preview graphics) can be in the CDWPF file directly or in a joint folder.
- ▶ The name of the collective file should correspond to the names of the XAML file.
- ▶ Only one XAML file may be contained.
- ▶ The preview graphic should be small and no more than 64 pixels high.
Name of the preview file: **preview.png** or the name of the XAML file with the suffix **png**.
- ▶ Any number of assemblies can be used. The distinction is made on the basis of the file version.
- ▶ Collective files do not need to contain an assembly.
- ▶ All subfolders are examined and only taken into account with ***.dll**, ***.xaml** or ***.png** files.
- ▶ If a collective file (***.cdwpf**) is replaced by a file with a different version, all corresponding CDWPF files in all symbols and images in all projects must be adapted.

6.3.2 create WPF element

To create a WPF element

1. In the elements toolbar, select the symbol for **WPF element** or the **Elements** entry in the menu
2. Select the start point in the main window.
3. Pull open the element with the mouse.
4. In properties, select **Representation** the property **XAML file** in the group.
5. The file selection dialog opens.
6. Select the desired file
Files of the following formats are valid:
 - *.xaml: Extensible Application Markup Language
 - *.cdwpf: WPF collective file, also shows preview image(The file must already be present in the Project Manager under **Files/graphics** or created in the dialog.)
7. Configure the links (on page 95).



Information

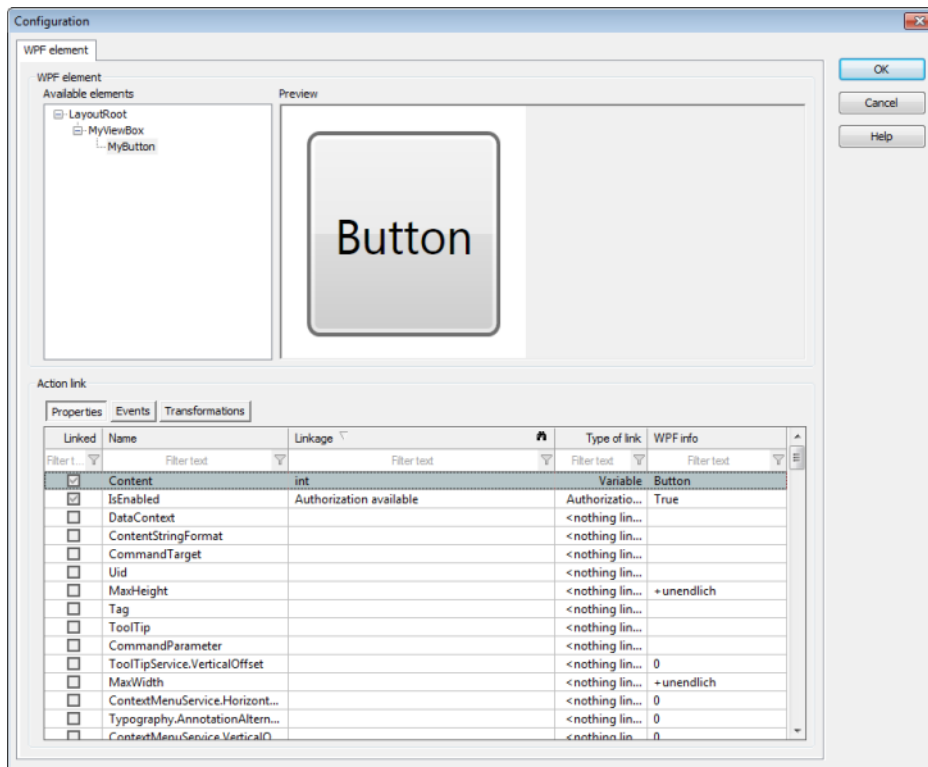
If referenced assemblies are used, note the instructions in the Referenced assemblies (on page 78) chapter.

6.3.3 Configuration of the linking

To configure a WPF element

1. In properties, select **WPF links** the property **Configuration** in the group.
2. The dialog with three tabs opens with a preview of the XAML file and the elements present in the file

DIALOG CONFIGURATION



Parameters

Available elements

Preview

Properties (on page 97)

Events (on page 103)

Transformations (on page 104)

Name

Connection

Link type

WPF info

Linked

Description

Shows the named file elements in a tree structure. The selected element can be linked with process data.

WPF is assigned to process data based on the element name. Therefore elements are only shown if they and the attendant elements have a name. Allocations are configured and shown in the **Properties, Events, Transformations** tabs.

Hint: If the corresponding elements are not displayed, check in the XAML file to see if this has a name (for example: `<Grid Name="GridName">`).

The selected element is shown flashing in the preview.

Configuration and display of properties (variables, authorizations, interlockings, linked values).

Configuration and display of events (functions).

Configuration and display of transformations.

Name of the property.

Selection of link.

Type of link (variable, authorization, function)

Shows the current value for properties in WPF content. For the user, it is directly visible what type of property it is (Boolean, string, etc.).

Shows if a property is currently being used.

Not contained by default in the view, but can be selected using Context menu->Column selection.



Information

Only logical objects can be displayed in the configuration dialog. Visual objects are not displayed. You can read about backgrounds and how visual objects can be animated in the Allocation of zenon object to WPF content.

EDIT HYPERLINKS

All configured hyperlinks can be edited from the properties of the element. Click on the element and open the property group **WPF links**. Hyperlinks can be further configured here, without having to open the dialog.

Limitations:

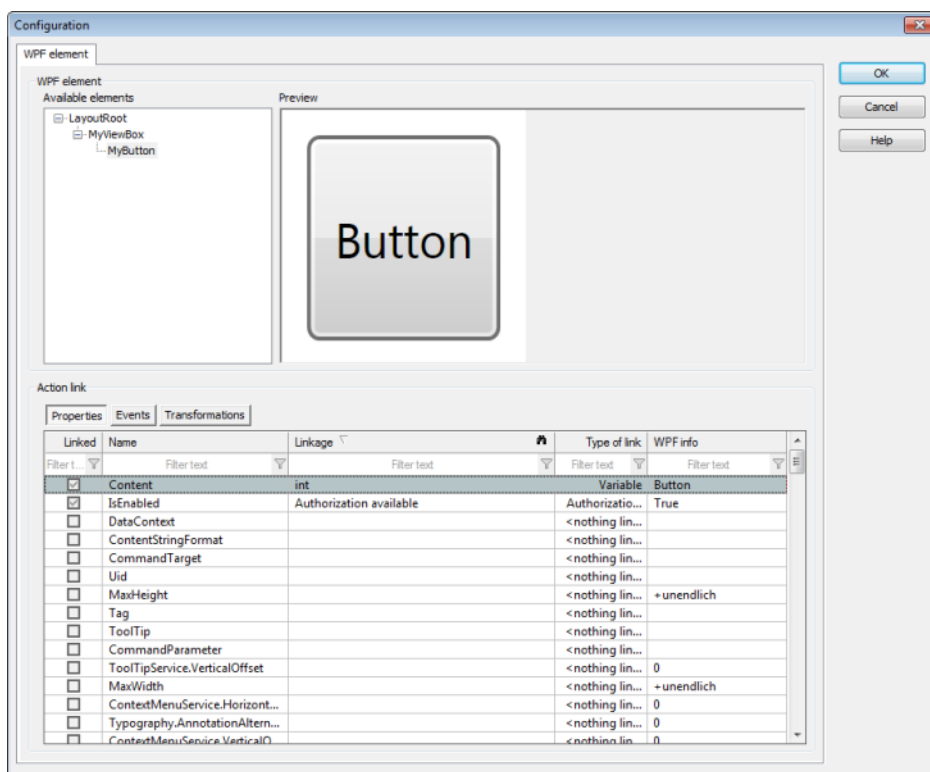
- ▶ The linking type cannot be changed here.
- ▶ New linkings can only be created via the configuration dialog.

- Insertion of a WPF elements into a symbol: WPF linkings cannot be exported.

Properties

The properties enable the linking of:

- Variables (on page 99)
- Values (on page 100)
- Authorizations and interlockings (on page 101)



Parameters	Description
Name	Name of the property.
Connection	Linked variable, authorization or linked value. Clicking in the column opens the respective selection dialog, depending on the entry in the Link type column.
Link type	Selection of linking.
WPF info	Shows the current value for properties in WPF content. For the user, it is directly visible what type of property it is (Boolean, string, etc.).
Linked	Shows if a property is currently being used. Not contained by default in the view, but can be selected using Context menu->Column selection.

CREATE LINK

To create a link:

1. Highlight the line with the property that is to be linked
2. Click in the **Link type cell**
3. Select the desired link from the drop-down list.

The following are available:

- <not linked> (deletes an existing link)
 - Authorization/Interlocking
 - Constant value
 - Variable
4. Click in the **Link** cell
 5. The dialog for configuring the desired link opens



Information

*Properties of WPF and zenon can be different. If, for example the **visibility** property is linked, there are three values available in .NET:*

0 - visible

1 - invisible

2- collapsed

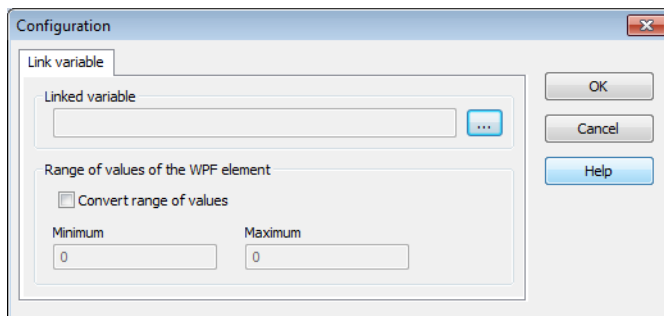
These values must be displayed via the linked zenon variable.

Link variable

To link a variable with a WPF property:

1. Highlight the line with the property that is to be linked
2. Click in the **Link type cell**
3. Select from the **variable** drop down list
4. Click in the **Link** cell
5. The dialog for configuring the variables opens

This dialog also applies for the selection of variables with transformations (on page 104). The configuration also makes it possible to convert from zenon into WPF units.



Parameters	Description
Linked variables	Selection of the variable to be linked. A click on the ... button opens the selection dialog.
Value range of WPF element	Data to convert variable values into WPF values.
Convert value range	<p>Active: WPF unit conversion is switched on.</p> <p>Effect on Runtime: The current zenon value (incl. zenon unit) is converted to the WPF range using standardized minimum and maximum values.</p> <p>For example: The value of a variable varies from 100 to 200. With the variables, the standardized range is set to 100 - 200. The aim is to display this change in value using a WPF rotary knob. For this:</p> <ul style="list-style-type: none"> ▶ for Transformations, the RotateTransform.Angle property is linked to the variables ▶ Adjust value activated ▶ a WPF value range of 0 to 360 is configured <p>Now the rotary knob can be turned at a value of 150, for example, by 180 degrees.</p>
Minimum	Defines the lowest WPF value.
Maximum	Defines the highest WPF value.
OK	Accepts settings and ends the dialog.
Cancel	Discards settings and ends the dialog.
Help	Opens online help.

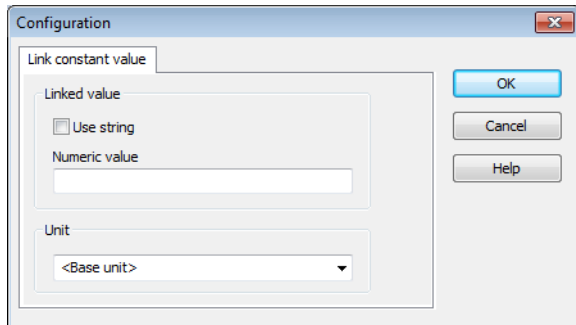
Link values

Linked values can either be a **String** or a numerical value of the type **Double**. When selecting the screen, the selected value is sent in WPF content after loading the WPF content.

To link a value with a WPF property:

1. Highlight the line with the property that is to be linked
2. Click in the **Link type cell**
3. Select **Value linkings** from the drop-down list
4. Click in the **Link** cell

5. The dialog for configuration of value linking opens



Parameters	Description
Linked value:	Entry of a numerical value or string value.
Use string	<p>Active: A string value is used instead of a numerical value.</p> <p>The language of string values can be switched. The text is translated in Runtime when the screen is called up and sent in WPF content. If the language is switched whilst the screen is opened, the string value is retranslated and sent.</p>
String value/numerical value	Depending on what is selected for the Use string property, a numerical value or a string value is entered into this field. For numerical values, a unit of measurement can also be selected.
Unit:	<p>Selection of a unit of measurement from the drop down list. You must have configured this in unit switching beforehand.</p> <p>The unit of measurement is allocated with the numerical value. If the units are switched in Runtime, the value is converted to the new unit of measurement and sent to WPF content.</p>
OK	Accepts settings and ends the dialog.
Cancel	Discards settings and ends the dialog.
Help	Opens online help.

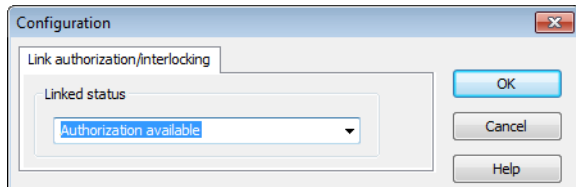
Link authorization or interlocking

Authorizations cannot be granted for the whole WPF element. The element is allocated a user level. Authorizations are granted within the user level for individual controls. If an authorization is active, the value 1 is written to the element.

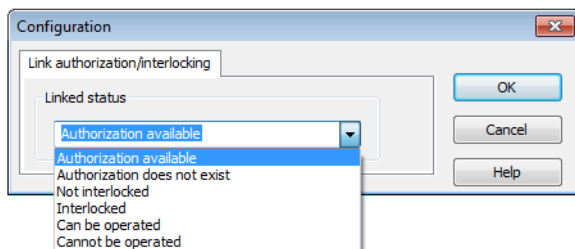
To link an authorization or interlocking with a WPF property:

1. Highlight the line with the property that is to be linked

2. Click in the **Link type cell**
3. Select **Authorization/interlocking** from the drop down menu
4. Click in the **Link** cell
5. The dialog for configuring the authorizations opens



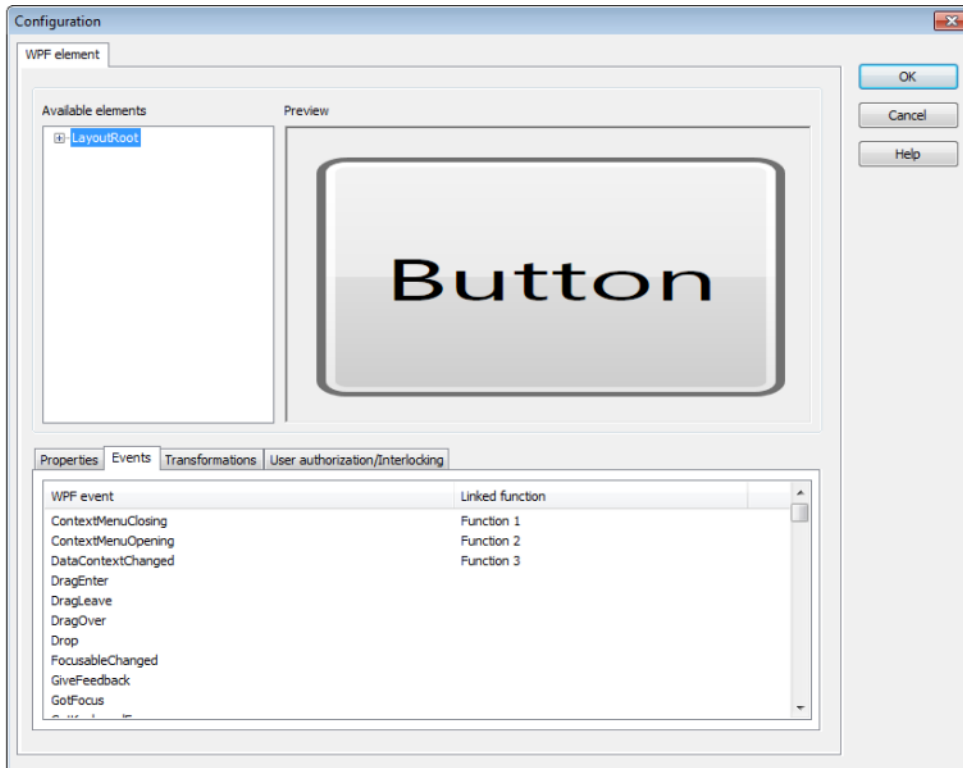
Parameters	Description
Link authorization/interlocking	Setting the authorizations.
Linked status	Selection of an authorization that is linked to a WPF control from the drop down list. For example, visibility and operability of a WPF button can depend on a user's status.



Authorization	Description
Authorization available	If the user has sufficient rights to operate the WPF element , a value of 1 is written to the property.
Authorization does not exist	If the user does not have sufficient rights to operate the WPF element , a value of 1 is written to the property.
Not interlocked	If the element is not locked, the value 1 is written to the property.
Interlocked	If the element is locked, the value 1 is written to the property.
Can be operated	If authorization is present and the element is not locked, then a value of 1 is written to the property.
Cannot be operated	If authorization is not present or the element is not locked, then a value of 1 is written to the property.

Events

Events make it possible to link zenon functions to a WPF element.



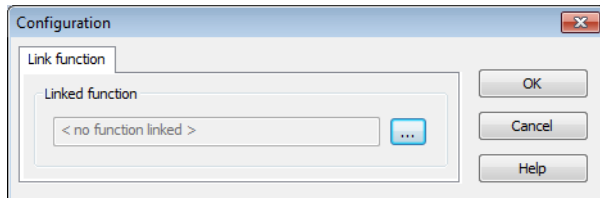
Parameters	Description
Name	Name of the property.
Connection	Linked function. Clicking in the cell opens the configuration dialog.
Link type	Selection of linking. Clicking in the cell opens the selection dialog.
WPF info	Shows the current value for properties in WPF content. For the user, it is directly visible what type of property it is (Boolean, string, etc.).
Linked	Shows if a property is currently being used. Not contained by default in the view, but can be selected using Context menu->Column selection.

LINK FUNCTIONS

To create a link:

1. Highlight the line with the property that is to be linked

2. Click in the **Link type cell**
3. Select from the drop down list function
4. Click in the **Link** cell
5. The dialog for configuring the function opens

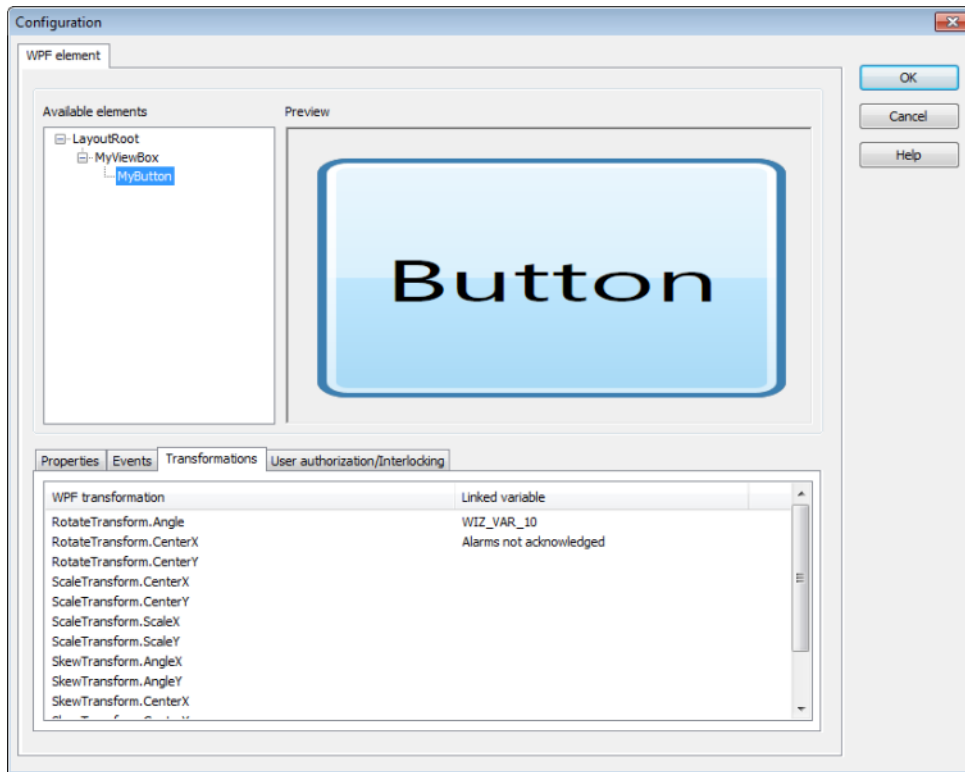


Parameters	Description
Linked function	Selection of the function to be linked. Clicking on the ... button opens the dialog for Function selection.
OK	Accepts selection and closes dialog.
Cancel	Discards changes and closes dialog.
Help	Opens online help.

Transformation

The **WPF element** does not support rotation. If, for example, the **WPF element** is in a symbol and the symbol is rotated, the **WPF element** does not rotate with it. Therefore there is a different mechanism for **Transformation** with WPF to turn elements or to otherwise transform them. These transformations are configured in the **Transformation** tab.

Attention: If the content is outside of the **WPF element** area, this part of the contents is lost or it is not shown.



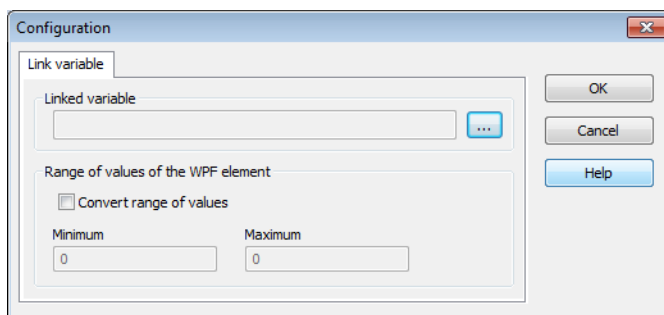
Parameters	Description
Name	Name of the property.
Connection	<p>Selection of the linked variables.</p> <p>Transformations are displayed in XAML as transformation objects with their own properties. If an element supports a transformation, then the possible properties of the transformation object are displayed in list view. (more on this in: Integrate button as WPF XAML in zenon (on page 145))</p> <p>For example, if the linked variable is set at the value of 10, then this value is written as a WPF target and the WPF element is rotated by 10°.</p>
Link type	Selection of transformation link type.
WPF info	Shows the current value for properties in WPF content. For the user, it is directly visible what type of property it is (Boolean, string, etc.).
Linked	<p>Shows if a property is currently being used.</p> <p>Not contained by default in the view, but can be selected using Context menu->Column selection.</p>

LINK TRANSFORMATIONS

To link a transformation with a WPF property:

1. Highlight the line with the property that is to be linked
2. Click in the **Link type cell**
3. Select from the **Transformation** drop down list
4. Click in the **Link** cell
5. The dialog for configuring the variables opens

The configuration also makes it possible to convert from zenon into WPF units.



Parameters	Description
Linked variables	Selection of the variable to be linked. A click on the ... button opens the selection dialog.
Value range of WPF element	Data to convert variable values into WPF values.
Convert value range	<p>Active: WPF unit conversion is switched on.</p> <p>Effect on Runtime: The current zenon value (incl. zenon unit) is converted to the WPF range using standardized minimum and maximum values.</p> <p>For example: The value of a variable varies from 100 to 200. With the variables, the standardized range is set to 100 - 200. The aim is to display this change in value using a WPF rotary knob. For this:</p> <ul style="list-style-type: none"> ▶ for Transformations, the RotateTransform.Angle property is linked to the variables ▶ Adjust value activated ▶ a WPF value range of 0 to 360 is configured <p>Now the rotary knob can be turned at a value of 150, for example, by 180 degrees.</p>
Minimum	Defines the lowest WPF value.
Maximum	Defines the highest WPF value.
OK	Accepts settings and ends the dialog.
Cancel	Discards settings and ends the dialog.
Help	Opens online help.

6.3.4 Validity of XAML Files

XAML files are valid subject to certain requirements:

- ▶ Correct name spaces
- ▶ No class references
- ▶ Scalability

CORRECT NAME SPACE

The **WPF element** can only display WPF content, i.e.:

Only XAML files with the correct WPF namespace can be displayed by the **WPF element**. Files that use a Silverlight namespace cannot be loaded or displayed. However, in most cases it is suffice to change the Silverlight namespace to the WPF namespace.

WPF-Namespaces:

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

NO USE OF CLASS REFERENCES

Because the XAML files can be loaded dynamically, it is not possible to use XAML files that contain references to classes ("class" key in header). Functions that have been programmed in independently-created C#- files cannot be used.

In order to use WPF user controls with code behind, the process as described in the Creating a simple WPF user control with code behind function must be carried out.

SCALABILITY

If the content of a **WPF element** is adjusted to the size of the **WPF element**, then the controls of the **WPF element** are interlaced in a control that offers this functionality, such as a **view box** for example. In addition, it must be ensured that the **height** and **width** for this elements are configured as **automatic**.

CHECKING AN XAML FILE TO SEE IF IT IS CORRECT

To check if an XAML file has the correct format:

- ▶ Open XAML file in Internet Explorer
 - If it can be opened without additional plug-ins (Java or similar), then it can be assumed with a high degree of certainty that this file can be loaded or displayed by zenon
 - if problems occur during loading, these are then shown in Internet Explorer and the lines in which problems arise can be clearly seen

The scaling can also be tested in this manner: If the file has been created correctly, the content will adjust to the size of the Internet Explorer window.

ERROR MESSAGE

If an invalid file is used in zenon, then an error message is displayed in the output window when loading the file in the WPF element.

For example:

"error when loading

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

The attribute "Class" cannot be found in XML namespace

"http://schemas.microsoft.com/winfx/2006/xaml". Line 7 Position 2."

6.3.5 Pre-built elements

zenon is already shipped with several WPF elements. More are available for download in the web shop.

All WPF elements have properties which determine the graphical design of the respective element (Dependency Properties). Setting the values via an XAML file or linking the property via zenon can directly change the look in the Runtime. The tables in the description of the individual elements contain the respective Dependency Properties, depending on the control.

Available elements:

- ▶ Analog clock (on page 110)
- ▶ Vertical bar graph (on page 110)
- ▶ Comtrade Viewer (on page 111)
- ▶ Energy class diagram (on page 122)
- ▶ Progress bar (on page 111)
- ▶ Pareto diagram (on page 123)
- ▶ Sankey diagram (on page 130)
- ▶ Round display (on page 127)
- ▶ Temperature control (on page 132)
- ▶ Universal slider (on page 133)
- ▶ Waterfall diagram (on page 134)

REPLACING ASSEMBLY WITH A NEWER VERSION

Per project only one assembly for a WPF element can be used in the zenon Editor as well as in the Runtime. If two versions of an assembly are available in a project, then the first loaded file is used. A user enquiry is made as to which version should be used. No further actions are needed for the maintenance of the versions used up until now. If a newer version is chosen, all corresponding CDWPF files in all symbols and images in all projects must be adapted.

Note for Multi-Project Administration: If an assembly in a project is replaced by a new version, it must also be replaced in all other projects that are loaded in the Editor or in Runtime.

Analog clock - AnalogClockControl

Property	Function	Value
ElementStyle	Shape/type of element.	Enum: <ul style="list-style-type: none"> ▸ SmallNumbers ▸ BigNumbers ▸ No
ElementBackgroundBrush	Color of element background.	Brush
ElementGlasReflection	Activate the glass effect on the element.	Visibility
Offset	Value in hours (h) which displays the time lag to the system clock.	Int16
OriginText	Text which is displayed in the clock (e.g. location).	String

Bar graph vertical - VerticalBargraphControl

Property	Function	Value
CurrentValue	Current value which should be displayed.	Double
MinValue	Minimum value of the scale.	Double
MaxValue	Maximum value of the scale.	Double
MajorTicksCount	Number of main ticks on the scale.	Integer
MinorTicksCount	Number of sub ticks on the scale.	Integer
MajorTickColor	Color of main ticks on the scale.	Color
MinorTickColor	Color of sub ticks on the scale.	Color
ElementBorderBrush	Color of the element border.	Brush
ElementBackgroundBrush	Color of element background.	Brush
ElementGlasReflection	Activate the glass effect on the element.	Visibility
ElementFontFamily	Element font.	Font
ScaleFontSize	Font size of the scale.	Double
ScaleFontColor	Font color of the scale.	Color
IndicatorBrush	Bar graph fill color.	Brush
BargraphSeparation	Number of bar graph division.	Integer
BargraphSeparationColor	Color of the scale division.	Color

Progress bar - ProgressBarControl

Property	Function	Value
CurrentValue	Current value which should be displayed.	Double
MinValue	Minimum value of the value area.	Double
MaxValue	Maximum value of the value area.	Double
ProgressbarDivisionCount	Number of divisions of the progress bar.	Integer
VisibilityText	Visibility of the value display.	Boolean
TextSize	Font size of the value display.	Double
TextColor	Color of the value display.	Color
ProgressBarBoxedColor	Color of the border of the progress bar.	Color
ProgressBarMarginDistance	Distance of the progress bar box from the element edge (left, top, right, down).	Double
ProgressBarInactiveBrush	Indicator color not active.	Brush
ProgressBarActiveBrush	Indicator color active.	Brush
ProgressBarPadding	Distance of the progress bar from the progress bar box (left, top, right, down).	Double
ElementBorderBrush	Color of the element border.	Brush
ElementBackgroundBrush	Color of element background.	Brush

COMTRADE-Viewer

The **COMTRADE-Viewer** WPF element is available to partners of COPA-DATA and is available to these via the Partner Portal.

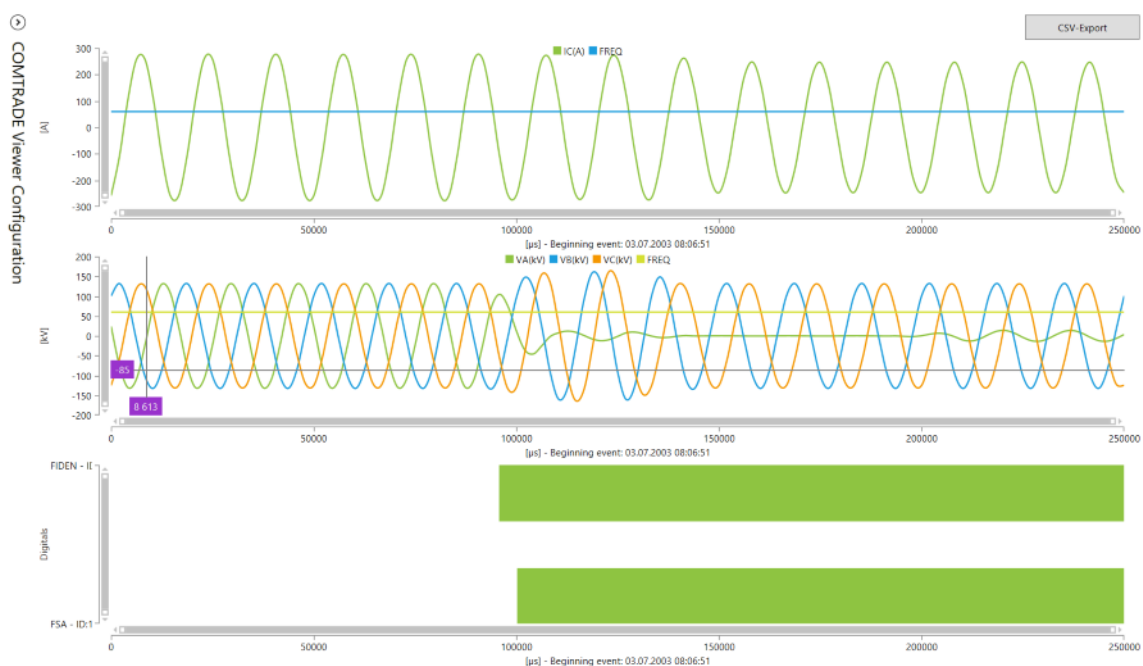
It is for the graphical analysis of digital error and result logging of a COMTRADE file.



Information

The control supports IEEE C37.111 (IEEE Standard Common Format for Transient Data Exchange (COMTRADE) for Power Systems) standards-compliant files. ASCII or binary files in accordance with the 1999 or 2013 edition can be visualized.

Older files or files without a year identification are not supported. This is displayed with a warning dialog in Runtime.



Possibilities of the **COMTRADE-Viewer** WPF control in zenon Runtime:

- ▶ Selection of a file in the COMTRADE file format
- ▶ Exports selected objects as an CSV file.
- ▶ Visualization of the selected COMTRADE file:

Note: The display colors can be configured in the zenon Editor.

 - Current (sinus wave display)
 - Voltage (sinus wave display)
 - Digital signals (binary bar chart display)
 - Display of values at a selected cursor position.
 - If an element that represents neither current or voltage is selected, (such as frequency), this is visualized in both analog areas again (current and voltage).
- ▶ Navigation:

- Zoom in and zoom out using the mouse wheel, scroll bar and Multi-Touch gestures
- Enlargement of the area
Selection of the area by clicking the mouse
- Move the display area using the right mouse button, scroll bar or Multi-Touch gestures.



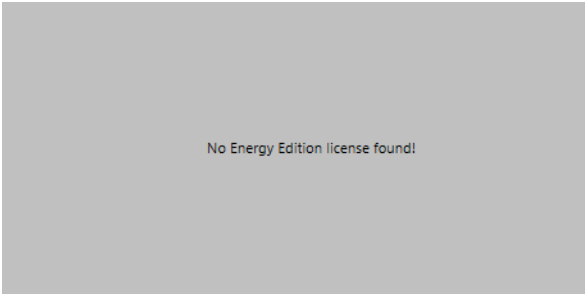
Hint

To be able to transport COMTRADE files to the zenon Runtime computer, you can also use the file transfer of the **IEC 61850 driver** or the **FTP function block** of zenon Logic.

You can find further information about this in the driver documentation of the IEC 61850 driver or in the zenon Logic documentation.

LICENSING

The **COMTRADE-Viewer** can only be configured in the zenon Editor with a valid Energy Edition license. If there is no valid license, the WPF is displayed as grayed out in the Editor. A valid Energy Edition license is also required for display in Runtime.



No Energy Edition license found!

Display during Runtime

The **COMTRADE** WPF element offers two views in Runtime:

- ▶ Configuration view
 - Selection of a COMTRADE configuration file
 - Selection of the elements to be displayed
- ▶ Graph view
 - Zoom in and zoom out
 - Display of values at a selected cursor position.
 - Export of the selected elements as an CSV file



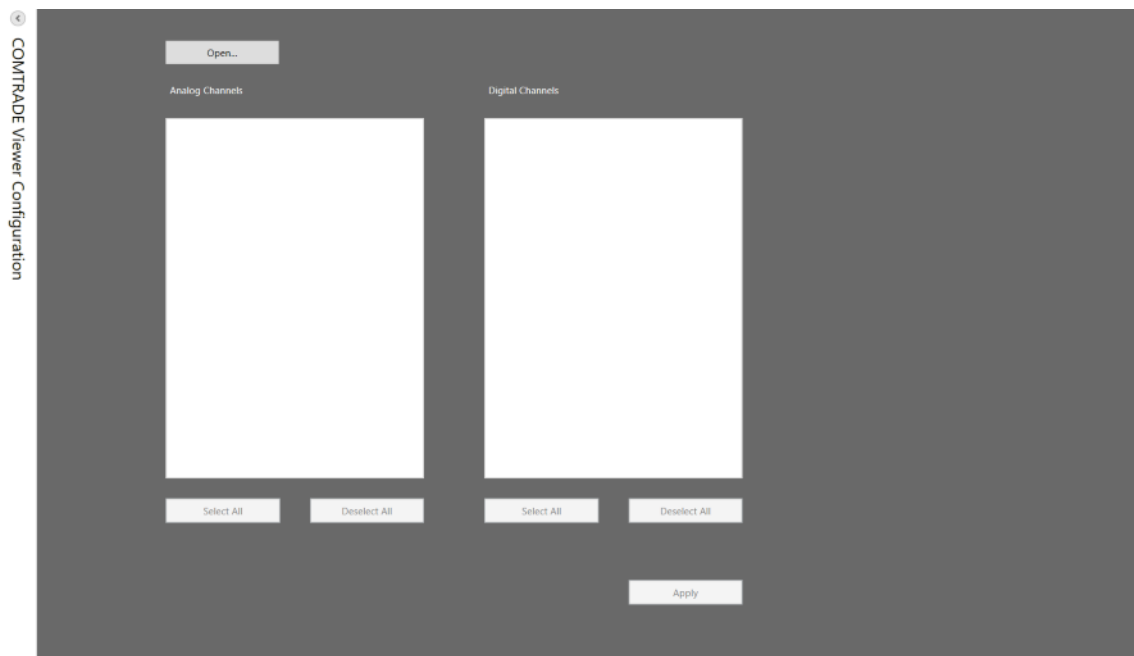
Information

The switch between the views is integrated in the WPF element. Additional project configuration of a screen switching function is not necessary.

Runtime view - configuration page

If a screen with a configured **COMTRADE-Viewer** WPF element is called up, the display of the respective configuration page is empty.

Note: This also applies if, in zenon Runtime, there is a switch from one screen to another screen with the screen switching function.



COMTRADE VIEWER CONFIGURATION

The **COMTRADE Viewer Configuration** switching, arranged vertically on the side, switches the display of the configuration to graphic view and vice versa.

SELECT FILE

The **Open...** button opens the file selection dialog to select a file.

There is a pre-selection for display in the file selection:

- ▶ In doing so, file pairs of *.cfg- and *.dat files are detected.
Note: Optional *.hdr or *.inf files are not taken into account.
- ▶ Only the corresponding *.dat files are displayed.
- ▶ All attendant files (*.dat, *.cfg) are loaded by clicking on the desired file and the **OK** button.
- ▶ One file can be loaded.
- ▶ After loading the file, the content of the file is shown in the **Analog Channels** and **Digital Channels** columns.
The labels and units of the elements originate from the COMTRADE configuration and cannot be changed.

ANALOG CHANNELS

Parameters	Description
[Liste der verfügbaren Kanäle]	Selection of the elements to be visualized. Multiple selection by clicking on the desired entry in the list. Selected elements are shown with a colored background. Another mouse click undoes the selection of the entry.
Select All	Selects all elements from the list.
Deselect All	Deactivates the existing selection of elements.

DIGITAL CHANNELS

Parameters	Description
[Liste der verfügbaren Kanäle]	Selection of the elements to be visualized Multiple selection by clicking on the desired entry in the list. Selected elements are shown with a colored background. Another mouse click undoes the selection of the entry.
Select All	Selects all elements from the list.
Deselect All	Deactivates the existing selection of elements.

SHOW SELECTION

To show your selection in the graphic view, click on the **Apply** button.

Note: Clicking on the vertically-arranged **COMTRADE Viewer Configuration** switching only changes the view. An amended selection of the channels is not taken into account in the process.

Runtime view - visualization of COMTRADE data

The selected channels are visualized in the graph view of the **COMTRADE-Viewer** WPF element. The coloring can be configured in the zenon Editor.

EXPORT OF THE SELECTED DATA

The selected analog and digital channels can be exported to a CSV file with the **CSV-Export** button.

GRAPH VIEW

The graph view of the **COMTRADE-Viewers** is divided into three sections:

- ▶ Current amperage
Upper area
- ▶ Voltage
Mid area
- ▶ Digital channels
Lower area



AXIS LABELING

- ▶ Horizontal axis
The horizontal axis represents the complete time period as illustrated in the COMTRADE file (*.dat).

The scaling of this time axis depends on the enlargement level. The higher the enlargement selected, the more detailed the time display.

► Vertical axis

The vertical axis represents the values.

- The scaling of the value axis depends on the enlargement level. The greater the enlargement selected, the more detailed the display of values.
- The labeling of the analog channels is shown vertically next to the values and corresponds to the measuring unit as defined in the COMTRADE file (*.cfg).
- The digital channels are displayed in the sequence as defined in the COMTRADE file (*.cfg).

The Channel identifier of the COMTRADE file serves as an identifier.

KEY

■ IA(A) ■ IB(A) ■ IC(A) ■ IG(A)

The color key of the graphs is shown at the head of the graph.

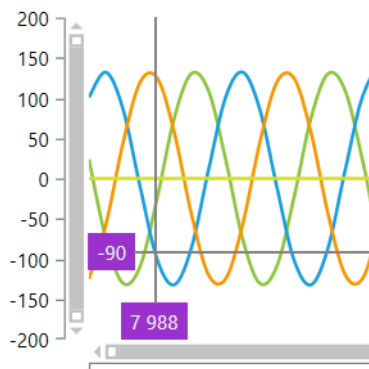
- The labeling of the digital channels corresponds to the channel description as defined in the COMTRADE file (*.cfg).
- The colors for each channel are assigned automatically with the configured color palette.
- The time is displayed in a footer under the graph. The start time is displayed as a text.

NAVIGATION AND ZOOM

Navigation (scroll and zoom) is always applied to all three areas of the graphic display.

- You can move the display within the horizontal time line with the scroll bar.
- Zoom in and zoom out
 - You can zoom at the current position of the mouse pointer in the graphics view or reduce the enlargement.
 - The selected area is displayed by selecting a display area with the mouse button held down. **Note:** The display of the values is always amended to the selected area. As a result, this can lead to a flattening of the curve in the enlarged graphic view.
 - Double clicking on the scroll bar resets the enlargement.

ANALYSIS



The precise values at the position of the mouse pointer are visualized with a display in value blocks. A crosshair offers additional visual support with the exact determination of the reading position.

Configurable control properties - color display

ENGINEERING IN THE EDITOR

The element with the name **COMTRADE.CDWPF** can be configured and placed in each zenon screen type.

The project configuration of **Width [pixels]** and **Height [pixels]** of the element depend on the proportions. This prevents the **COMTRADE-Viewer** being displayed as distorted in Runtime.

Note: When configuring the project, ensure that there is sufficient size to guarantee a clear overview.

GRAPHICAL AMENDMENTS

You configure the graphic design in the properties of the WPF element.

You can find further information in the configuration of the linking (on page 95) chapter in this manual.

Possible color values:

- ▶ Hexadecimal color values
#RRGGBB
Example color values: #000000 = black , #FFFFFF = white, #FF0000 = red
- ▶ Color values by name
Reference: <https://msdn.microsoft.com/en-us/library/system.drawing.color.aspx>
(<https://msdn.microsoft.com/en-us/library/system.drawing.color.aspx>)

**Hint**

The properties for the **COMTRADE-Viewer** WPF element have a "z" as a starting color. Use name filtering for a clear display when configuring the linking.

CONFIGURATION PAGE

Text and background color of the configuration page.

Analog Channels

Parameters	Description	Value
zConfiguratinPageTextColor	Text color of the configuration page	String
zConfigurationPageBackgroundColor	Background color of the configuration page	String

BUTTONS

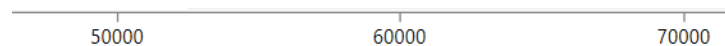
Text and background color of the button.

Open...

Parameters	Description	Value
zButtonTextColor	Text color of the button	String
zButtonBackgroundColor	Background color of the button	String

CHART

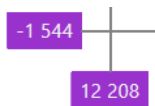
Text color of the axis labeling or key and background color.



Parameters	Description	Value
zChartTextColor	Text color of the axis labeling.	String
zChartBackgroundColor	Background color of the axis labeling	String

LABEL

Text and background color of the display of a selected cursor position.

















Parameters	Description	Value
zChartLabelTextColor	Text color of the value display	String
zChartLabelBackgroundColor	Background color of the value display	String

CHART

Color palette of the graph view and the attendant keys.

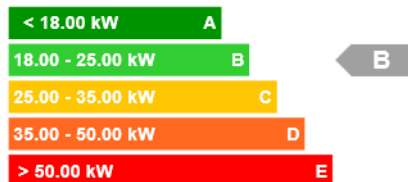
Parameters	Description	Value
zChartPalette	<p>Color palette of the colors for graphs and keys.</p> <p>Referencing with color palette name (see overview).</p> <p>Default: if no color palette is configured, the color palette of the computer's operating system is used.</p>	String

POSSIBLE COLOR PALETTES - OVERVIEW

Arctic	
Autumn	
Cold	
Flower	
Forest	
Grayscale	
Ground	
Lilac	
Natural	
Pastel	
Rainbow	
Spring	
Summer	
Warm	
Windows8	

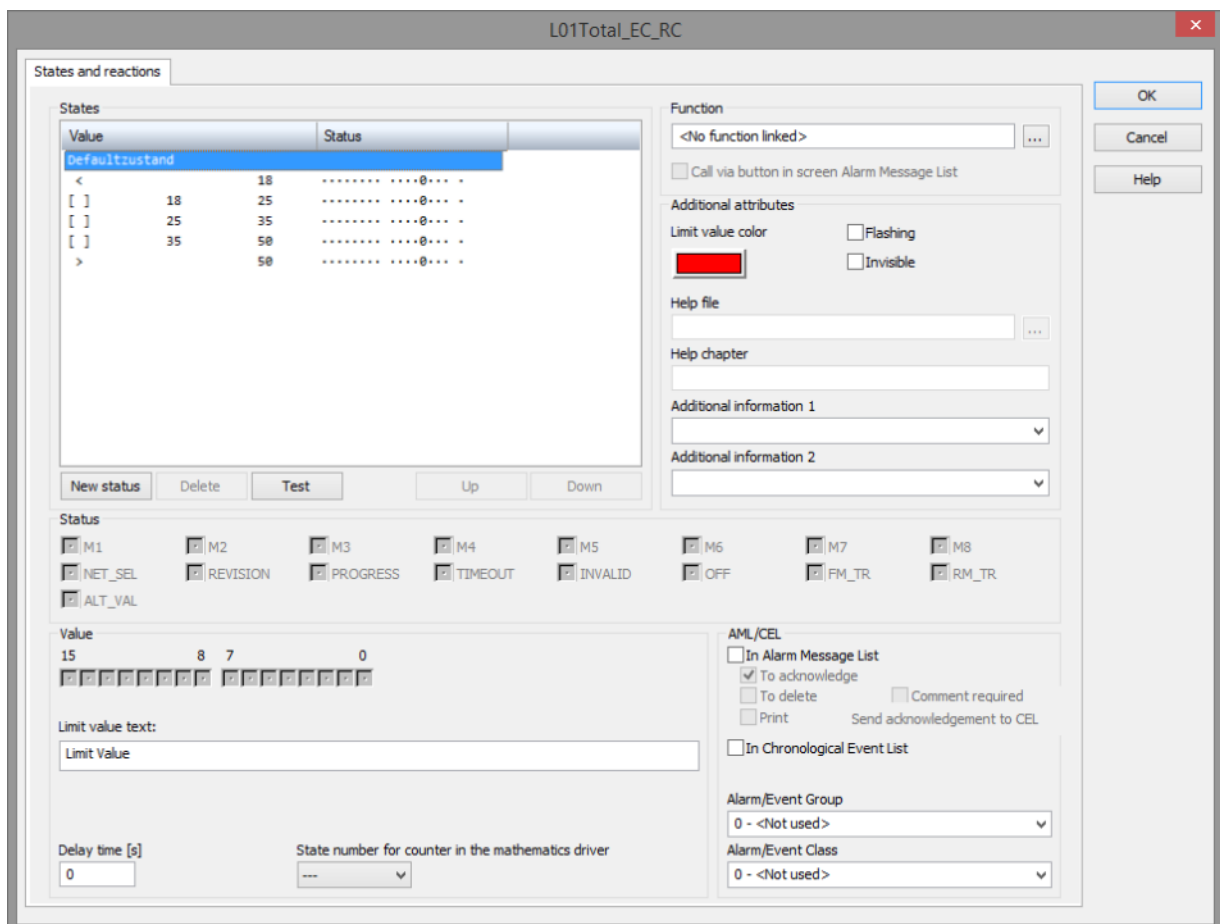
Energy class diagram

The energy class diagram, WPF element is available to partners of COPA-DATA and is available to these via the Partner Portal.



A reaction matrix must be used to model an energy class diagram. This reaction matrix must be linked to the variable whose value is envisaged for display and distribution in energy classes. The name of the variable must be transferred to the "zVariableName" property.

REACTION MATRIX FOR ENERGY CLASS DIAGRAM



The linked reaction matrix must correspond to the following schematic:

- The first status must be an area, or a "less than" definition

- ▶ As many different areas as desired can then be defined.
- ▶ The last status must be an area or a "greater than" definition.

The following is applicable for project configuration:

1. If the first status is an area and the value of the variable comes under this area, the first status in the diagram is shown nevertheless. The same is applicable for the last status the other way round.
2. The colors that the WPF diagram uses for the classes are the limit value colors that were defined in the reaction matrix.
3. The letters for the classes are set in alphabetical order starting with "A".

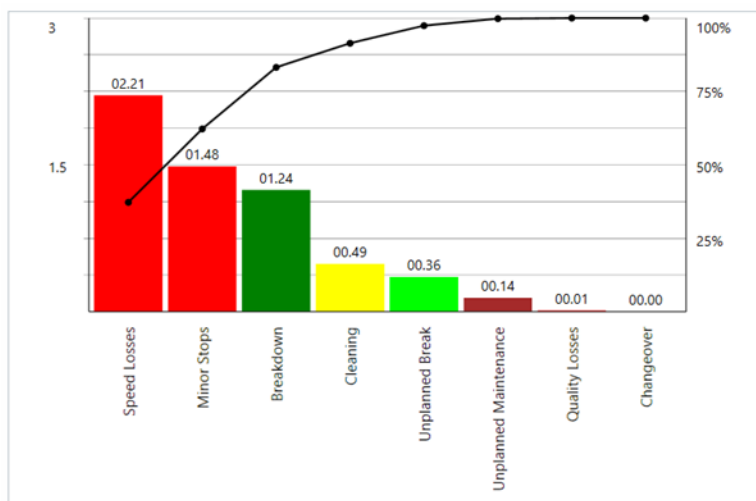
Property	Description	Value
zenonFontID	ID for a font from the first font list (font size is not taken into account)	Integer
zenonNumberOfDecimalPlaces	Number of displayed decimal points	Integer
zenonVariableName	Name of the variable to be displayed.	String

Note: Additional VSTA programming is necessary for the display of the energy class diagram in the zenon web client. You can find details on this in the display of WPF elements in the zenon web client (on page 135).

Pareto diagram

The Pareto diagram, WPF element is available to partners of COPA-DATA and is available to these via the Partner Portal.

An example of a Pareto diagram in Runtime is shown below:



The following settings can be made in the WPF configuration window under **COPADATA-ELEMENT**:

Property	Function	Value
zenonBarColor1	Color of the first Bar	Color (String)
zenonBarColor2	Color of the second Bar	Color (String)
zenonBarColor3	Color of the third Bar	Color (String)
zenonBarColor4	Color of the fourth Bar	Color (String)
zenonBarColor5	Color of element fifth Bar	Color (String)
zenonBarColor6	Color of element sixth Bar	Color (String)
zenonBarColor7	Color of element seventh Bar	Color (String)
zenonBarColor8	Color of element eighth Bar	Color (String)
zenonBarColor9	Color of element ninth Bar	Color (String)
zenonBarColor10	Color of element tenth Bar	Color (String)
zenonColorPercentageLine	Color of the percentage line (relative sum frequency).	Color (String)
zenonLineVisibility	Visibility of the percentage line (relative sum frequency).	Boolean
zenonVariable1_Label	Labeling for the 1st Bar	String
zenonVariable1_Value	Value of the 1st Bar	Double
zenonVariable2_Label	Labeling for the 2nd Bar	String
zenonVariable2_Value	Value of the 2nd Bar	Double
zenonVariable3_Label	Labeling for the 3rd Bar	String
zenonVariable3_Value	Value of the 3rd Bar	Double
zenonVariable4_Label	Labeling for the 4th Bar	String
zenonVariable4_Value	Value of the 4th Bar	Double
zenonVariable5_Label	Labeling for the 5th Bar	String
zenonVariable5_Value	Value of the 5th Bar	Double
zenonVariable6_Label	Labeling for the 6th Bar	String
zenonVariable6_Value	Value of the 6th Bar	Double
zenonVariable7_Label	Labeling for the 7th Bar	String

zenonVariable7_Value	Value of the 7th Bar	Double
zenonVariable8_Label	Labeling for the 8th Bar	String
zenonVariable8_Value	Value of the 8th Bar	Double
zenonVariable9_Label	Labeling for the 9th Bar	String
zenonVariable9_Value	Value of the 9th Bar	Double
zenonVariable10_Label	Labeling for the 10th Bar	String
zenonVariable10_Value	Value of the 10th Bar	Double

The following events can be used and linked to zenon functions:

Event	Function	Value
zenonBar1Click	Function that is executed when the 1st bar is clicked on.	Function
zenonBar2Click	Function that is executed when the 2nd bar is clicked on.	Function
zenonBar3Click	Function that is executed when the 3rd bar is clicked on.	Function
zenonBar4Click	Function that is executed when the 4th bar is clicked on.	Function
zenonBar5Click	Function that is executed when the 5th bar is clicked on.	Function
zenonBar6Click	Function that is executed when the 6th bar is clicked on.	Function
zenonBar7Click	Function that is executed when the 7th bar is clicked on.	Function
zenonBar8Click	Function that is executed when the 8th bar is clicked on.	Function
zenonBar9Click	Function that is executed when the 9th bar is clicked on.	Function
zenonBar10Click	Function that is executed when the 10th bar is clicked on.	Function

Circular gauge control

Property	Function	Value
CurrentValue	Current value which should be displayed.	Double
IsReversed	Scale orientation - clockwise or anti-clockwise.	Boolean
ElementFontFamily	Element font.	Font
MinValue	Minimum value of the scale.	Double
MaxValue	Maximum value of the scale.	Double
ScaleRadius	Radius of the scale.	Double
ScaleStartAngle	Angle at which the scale starts.	Double
ScaleLabelRotationMode	Alignment of the scale caption.	Enum: <ul style="list-style-type: none"> ▸ None ▸ Automatic ▸ SurroundIn ▸ SurroundOut
ScaleSweepAngle	Angel area which defines the size of the scale.	Double
ScaleLabelFontSize	Font size of the scale caption.	Double
ScaleLabelColor	Font color of the scale caption.	Color
ScaleLabelRadius	Radius on which the scale caption is orientated.	Double
ScaleValuePrecision	Accuracy of the scale caption.	Integer
PointerStyle	Shape of the pointer displaying the value.	Enum: <ul style="list-style-type: none"> ▸ Arrow ▸ Rectangle ▸ TriangleCap ▸ Pentagon ▸ Triangle
MajorTickColor	Color of main ticks on the scale.	Color
MinorTickColor	Color of sub ticks on the scale.	Color
MajorTickSize	Size of main ticks on the scale.	Size
MinorTickSize	Size of sub ticks on the scale.	Size
MajorTicksCount	Number of main ticks on the scale.	Integer
MajorTicksShape	Shape/type of main ticks on the scale.	Enum: <ul style="list-style-type: none"> ▸ Rectangle

		<ul style="list-style-type: none">▶ Trapezoid▶ Triangle
--	--	--

MinorTicksShape	Shape/type of sub ticks on the scale.	Enum: <ul style="list-style-type: none"> ▸ Rectangle ▸ Trapezoid ▸ Triangle
MinorTicksCount	Number of sub ticks on the scale.	Integer
PointerSize	Size of the pointer.	Size
PointerCapRadius	Size of the pointer fastening point.	Double
PointerBorderBrush	Color of pointer border.	Brush
PointerCapStyle	Shape/type of pointer fastening point.	Enum: <ul style="list-style-type: none"> ▸ BackCap ▸ FrontCap ▸ Screw
PointerCapBorderBrush	Color of pointer fastening point.	Brush
PointerBrush	Color of pointer.	Brush
GaugeBorderBrush	Color of the element border.	Brush
GaugeBackgroundBrush	Color of element background.	Brush
PointerCapColorBrush	Color of pointer fastening point.	Brush
GaugeMiddlePlate	Radius of the element background middle plate.	Double
PointerOffset	Offset of the pointer (displacement).	Double
RangeRadius	Radius of the total range display.	Double
RangeThickness	Thickness of the total range display.	Double
RangeStartValue	Start value of the total range display.	Double
Range1EndValue	End value of the 1st area and start value of the 2nd range.	Double
Range2EndValue	End value of the 2nd area and start value of the 3rd range.	Double
Range3EndValue	End value of the 3rd area and start value of the 4th range.	Double
Range4EndValue	End value of the 4th area and start value of the 5th range.	Double
Range5EndValue	End value of the 5th area and start value of the 6th range.	Double
Range6EndValue	End value of the 6th range.	Double
Range1ColorBrush	Color of the first range.	Brush
Range2ColorBrush	Color of the second range.	Brush
Range3ColorBrush	Color of the third range.	Brush
Range4ColorBrush	Color of the fourth range.	Brush
Range5ColorBrush	Color of element fifth range.	Brush
Range6ColorBrush	Color of element sixth range.	Brush

ScaleOuterBorderBrush	Color of the scale border.	Brush
ScaleBackgroundBrush	Color of scale background.	Brush
ValueTextFrameStyle	Shape/type of value display.	Enum: <ul style="list-style-type: none"> ▸ LargeFrame ▸ SmallFrame ▸ None
ValueTextContent	Content of the value display.	Enum: <ul style="list-style-type: none"> ▸ Text ▸ TextValue ▸ Value
ValueTextSize	Font size of the value display.	Double
ValueTextColor	Font size of the value display.	Color
IsGlasReflection	Activate the glass effect on the element.	Boolean
GaugeOffsett	Lowering the rotation point of the whole element.	Double

Sankey diagram

The Sankey diagram, WPF element is available to partners of COPA-DATA and is available to these via the Partner Portal.

The Sankey wizard must be used to model a Sankey diagram. The wizard creates an XML file that is then evaluated by the WPF element. To do this, the **zSankeyName** property must be given the name of the XML file. The XML file must be in the `Other` folder of a project. This is saved there by the wizard.

An example of a Sankey diagram in Runtime is shown below:

The following settings can be made in the WPF configuration window under **COPADATA-ELEMENT**:

Property	Function	Value
FontSize	Font size of the texts.	Integer
zBackgroundColor	Background color of the diagram.	Color (String)
zFontColor	Color of the texts.	Color (String)
zFontFamily	Font of all texts.	Font (String)
zLossDetectionActive	Automatic loss detection activated/deactivated. If <code>true</code> , then losses are automatically shown at a node points as flows.	Boolean
zNoDataText	Text that is displayed if there are no values to display and zPrevireActive is <code>false</code> .	String
zNoValidXMLText	Text that is displayed if no valid XML file with entered name has been found and zPreviewActive is <code>false</code> .	String
zNumberOfDecimalPlaces	Denotes how many decimal places are to be displayed.	Integer
zPreviewActive	Display of a preview activated/deactivated. The preview can be displayed if There is no data present (the modeled diagram is filled with default values) or the XML file was not found or this does not contain a valid definition (an example Sankey diagram is displayed).	Boolean
zRefreshRate	Rate at which the diagram is refreshed in ms.	Integer
zSankeyName	Name of the XML file with the modeling of the diagram.	String
zShowRelativeValues	Display of the values in absolute <code>false</code> or relative values <code>true</code> .	Boolean

Note: Additional VSTA programming is necessary for the display of the Sankey class diagram in the zenon web client. You can find details on this in the display of WPF elements in the zenon web client (on page 135).

Temperature indicator - TemperatureIndicatorControl

Property	Function	Value
CurrentValue	Current value which should be displayed.	Double
MinValue	Minimum value of the scale.	Double
MaxValue	Maximum value of the scale.	Double
MajorTicksCount	Number of main ticks on the scale.	Integer
MinorTicksCount	Number of sub ticks on the scale.	Integer
TickNegativColor	Color of the negative main tick (gradient to TickPositivColor).	Color
TickPositivColor	Color of the positive main tick (gradient to TickNegativColor).	Color
MinorTickColor	Color of the sub ticks.	Color
ElementBorderBrush	Color of the element border.	Brush
ElementBackgroundBrush	Color of element background.	Brush
ElementGlasReflection	Activate the glass effect on the element.	Visibility
ElementFontFamily	Element font.	Font
IndicatorColor	Color of the indicator fill color.	Color
IndicatorBorderColor	Color of the indicator border.	Color
MajorTickSize	Size of main ticks on the scale.	Size
MinorTickSize	Size of sub ticks on the scale.	Size
ScaleLetteringDistance	Distance of the scale caption (vertical), each x. main tick should be captioned.	Integer
IndicatorScaleDistance	Distance between indicator and scale (horizontal).	Double
ScaleFontSize	Font size of the scale.	Double
ScaleFontColor	Font color of the scale.	Color
Unit	Unit.	String
ElementStyle	Shape/type of element.	Enum: ▶ SmallFrame ▶ Unit ▶ None

Universal slider - UniversalReglerControl

Property	Function	Value
CurrentValue	Current value which should be displayed.	Double
ElementFontFamily	Element font.	Font
MinValue	Minimum value of the scale.	Double
MaxValue	Maximum value of the scale.	Double
Radius		Double
ScaleRadius	Radius of the scale.	Double
ScaleStartAngle	Angle at which the scale starts.	Double
ScaleLabelRotationMode	Alignment of the scale caption.	Enum: <ul style="list-style-type: none"> ▸ None ▸ Automatic ▸ SurroundIn ▸ SurroundOut
ScaleSweepAngle	Angel area which defines the size of the scale.	Double
ScaleLabelFontSize	Font size of the scale caption.	Double
ScaleLabelColor	Font color of the scale caption.	Color
ScaleLabelRadius	Radius on which the scale caption is orientated.	Double
ScaleValuePrecision	Accuracy of the scale caption.	Integer
ElementStyle	Display type of the element	Enum: <ul style="list-style-type: none"> ▸ Knob ▸ Plate ▸ None
MajorTickColor	Color of main ticks on the scale.	Color
MinorTickColor	Color of sub ticks on the scale.	Color
MajorTickSize	Size of main ticks on the scale.	Size
MinorTickSize	Size of sub ticks on the scale.	Size
MajorTicksCount	Number of main ticks on the scale.	Integer
MajorTicksShape	Shape/type of main ticks on the scale.	Enum: <ul style="list-style-type: none"> ▸ Rectangle ▸ Trapezoid ▸ Triangle

MinorTicksShape	Shape/type of sub ticks on the scale.	Enum: <ul style="list-style-type: none"> ▸ Rectangle ▸ Trapezoid ▸ Triangle
MinorTicksCount	Number of sub ticks on the scale.	Integer
BackgroundBorderBrush	Color of the element border.	Brush
BackgroundBrush	Color of element background.	Brush
PointerCapColorBrush	Color of pointer fastening point.	Brush
GaugeMiddlePlate	Radius of the element background middle plate.	Double
ValueFontSize	Font size of the value display.	Double
ValueFontColor	Font size of the value display.	Color
IsGlasReflection	Activate the glass effect on the element.	Boolean
KnobBrush	Color of the knob.	Brush
IndicatorBrush	Color of the indicator.	Brush
IndicatorBackgroundBrush	Background color of the inactive indicator.	Brush
KnobSize	Diameter of the knob.	Double
KnobIndicatorSize	Indicator size of the knob.	Size
ElementSize	Size of the element.	Size
VisibilityKnob	Activating of the knob.	Boolean
ValuePosition	Position of the value display.	Double
ValueVisibility	Activating the value display.	Boolean

Waterfall diagram

The waterfall diagram, WPF element is available to partners of COPA-DATA and is available to these via the Partner Portal.

The meaning and waterfall chart wizard must be used to model a waterfall diagram. A waterfall can be modeled with this wizard. The information is saved directly for the variables concerned in the **Analyzer** --> **Parameters for waterfall diagram**.

An example of a waterfall diagram in Runtime is shown below:



Note: This screenshot is only available in English.

The following settings can be made in the WPF configuration window under **COPADATA-ELEMENT**:

Property	Function	Value
zenonRefreshRate	Time between the refreshes of the diagram in ms.	Integer
zenonWaterfallIdentifier	Name of the waterfall diagram.	String
zenonZSystemModel	Equipment group of the variables used.	String

Note: Additional VSTA programming is necessary for the display of the waterfall diagram in the zenon web client. You can find details on this in the display of WPF elements in the zenon web client (on page 135).

6.3.6 Display of WPF elements in the zenon web client

In order to also be able to also use the pre-made WPF elements **"energy class diagram"**, **"Sankey diagram"** and **"waterfall chart"** for the display in a zenon web client, amendments are necessary in the project:

- ▶ Engineering in the zenon Editor (on page 136)
- ▶ Adapt VSTA code (on page 136)

Engineering in the zenon Editor

Carry out the following project configuration steps in the zenon Editor, in order to also be able to display certain WPF elements in the zenon web client:

PLACE WPF IN THE ZENON SCREEN:

- ▶ Place the WPF element in a zenon screen.
- ▶ Give it a unique name in the **Element name** property.
You can find this property in the **General** properties group.
Note: A warning dialog appears if the name for an element has already been issued in another screen.
- ▶ Use the element name issued here in the VSTA code.

VSTA code (complex)

In order to add the programmer code for the display of WPF elements in the zenon web client, carry out the following steps:

1. In the zenon Editor, switch to the **programmer interfaces** node.
2. Select the **VSTA** node and select the **Open VSTA Editor with project add-in...** with a right mouse click
3. The dialog to create a VSTA project is opened.
4. Select the C# entry in the **Create new VSTA project** dialog.
5. Create (copy) the code below.
6. Enter the name of the WPF element in the code.

Note: When opening the VSTA editor, note whether the content of the following code is already present in the project configuration. For the display of the WPF element in the web client, compare the existing code and undertake the necessary additions. Please note the comments in relation to this in the model code.

VSTA CODE

```
//As member:
zenOn.IDynPictures zScreens = null;

string[] WPFElements={"WPF_Control", "WPFWebclient_1", "WPFWebclient_2"}; //Names of the
WPF screen elements that appear in the zenon project and that need access to the API (as
many/few as you want)
```



```
//Add the following three lines of code in the project archive function:
void ThisProject_Active()
{
    zScreens = this.DynPictures();
    zScreens.Open += new zenOn.DDynPicturesEvents_OpenEventHandler(zScreens_Open);
    zScreens.Close += new zenOn.DDynPicturesEvents_CloseEventHandler(zScreens_Close);
}

//Add the following two lines of code in the project inactive function:
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();
}

//Add two new event handlers:
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 (simplified)

If only one WPF element is used in a zenon screen, the following more streamlined code can be used as an alternative. To do this, the names of the WPF element, and the screen in which the element is used, must be entered. This code is then recommended if, for each project, only one of the pre-made WPF elements is used.

VSTA CODE

```

zenOn.IDynPicture zScreen = zero;
string wpfElement = "WPF_Control"; //Name of the WPF element in the screen
string wpfPicture = "@Details_Overview_Online"; //Name of the zenon screen

//Add to the project active function:
void ThisProject_Active()
{
    zScreen = this.DynPictures().Item(wpfPicture);
    zScreen.Open += new zenOn.OpenEventHandler(zScreen_Open);
    zScreen.Close += new zenOn.CloseEventHandler(zScreen_Close);
}

//Add to the project inactive function:
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;
    }
}

```

6.3.7 Examples: Integration of WPF in zenon

You can see how XAML files are created and integrated as WPF elements in zenon from the following examples:

- ▶ Integrate button as WPF XAML in zenon (on page 145)
- ▶ Integrate bar graph as WPF XAML in zenon (on page 139)
- ▶ Integrate DataGrid Control in zenon (on page 151)

Integrate bar graph as WPF XAML in zenon

Example structure:

- ▶ Creating a bar graph (on page 85) in Adobe Illustrator and converting it to WPF
- ▶ Integrate into zenon
- ▶ Linking with variables
- ▶ Adapting the bar graph WPF element

CREATE BAR GRAPH

The first step is to generate a bar graph as described in the Workflow with Adobe Illustrator (on page 85) chapter. To be able to use the XAML file in zenon, insert this in the project tree in the **Files/graphics** folder.

INTEGRATE BAR GRAPH

Note: A zenon project with the following content is used for the following description:

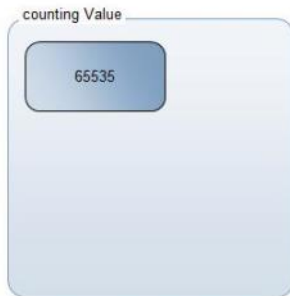
- ▶ An empty screen as a start screen
- ▶ Four variables from the internal driver for
 - Scale 0
 - Scale `central`
 - Scale `high`
 - Current value
- ▶ A variable from the mathematics driver for displaying the current value (255)

To integrate the bar graph:

1. open the empty screen
2. place a **WPF element** (on page 94) in the screen
3. select **XAML file** in the properties window
4. Select the desired XAML file (for example **bar graph_vertical.xaml**) and close the dialog

ADJUST BAR GRAPH

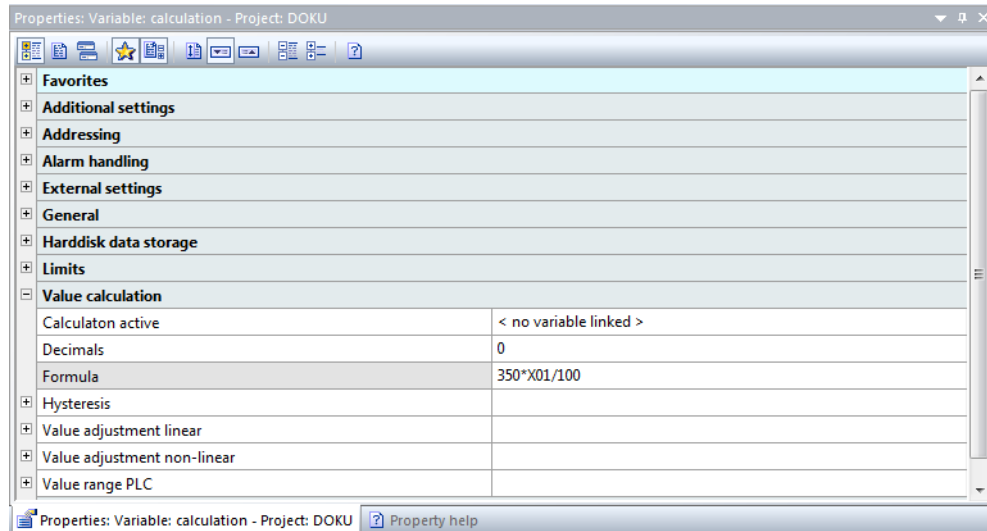
Before configuration, the scale of the XAML file is adapted if necessary:



To do this:

- Create a new mathematics variable that calculates the new value in relation to the scaling, for example:
- Variable: 0-1000

- Mathematic variable {value created in xaml file}*Variable/1000

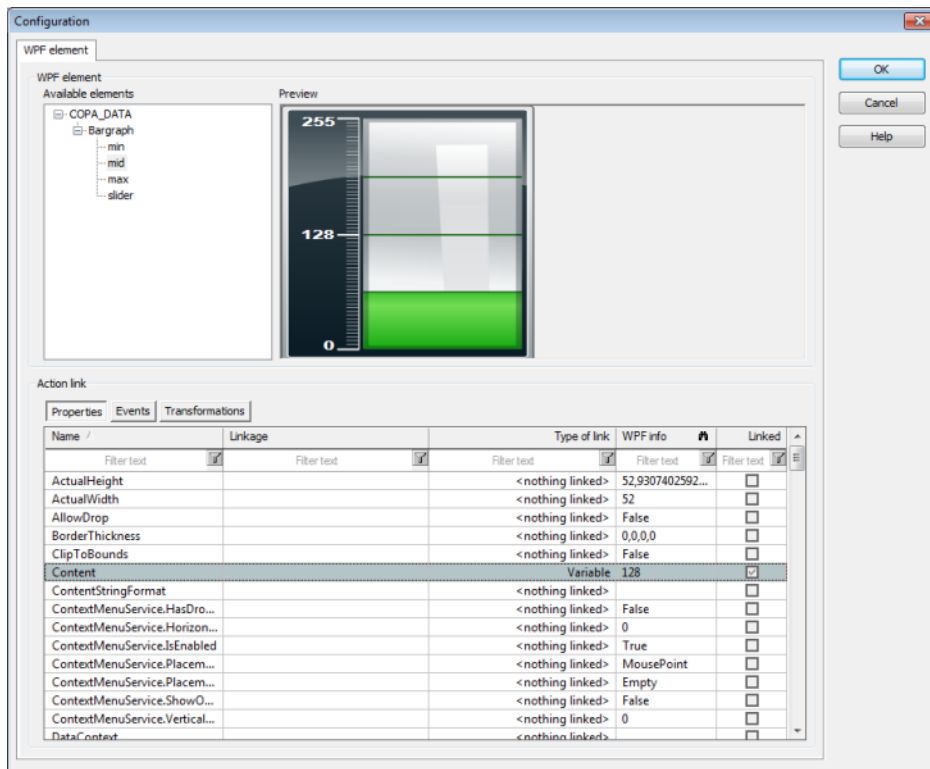


The XAML file is then configured.

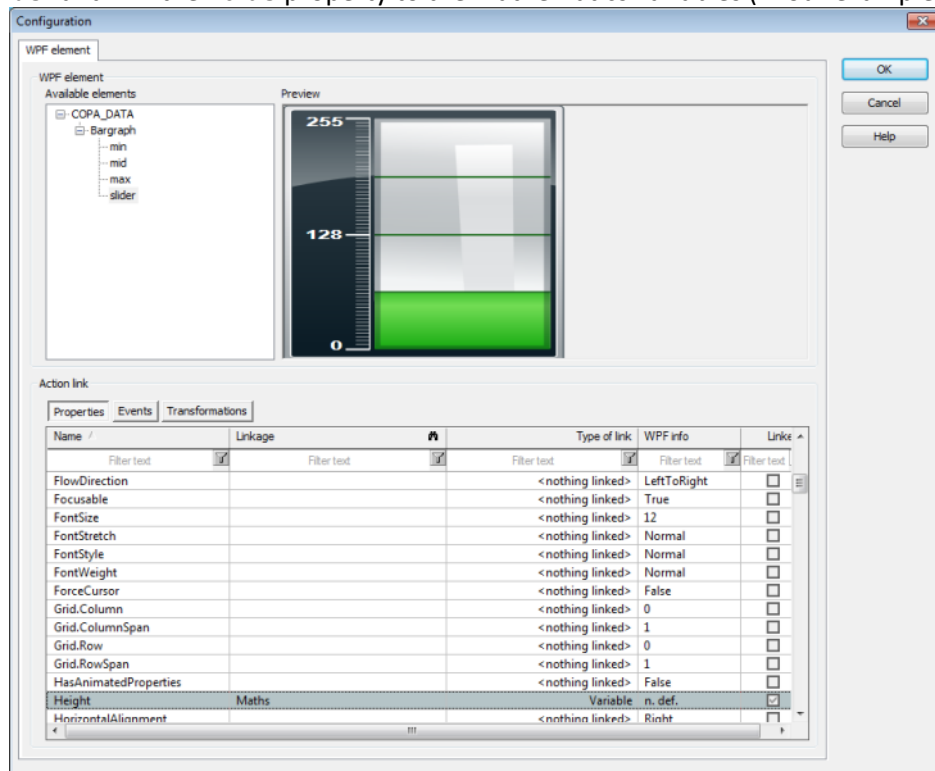
CONFIGURE BAR GRAPH

1. Click on the WPF element and select the **Configuration** property
2. The configuration dialog shows a preview of the selected XAML file.

3. Select the minimum value, the average value and the maximum value and link each of these to the corresponding variable in the **Content** property

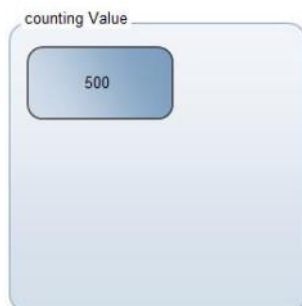


4. Select the **Slider** and link the **Value** property to the mathematics variables (in our example:



calculation)

5. Check the project planning in Runtime:



Integrate button as WPF XAML in zenon

Example structure:

- ▶ Creating a button (on page 81) in Microsoft Expression Blend
- ▶ Integrate into zenon
- ▶ Link to a variable and a function
- ▶ adjust the button to the size of the element
- ▶ Create button

As a first step, create a button as described in the Create button as XAML file with Microsoft Expression Blend (on page 81) chapter. To be able to use the XAML file in zenon, insert this in the project tree in the **Files/graphics** folder.

INTEGRATE BUTTON

Note: A zenon project with the following content is used for the following description:

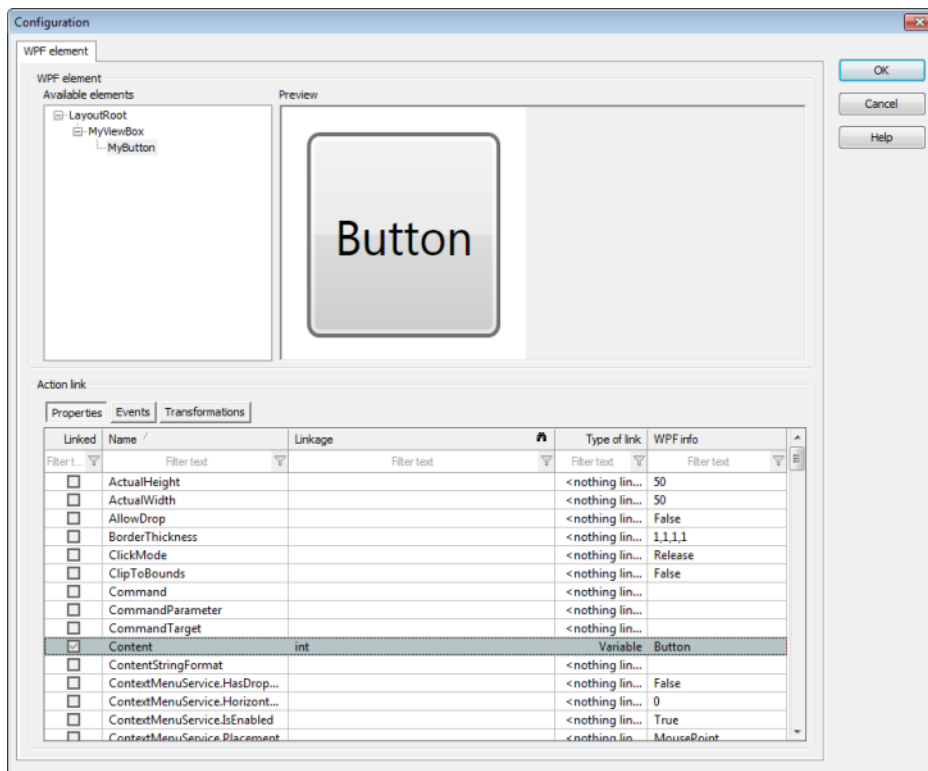
- ▶ An empty screen as a start screen
- ▶ an internal variable **int** of type **Int**
- ▶ a function **Funktion_0** of type **Send value to hardware** with:
 - **Direct to hardware** option activated
 - Set was set to 45

To integrate the button:

1. open the empty screen
2. place a **WPF element** (on page 94) in the screen
3. select **XAML file** in the properties window
4. select the XAML file (e. g. **MyButton.xaml** and close the dialog
5. select the **Configuration** property

CONFIGURE THE BUTTON

The configuration dialog shows a preview of the selected XAML file. All elements named in the XAML file are listed in the tree:

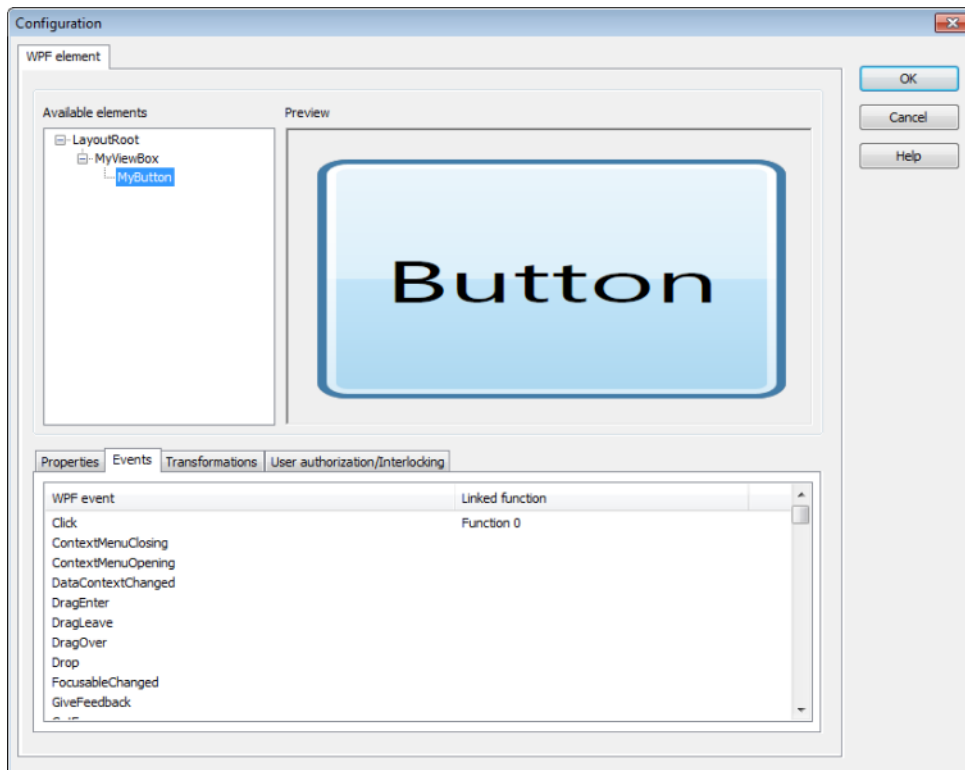


1. select the WPF button, which is in LayoutRoot->MyViewBox->MyButton
2. Look in the **Properties** Entry **Content** tab; this contains the button's text
3. Click the **Link type** column
4. Select **Variable** from the drop down list
5. Click in the **Link** column
6. the variable selection dialog is opened
7. select the `int` variable to link this variable with the **Content** property

EVENTS

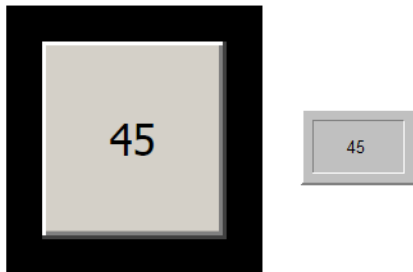
To also assign events:

1. select the events tab



2. look for the 'Click' entry, this event is triggered by the WPF element, as soon as the button is clicked
3. Click in the **Link type** column
4. Select **Function** from the drop down list
5. Click in the **Link** column
6. the **function selection dialog** is opened
7. select **Function_0**
8. Confirm the changes with **OK**
9. Insert a **numerical value element** into the screen
10. Link this **numerical value element** to the `int` variables too.
11. Compile the Runtime files and start Runtime.

The **WPF element** is displayed in Runtime, the button text is 0. As soon as you click on the button, the **click** event is triggered and the **set value** function is carried out. The value 45 is sent directly to the hardware and both **numerical value** and **button** display the value 45 .

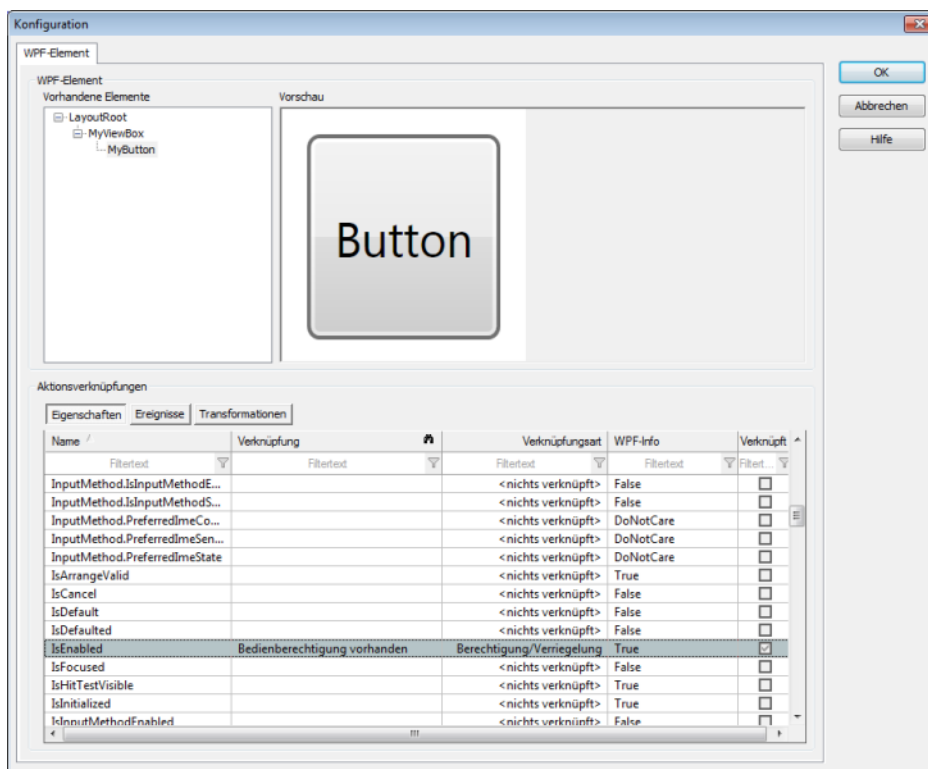


Define a set value of 30 via the **numerical value element**; this value is then also assumed by the **WPF element**.

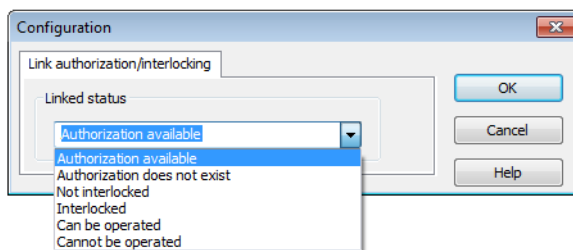
AUTHORIZATION

Similar to a **numerical value**, a **WPF element** can be locked according to authorizations (lock symbol) or switched to be operable. Set the user authorization level to 1 for the **WPF element** and create a user called **Test** with **authorization level 1**. In addition, set up the functions **Login with dialog** and **Logout** . You link these two functions with 2 new text buttons on the screen.

In the **WPF element** configuration dialog, select the **MyButton** WPF button and select the **Properties:** tab

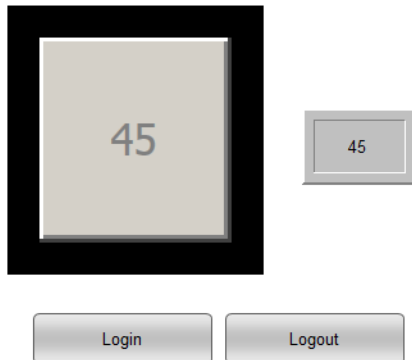


1. Select the **IsEnabled** element
2. Click in the **Link type** column
3. Select **Authorizations/interlocking** from the drop down list
4. Click in the **Link** column
5. In the drop-down list, select the **Authorized** option



6. Close the dialog with **OK**

Compile the Runtime file and note that Authorizations to be Transferred must also be selected. After Runtime has been started, the WPF button is displayed as deactivated on the screen and cannot be operated. If you now log in as the user **Test**, the button is activated and can be operated. The button is locked again as soon as you log out.



TRANSFORMATION

The XAML files must still be adapted to use transformations:

1. switch to the **Expression Blend** program
2. select **MyButton**, so that the properties of the element are visible in the events window

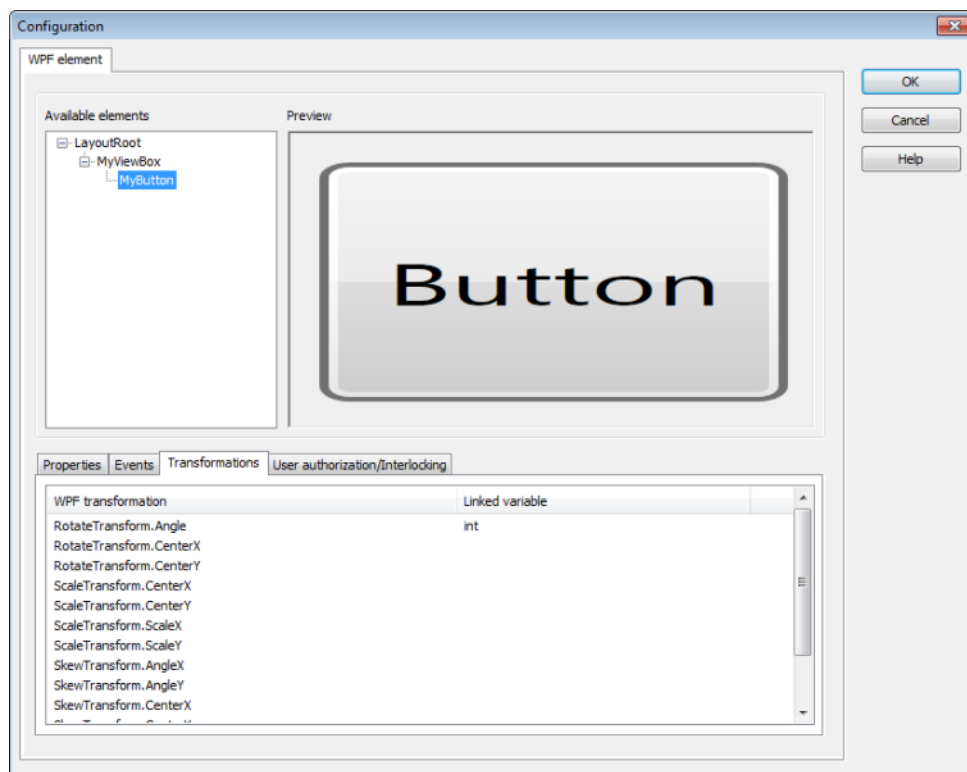


3. Under **Transform** at **RenderTransform** select the **Apply relative transform** option

As a result of this, a block is inserted into the XAML file, which save the transformation settings in runtime.

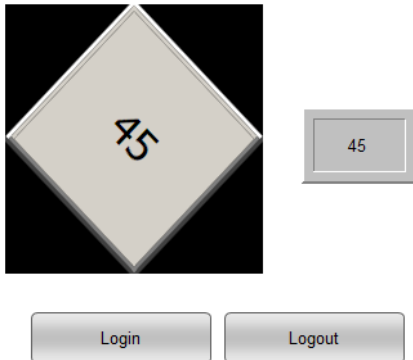
```
<Button.RenderTransform>
  <TransformGroup>
    <ScaleTransform ScaleX="1" ScaleY="1"/>
    <SkewTransform AngleX="0" AngleY="0"/>
    <RotateTransform Angle="0"/>
    <TranslateTransform X="0" Y="0"/>
  </TransformGroup>
</Button.RenderTransform>
```

4. Save the file and replace the old version in zenon with this new file.
5. Open the **WPF element** configuration dialog again:
 - a) select the **MyButton** button
 - b) select the **Transformations** tab



- c) select the **RotateTransform.Angle** element
- d) Click in the **Link type** column
- e) Select **Transformations** from the drop down list
- f) Click in the **Link** column
- g) the variable selection dialog is opened
- h) select the **int** variable to link this variable with the **RotateTransform.Angle** property

Compile the Runtime files and start Runtime. Log in as the **Test** user and click on the button. The button has the value 45 and the **WPF element** rotates by 45°.



Integrate DataGrid Control in zenon

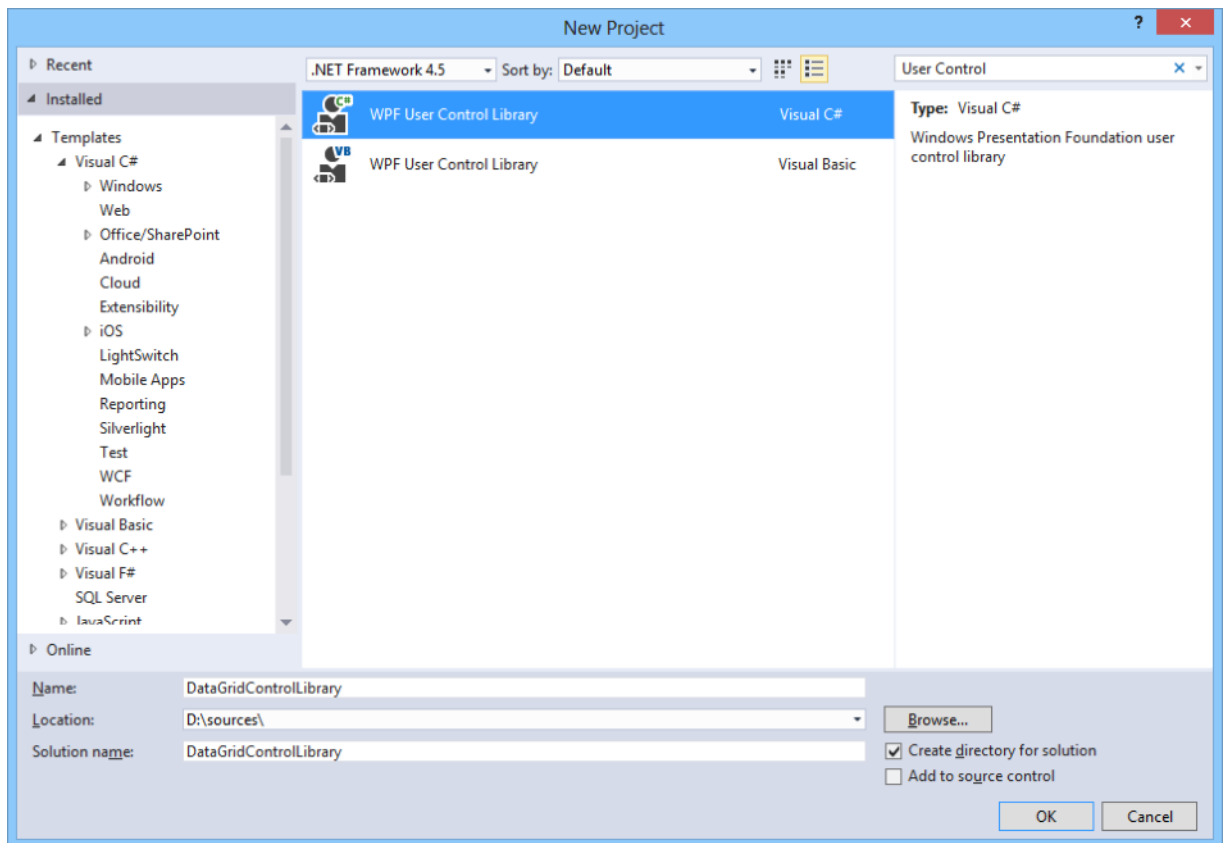
To create DataGrid control for zenon, you need:

- Visual Studio (Visual Studio 2015 in this example)

CREATE WPF USER CONTROL

1. in Visual Studio, create a new **Solution** and a **WPF User Control Library** project in .NET Framework version 4 or higher therein.

Info: If the corresponding project template does not appear in the list of available templates, this can be added by means of the search (field at the top right of the dialog).



In our example, the project is given the name **DataGridControlLibrary**.

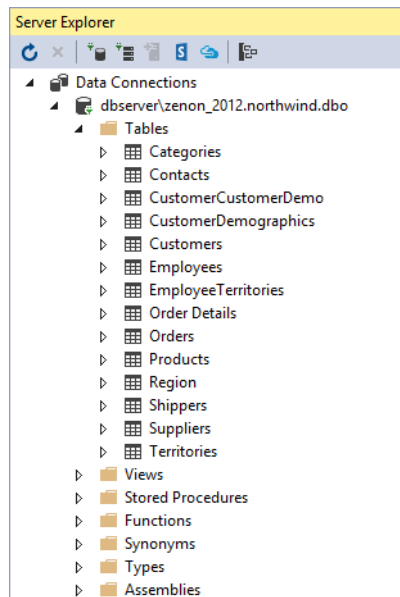
2. Create a new data connection in the **Server Explorer**.

In our example, the database `Northwind` is used, which is provided by Microsoft as an example database that can be downloaded for free.

To set up the database connection:

- a) Right-click on **Data Connections**.
- b) Select **Add connection....**
- c) Select `Microsoft SQL Server (SQLClient)` as **Data source**.
- d) Select the corresponding server and database name.

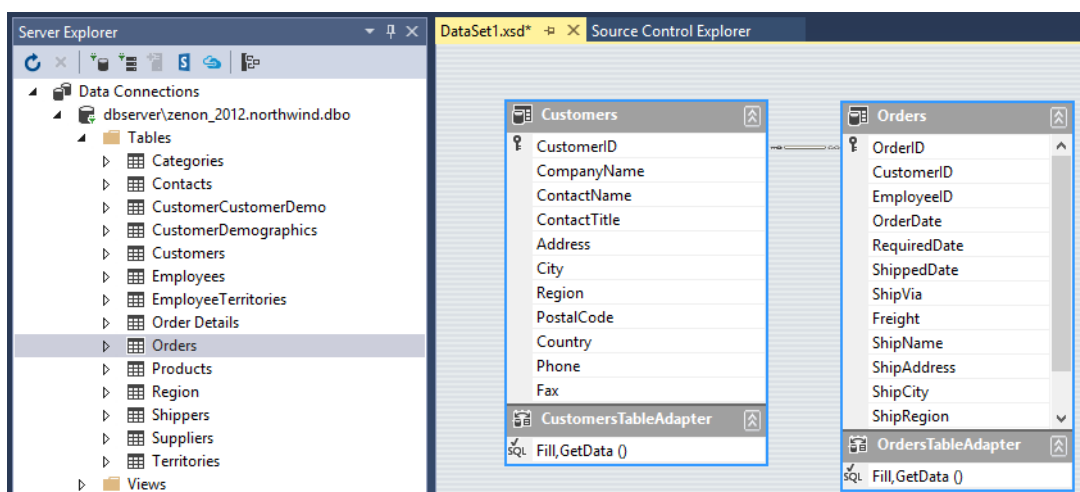
After adding the connection, the Server Explorer window should look a little like this:



A new DataSet is created in the next step.

CREATING A DATASET

1. Right-click on the project
2. Select **Add – New Item...** in the context menu
3. Create a new **DataSet** with the name `DataSet1`.
4. Double click on the DataSet in order to open it in the Designer.
5. Drag the tables that you need (`Customers` and `Orders` in this example) to the DataSet design window.



The XAML file is modified in the next step.

CONFIGURATION OF THE XAML FILE

1. If not already there, add the **Namespace** as a reference to the class in the XAML file:

```
<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. Define the resources and the DataGrid that is to be used in the WPF:

```
<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. Open the code-behind file (**UserControl1.xaml.cs**) and insert the following lines in the constructor:

```
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();
}
```

In doing so, the following happens:

- The DataSet is obtained
- A new TableAdapter is created
- The DataSet is filled

- The information is provided to the DataGrid control

The solution can now be built.

BUILD

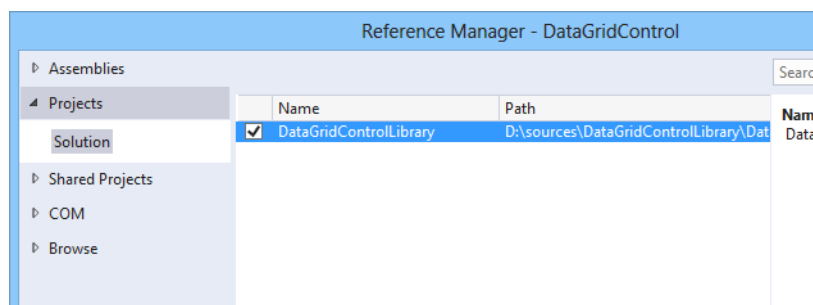
Now build the solution. The corresponding DLL (**DataGridControlLibrary.dll**) is created in the output folder of the project.

Now you have a DLL with the necessary functionality available.

However zenon can only display XAML files that cannot be linked to the code behind file, which is why an additional XAML file is needed that references the DLL that has just been created.

To do this:

1. Create a further project, again as a **WPF User Control Library**
2. It was called **DataGridControl** in our example.
3. Insert a reference to the project that has just been built into this new project.



4. The XAML files (**UserControl1.xaml**) looks as follows:

```
<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. Because all necessary content is contained in the DLL that has been created and no code-behind is necessary, delete the following lines:

```
x:Class="DataGridControl.UserControl1"
xmlns:local="clr-namespace:DataGridControl"
```

6. Also delete (for the positioning) the following lines:

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

7. Delete the code-behind file (**UserControl1.xaml.cs**) in this project.
8. Define what is to be displayed in the XAML file.

To do this, modify the XAML file as follows:

```
<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>
```

The

line `xmlns:dataGridLibrary="clr-namespace:DataGridControlLibrary;assembly=DataGridControlLibrary"` defines the namespace **dataGridLibrary** and stipulates that this should use the assembly that has been created.

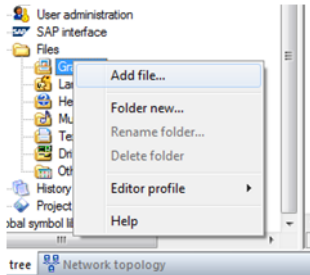
9. Assign a name for the grid.
10. Insert the control **dataGridLibrary:UserControl1** from our library and give it a name, because zenon can only modify objects that have a name.
11. Build the solution.

In the next step, how the DLL and XAML file are added to zenon is explained.

STEPS IN ZENON

1. Open the zenon Editor
2. Go to File -> Graphics.

3. Select **Add file...** in the context menu



4. Select the XAML file at the save location (**UserControl1.xaml** from the **DataGridControl** project) and add this:

Status	File name	Type	Size	Preview
Filter text	Filter text	Filter...	Filter...	Filter text
	UserControl1.xaml	xaml	0 KB	

5. Insert the DLL with the functionality for the XAML file.

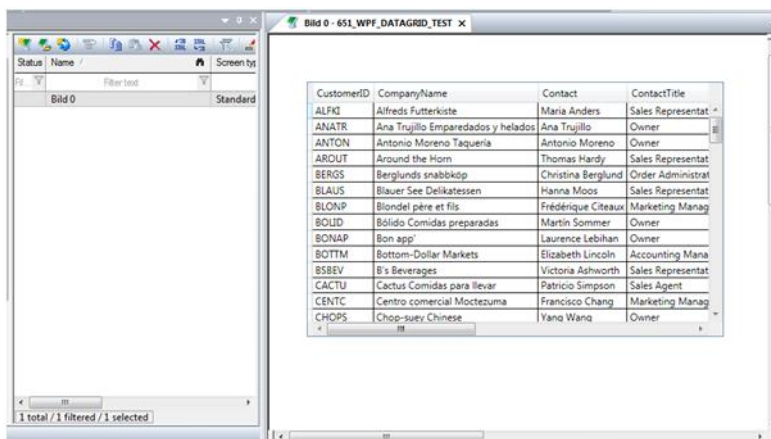
To do this:

- a) Select, in the context menu, File -> Other **Add file...**
- b) Select the file **DataGridControlLibrary.dll** of the first project (**DataGridControlLibrary**).

Status	File name	Type	Size
Filter text	Filter text	Filter...	Filter...
	DataGridControlLibrary.dll	dll	36 KB

6. Create a zenon screen.
7. Add a WPF element and select the previously-incorporated XAML file.

You should now see the following in the zenon Editor:



8. Start zenon Runtime in order to also test the control there.



Attention

Assemblies are only removed after loading when the application is ended. This means:

*If a WPF file with a referenced assembly in zenon is displayed, then this assembly is loaded in the memory until zenon is ended, even if the screen is closed again. If you would like to remove an assembly from the *Files/Other* folder, the Editor must first be restarted, so that the assembly is removed.*

6.3.8 Error handling

ENTRIES IN LOG FILES

Entry	Level	Meaning
Xaml file found in %s with different name, using default!	Warning	The name of the collective file and the name of the XAML file contained therein do not correspond. To avoid internal conflicts, the file with the name of the collective file and the suffix .xaml is used.
no preview image found in %s	Warning	The collective file does not contain a valid preview graphic (preview.png or [names of the XAML file].png). Thus no preview can be displayed.
Xaml file in %s not found or not unique!	Error	The collective file does not contain an XAML file or several files with the suffix .xaml . It cannot be used.
Could not remove old assembly %s	Warning	There is an assembly that is to be replaced with a newer version, but cannot be deleted.
Could not copy new assembly %s	Error	A new version is available for an assembly in the work folder, but it cannot be copied there. Possible reason: The old example is still loaded, for example. The old version continues to be used, the new version cannot be used,
file exception in %s	Error	A file error occurred when accessing a collective file.
Generic exception in %s	Error	A general error occurred when accessing a collective file.