zenon
by COPA-DATA

zenon manual

Controls

v.8.10

Contents

1 Welcome to COPA-DATA help

ZENON VIDEO-TUTORIALS

You can find practical examples for project configuration with zenon in our YouTube channel (https://www.copadata.com/tutorial_menu). The tutorials are grouped according to topics and give an initial insight into working with different zenon modules. All tutorials are available in English.

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.

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.

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.

2 Controls

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

- .NET user controls (on page 42) (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

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

> **Attention**
>
> 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

- Select, in **Project References**, via **Add References...**, the **zenon Runtime 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.ReleaseComObject(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:	The control can use zenon variables. 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).
0:	The control cannot use zenon variables or does not have the method. You can state variables with all types without restricting the number. In the Runtime however they only can be used with VBA.

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 9) 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 13)
- Example CD_SliderCtrl (C++) (on page 22)
- Example :NET control as ActiveX (C#) (on page 30)

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. Additionally the properties of the control can also be defined by the dynamic element.

For this the interface (dispatch interface) of the control must support a number 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 9)
- zenonExitEd (on page 9)
- 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.

4.1.1.1 CanUseVariables

Prototype: short CanUseVariables();

This method either returns *1* or *0*

Value	Description
1:	The control can use zenon variables. 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).
0:	The control cannot use zenon variables or does not have the method. You can state variables with all types without restricting the number. In the Runtime however they only can be used with VBA.

4.1.1.2 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.

4.1.1.3 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

4.1.1.4 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.

4.1.1.5 zenonExitEd

Equals zenonExit (on page 9) 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.

4.1.1.6 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**.

4.1.1.7 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.

4.2 Example LatchedSwitch (C++)

The following example describes an ActiveX control, that realises 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 17)
- VariableTypes (on page 17)
- MaxVariables (on page 18)
- zenonInit (on page 18)
- zenonExit (on page 19)

4.2.3.1 CanUseVariables

This method returns 1, so zenon variables can be used.

```
short CLatchedSwitchCtrl::CanUseVariables()
{

return 1;
}
```

4.2.3.2 VariableTypes

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

```
short CLatchedSwitchCtrl::VariableTypes()
{

return 0x0004;    // Only bit variables
}
```

4.2.3.3 MaxVariables

Two variables can be used. Therfore *2* is returned.

```
short CLatchedSwitchCtrl::MaxVariables()
{

return 2; // 2 variables
}
```

4.2.3.4 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 *IUnknow* 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.

4.2.3.5 zenonExit

This method releases the dispatch driver.

```
BOOL CLatchedSwitchCtrl::zenonExit()
{

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

4.2.4 Operate and display

4.2.4.1 Setting values

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);
}
COleControl::OnLButtonDown(nFlags, point);
}
```

4.2.4.2 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 theses 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 **IVaribale** 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 26)
- VariableTypes (on page 26)
- MaxVariables (on page 27)
- zenonInit (on page 27)
- zenonExit (on page 28)

4.3.3.1 CanUseVariables

This method returns *1* so zenon variables can be used.

```
short CCD_SliderCtrlCtrl::CanUseVariables()
{

return 1;
}
```

4.3.3.2 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
}
```

4.3.3.3 MaxVariables

Only one variable can be linked to this control.

```
short CCD_SliderCtrlCtrl::MaxVariables()
{

return 1; // 1 variables
}
```

4.3.3.4 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;
}
```

4.3.3.5 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 Operate and display

4.3.4.1 Drawing

With **DoSuperclassPaint** the SliderCtrl is drawn (as is 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);
}
}
```

4.3.4.2 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 theses 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:

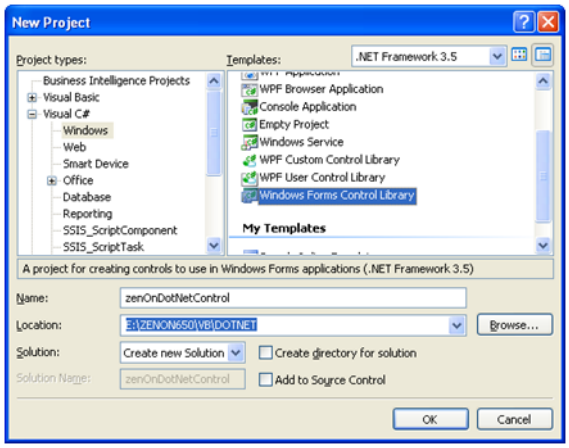
1. Create Windows Form Control (on page 30)
2. Change .NET User Control to dual control (on page 33)
3. Work via VBA with ActiveX in the Editor (on page 37)
4. <Connect CD_PRODUCTNAME> variables with the .NET user control (on page 38)

> **Attention**
>
> When using zenon COM objects with self-created user controls or external applications, they must be enabled using the **Marshal.ReleaseComObject** method. Enabling by means of the **Marshal.FinalReleaseComObject** method must not be used, because this leads to a malfunction of zenon Add-ins.
>
> **Information**
>
> The screenshots for this theme are only available in English.

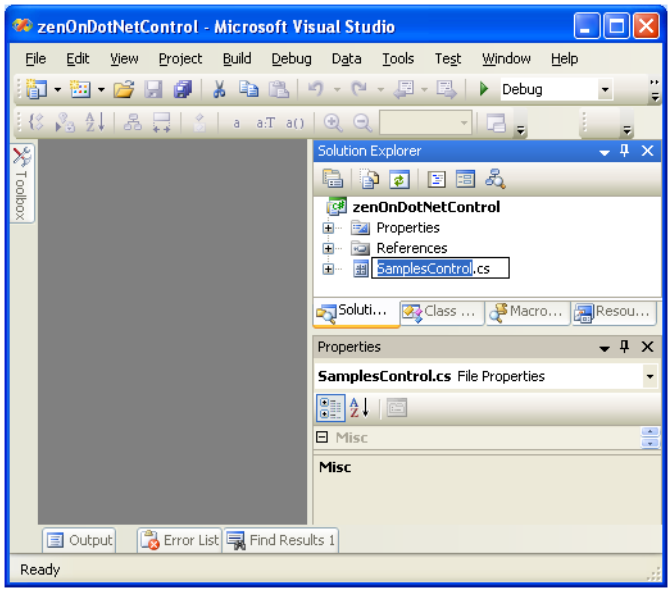
4.4.1 Creat Windows Form Control

To create a Windows Form Control:

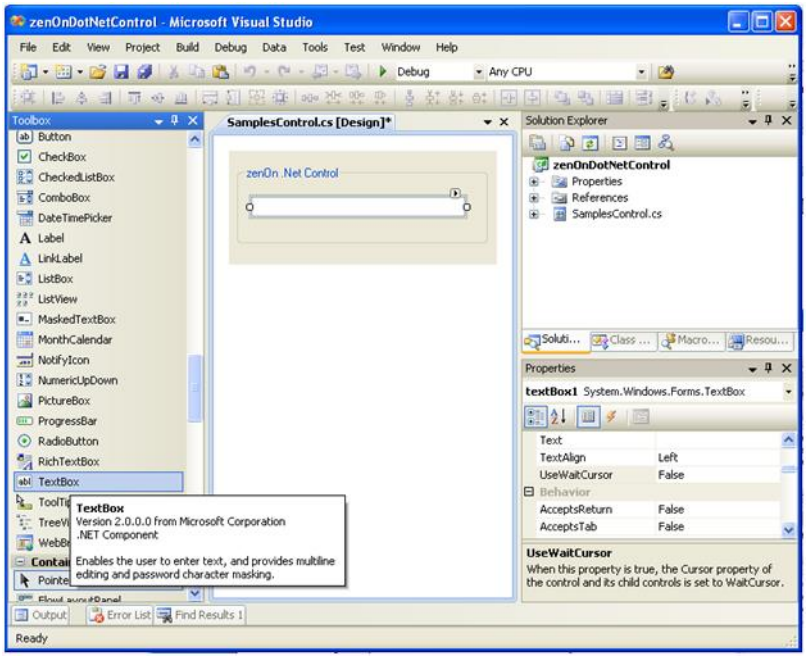
1. Start Visual Studio 2008 and create a new Windows **Form Control Library** project:



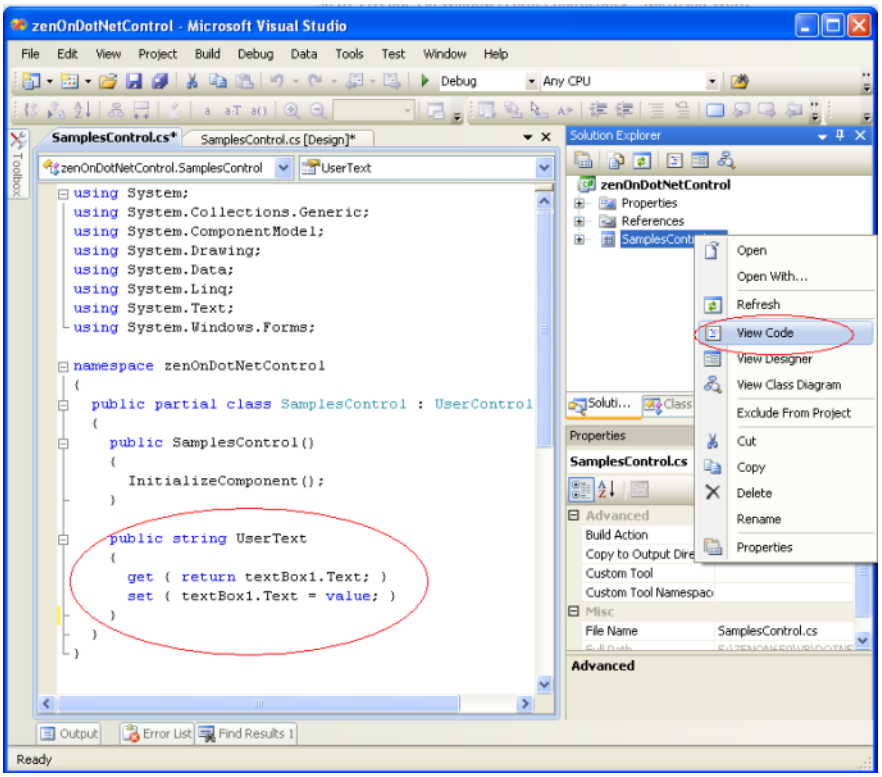
2. Rename the default control to the desired control name.
   In our example: **SampesControl.cs**.



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



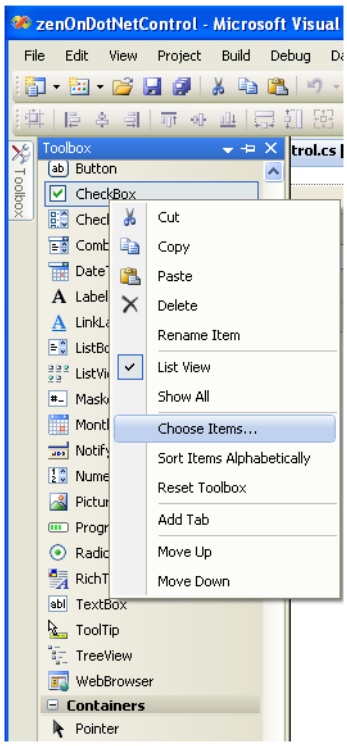
4. Normally controls have properties. Open the Code Designer via **View Code** and ass the desired properties which should be available externally.
   In our example: Externally visible property „**UserText**" with **get** and **set** access which contains the text of the text box:



5. Compile the project.

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

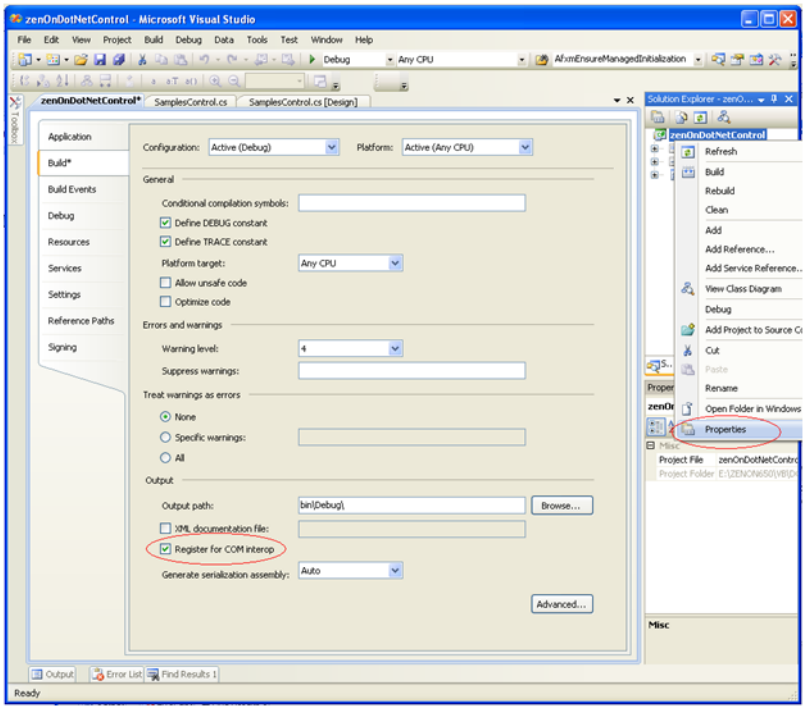
Important: The control must be inserted manually in the control tool box via **Choose Items**.



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.

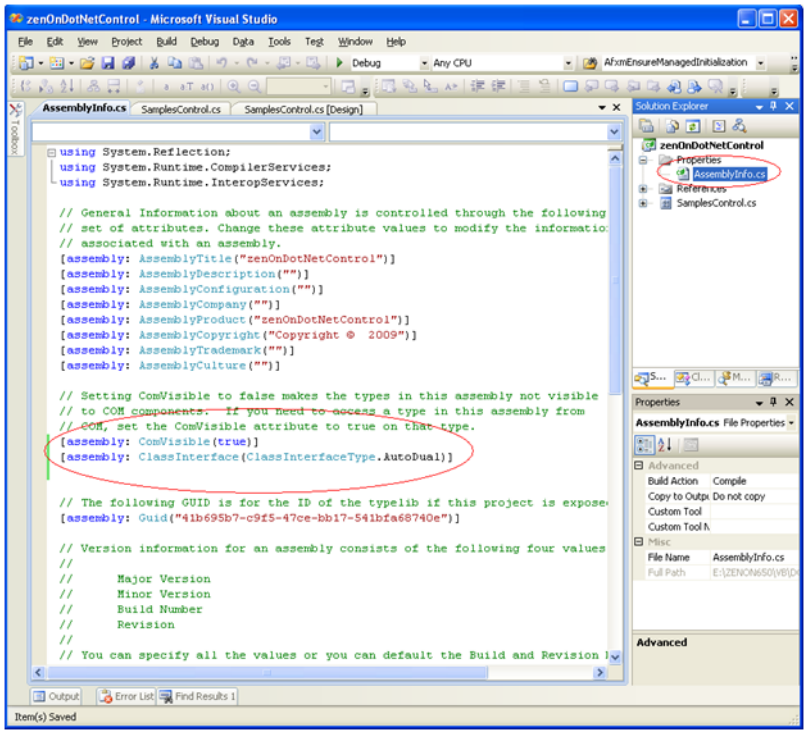
1. Open the project and activate property **Register for COM interop** in the **Build** settings:



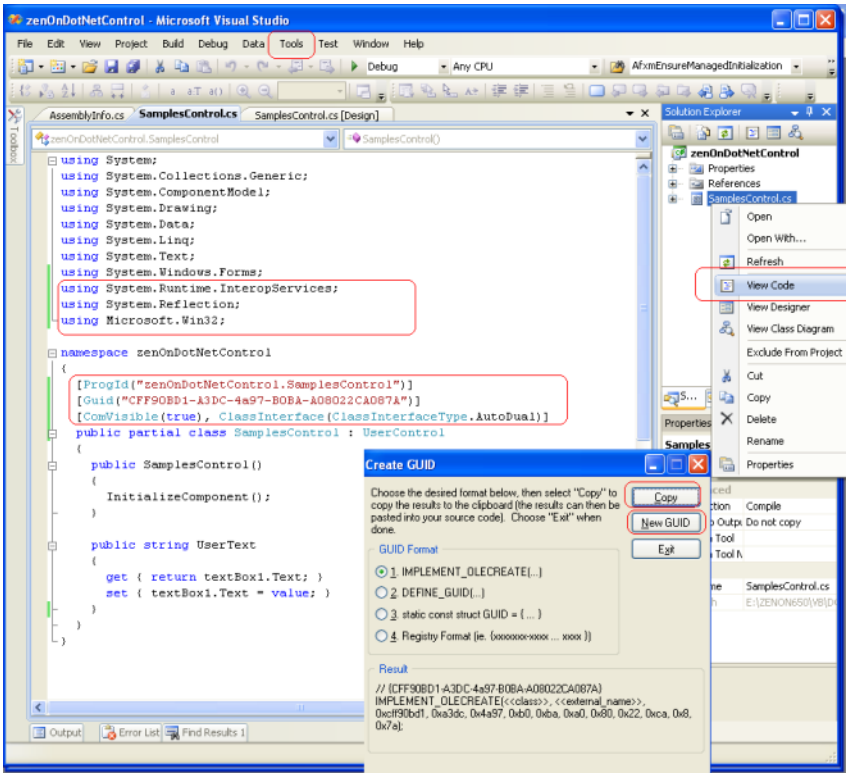
2. Open the file **AssemblyInfo.cs** and
   - set attribute **ComVisible** to *true*
   - add attribute **ClassInterface**

   [assembly: ComVisible(true)]

[assembly: ClassInterface(ClassInterfaceType.AutoDual)]

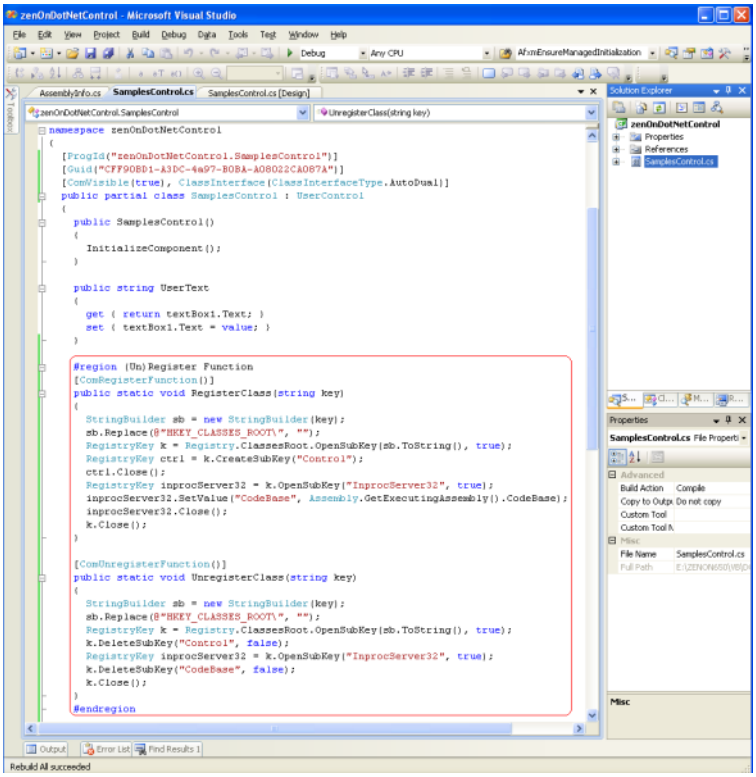


3. Open the code designer via **View Code** and add the necessary ActiveX attributes and **using** entries. Via menu **Tools/Create GUID** create a new GUID for the GUID attribute:



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.

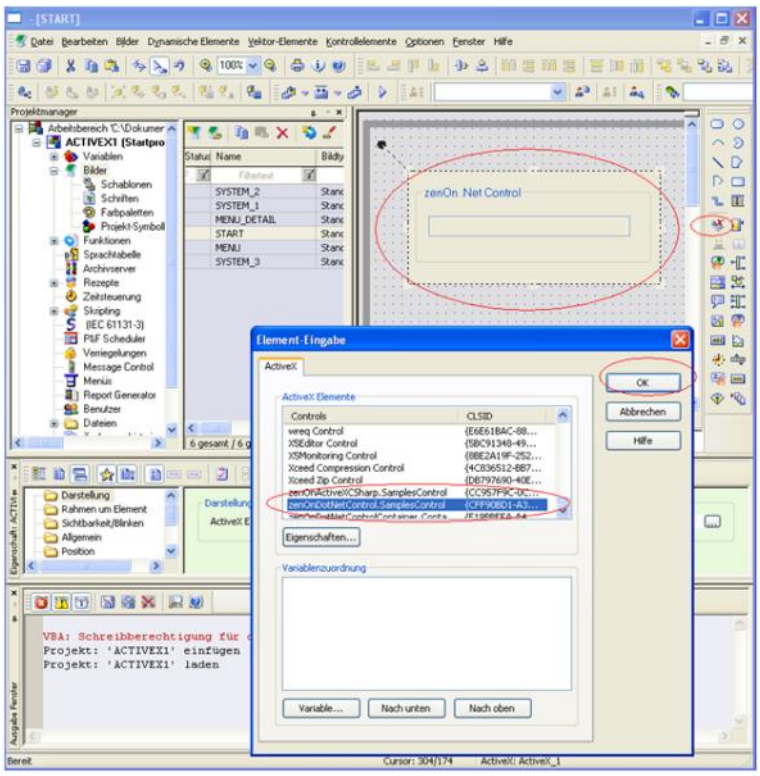
   The Windows Form Control is now ActiveX-able and was registered automatically during the rebuild. An additional typelib file **zenOnDotNetControl.tlb** was created in the output directory.

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:



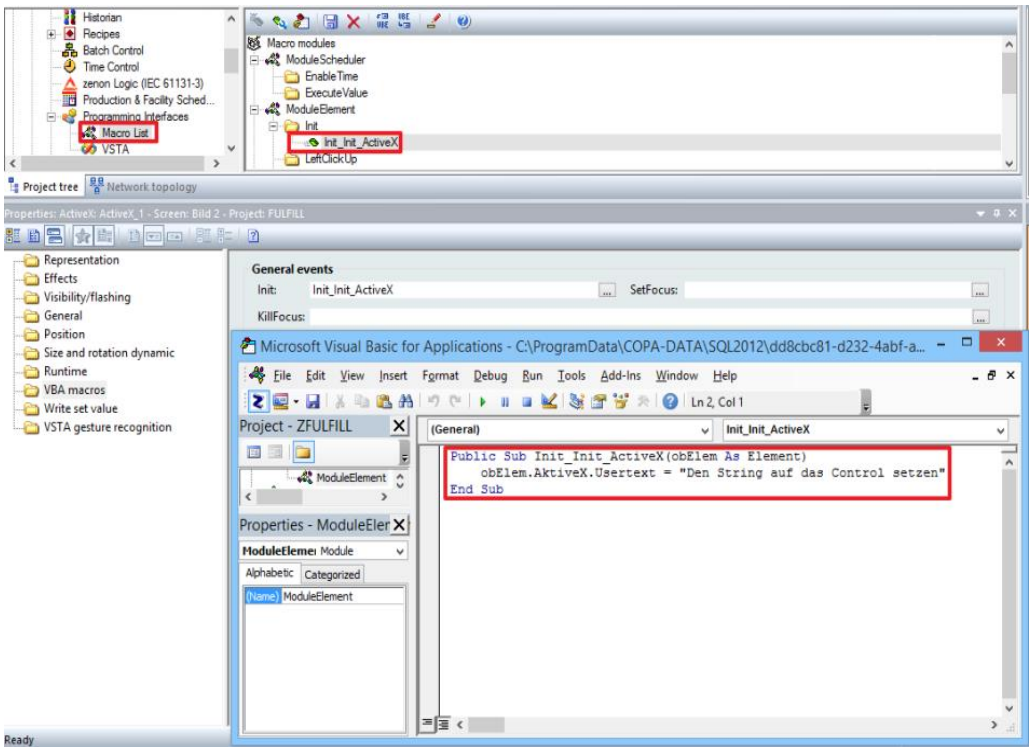
4.4.3 Work via VBA with ActiveX in the Editor

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.AktiveX.Usertext = "Set the string to the control"
End Sub
```

4.4.4 <Connect CD_PRODUCTNAME> 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.

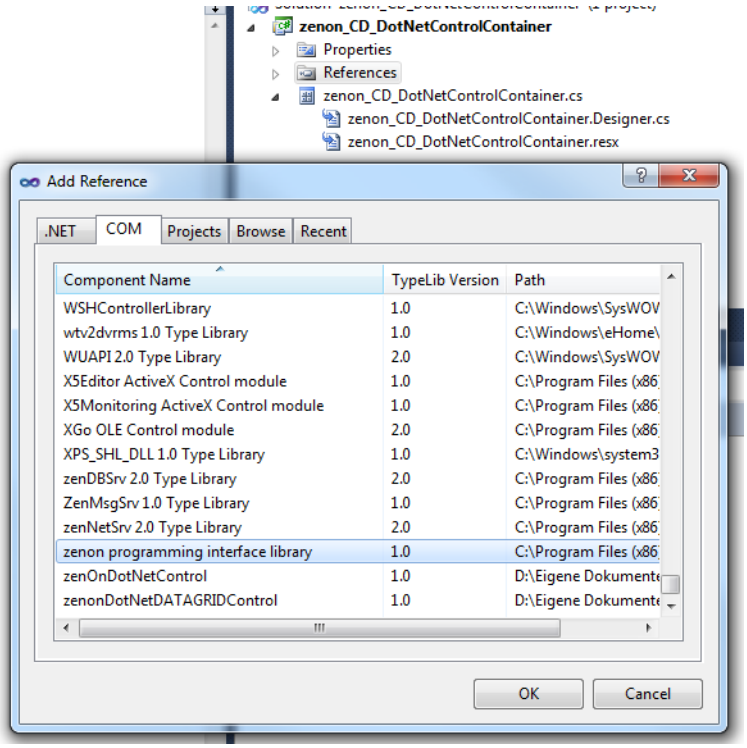
NECESSARY METHODS

- public bool zenOnInit (on page 40) (Is called up during control initializing in the zenon Runtime.)
- public bool zenOnInitED (on page 40) (Is used in the Editor.)
- public bool zenOnExit() (on page 41) (Is called up during control destruction in the zenon Runtime.)
- public bool zenOnExitED() (on page 41) (Is used in the Editor.)
- public short CanUseVariables() (on page 41) (Supports linking variables.)

- public short VariableTypes() (on page 41) (Supported data types by the control)
- public MaxVariables() (on page 42)(Maximum number of variables which can be linked to the control.)

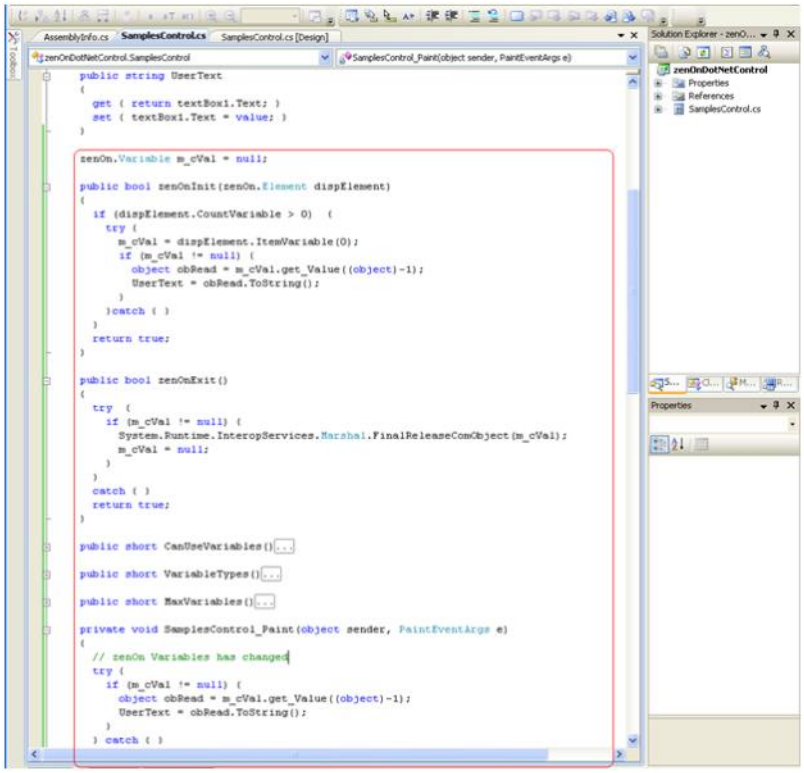
ADD REFERENCE

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.



4.4.4.1 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

4.4.4.2 public bool zenOnInitED(zenOn.Element dispElement)

Equals public bool zenOnInit (on page 40) and is executed when opening the ActiveX in the Editor (double click on ActiveX).

4.4.4.3 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.

4.4.4.4 public bool zenOnExitED()

Equals public bool zenOnExit() (on page 41) and is executed in closing the ActiveX in the Editor. With this you can react to changes, e.g. value changes, in the Editor.

4.4.4.5 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.

4.4.4.6 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*

4.4.4.7 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 42)
- Example .NET control container (on page 43)
- Example :NET control as ActiveX (C#) (on page 30)

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

> **Attention**
>
> When using zenon COM objects with self-created user controls or external applications, they must be enabled using the **Marshal.ReleaseComObject** method. Enabling by means of the **Marshal.FinalReleaseComObject** method must not be used, because this leads to a malfunction of zenon Add-ins.

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 (regsrv32).
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	Easy access via VBA and VSTA.

CD_DotNetControlContainer control	ActiveX control
possible via the CD_DotNetControlContainer method.	

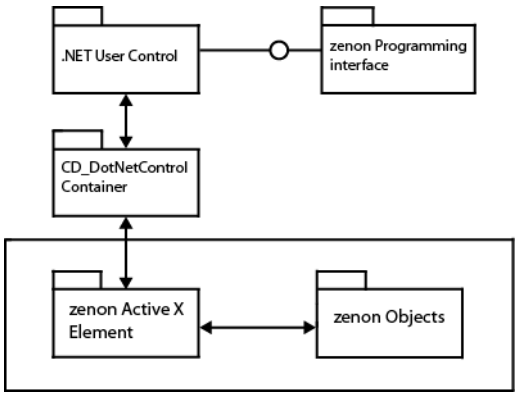
5.2 Example .NET control container

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 45)
- add a CD_DotNetControlContainer and a .NET User Control (on page 53)
- enable the access to the user control via VSTA (VBA) (on page 58)

PATH FOR DLL IN EDITOR AND RUNTIME

The path to **.Net DLL** that is selected in the Editor is also used in the 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.

5.2.1.1 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

5.2.1.2 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.

5.2.1.3 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.

5.2.1.4 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*

Parameters	Value	Description
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*

5.2.1.5 public MaxVariables()

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

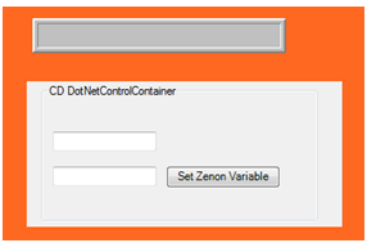
1: Multi-select is disabled in the variable list. A warning is displayed when several variables are selected anyway.

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.

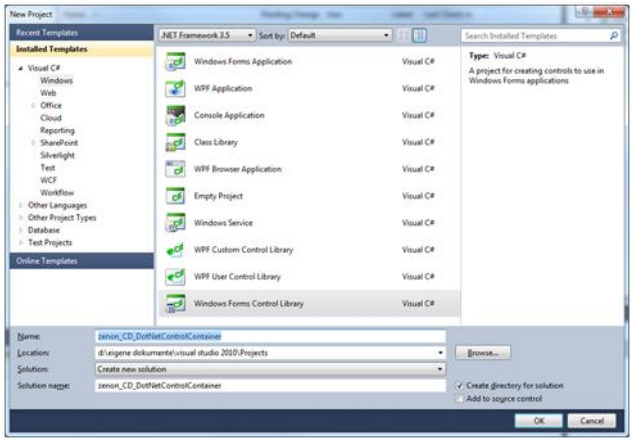
> **Information**
>
> The screenshots for this theme are only available in English.



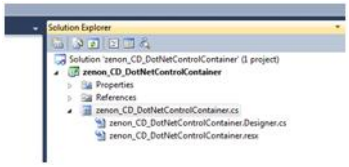
WORK STEPS

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

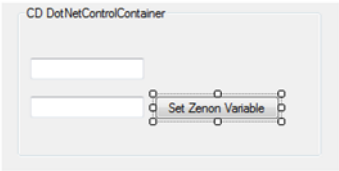
Important: Set framework to 3.5!



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

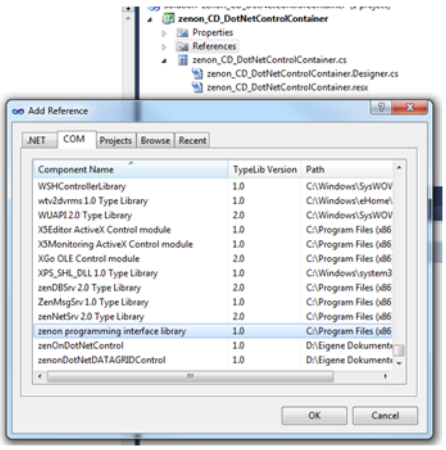


3. In the next step you create the user control. For this use two text boxes one each for the input and the output and a button for writing new values to the zenon variable.
   Name:
   - the first text box "**txtGetZenonVariable**"
   - the second text box "**txtSetZenonVariable**"
   - the button "**btnSetZenonVariable**"

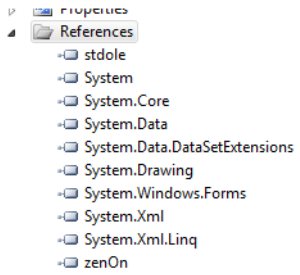


4. In order to access zenon objects you need a reference to the <CD_PRODUCNAME> Programming Interface. To do this:
   - click on node "**References**" in the Solution Explorer
   - open the context menu
   - select **Add References...**
   - switch to tab **COM**

- 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**:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using zenOn;

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

        public zenon_CD_DotNetControlContainer()
        {
            InitializeComponent();
        }


    }
}

```

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

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

and enabled via public method **zenOnExit**:

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

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

8. 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.

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

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)
```

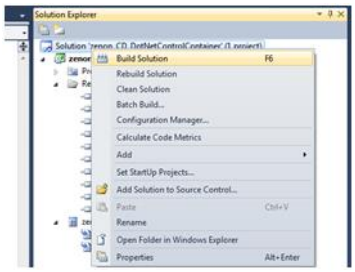
```
System.Runtime.InteropServices.Marshal.ReleaseComObject(m_cVal);
m_cVal = null;
}
}
catch { }
return true;
}
  /// <summary>
  /// This public Method is needed to link zenon Variables
  /// to the control.
  /// </summary>
  /// <returns></returns>
public short CanUseVariables()
{
  return 1; // Only this Variable is supported
}
/// <summary>
/// This public Method returns the Type of
/// supported zenon Variables
/// </summary>
/// <returns></returns>
public short VariableTypes()
{
return short.MaxValue; // all Data Types supported
}
/// <summary>
/// This public Method returns the number of
/// supported zenon Variables
```

```
/// </summary>
/// <returns></returns>
public short MaxVariables()
{
return 1; // Only 1 Variable should linked to the Control
}
/// <summary>
/// This will be 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 name="e">
private void zenon_CD_DotNetControlContainer_Paint(object sender, PaintEventArgs e)
{
if (m_cVal != null)
```

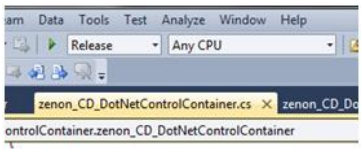
```
{
this.txtGetZenonVariable.Text = m_cVal.get_Value(0).ToString();
return;
}
else
{
this.txtGetZenonVariable.Text = "Variable Value";
return;
}
}
}
}
```

CREATE RELEASE

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



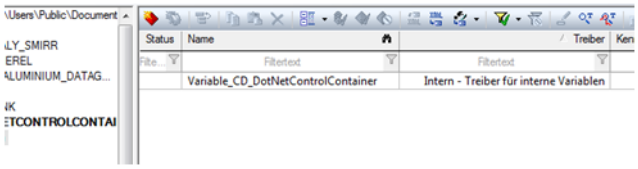
For this it is necessary that you switch from **Debug** to **Release** in the settings.



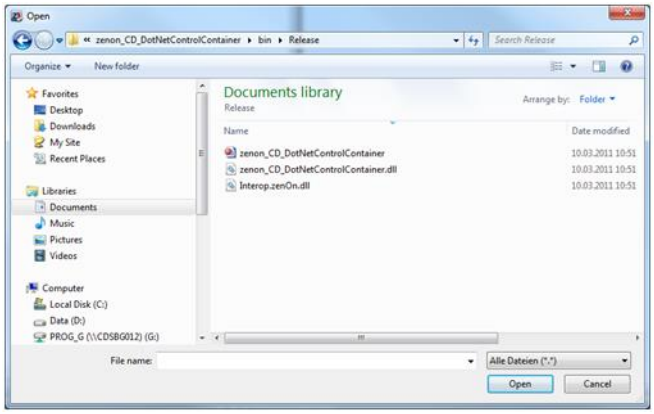
5.2.3 add a CD_DotNetControlContainer and a .NET User Control

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

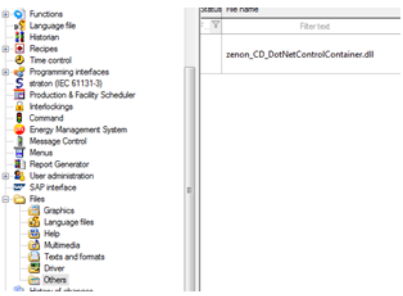
1. Create an internal variable of type *String* and set the string length to *30*.



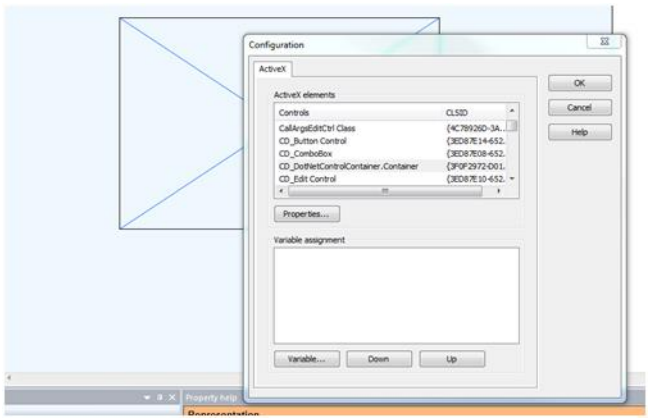
2. In the zenon project node *Project/Files/Others* add the DLL of the created .NET user controls. Here, the file is copied to the **Additional** file at file system level.



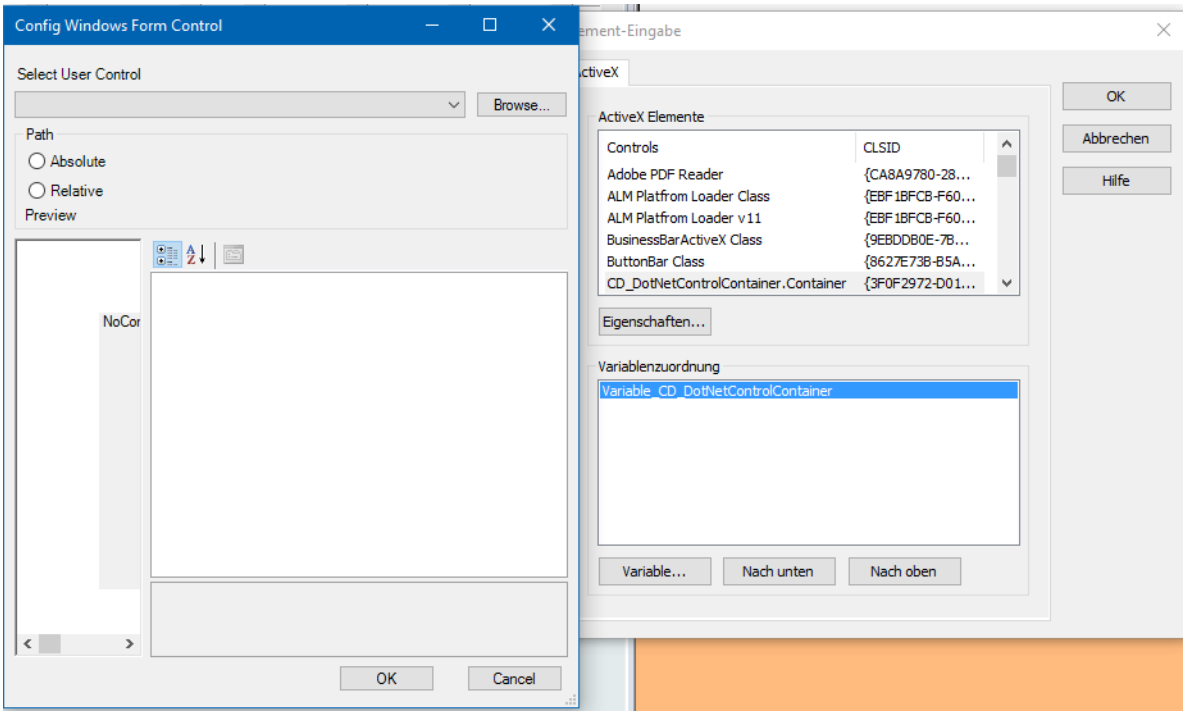
3. The DLL is located in the Visual Studio Project output folder under *...\bin\Release\zenon_CD_DotNetControlContainer.dll*.



4. 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.

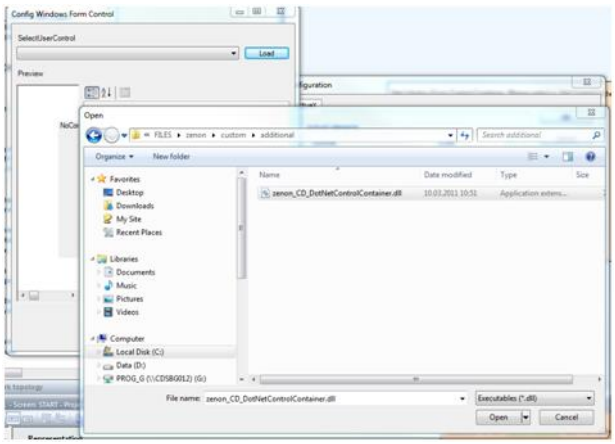


5. To embed the .NET user control in the **CD_DotNetControlContainer** control:
   - Click on button **Properties**
   - A new dialog is opened



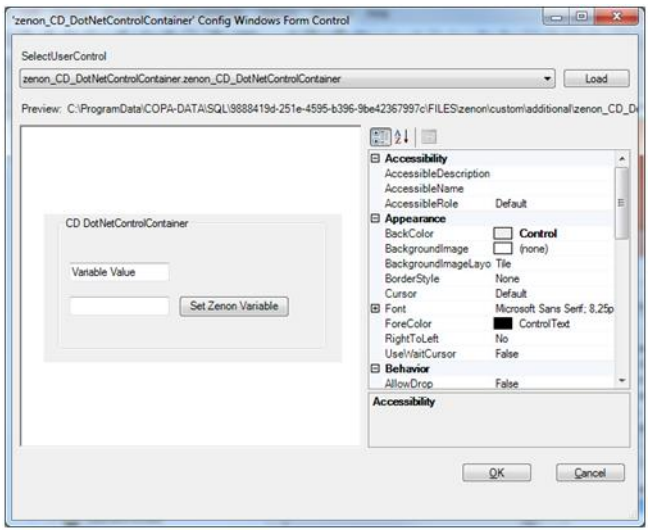
   - Click on button **Load** in order to select the path of the project folder, for example: *C:\ProgramData\COPA-DATA\SQL\9888419d-251e-4595-b396-9be42367997c\FILES\zenon\custom\additional\zenon_CD_DotNetControlContainer.dll*

Note: Controls should always be saved in the project's **Additional** folder. They are thus taken into account during backups and transfers. In doing so, the path is to be defined as relatively close to the save location. Alternatively, you can, for **Path**, use the *Absolute* option; however, in doing so, it must be ensured that there is the same directory structure on the target system.

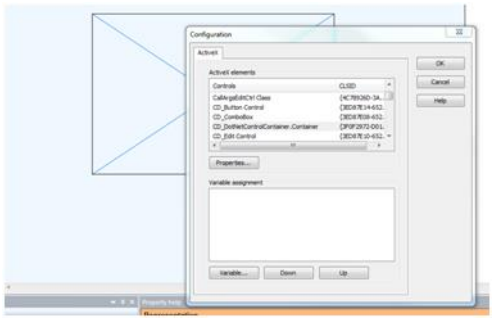


Now the .NET user control should be displayed.

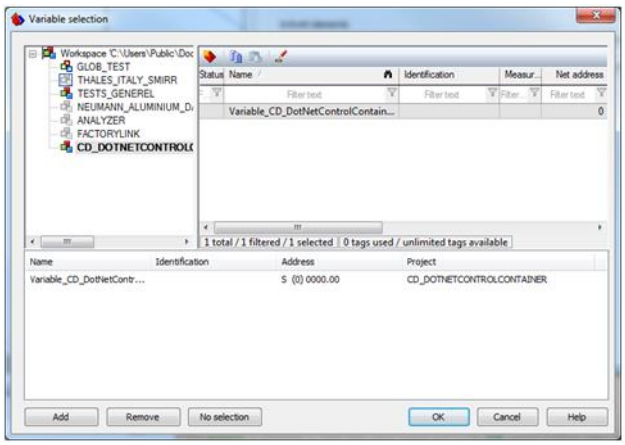
Confirm the dialog by clicking on **OK**.



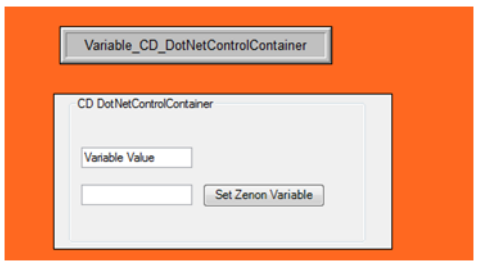
6. In the last step link a variable with the control via button **Variables**.



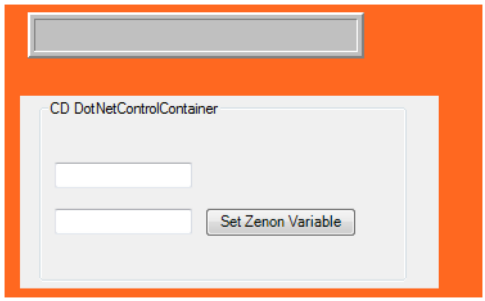
The variable selected first is automatically linked with our globally defined variable (.NET UserControl) via public method `zenonInit`. The linking with the control is carried out after the Runtime start.



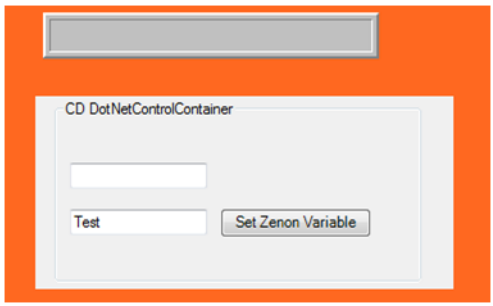
Then link the internal variable with a text element.



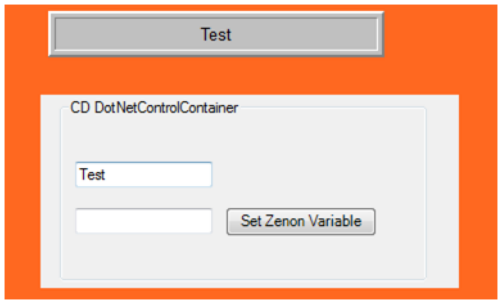
7. After the Runtime start the control is initially empty.



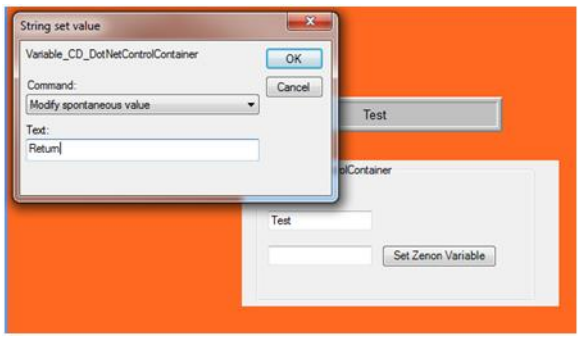
If you enter a value in the second text box and then confirm it with button **Set zenon variable**, the value is written to the zenon variable. (The **btnSetZenonVariable_Click** event is carried out.)



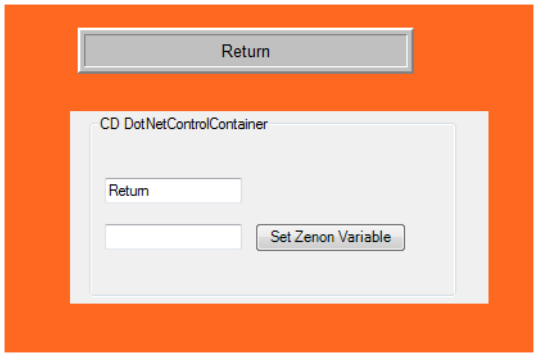
This is also displayed in the zenon text element.



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



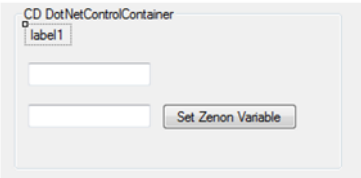
the value is directly written in the first text box via the **Paint** event of the .NET control.



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 l**blzenonInfo**. In this label the value of another zenon variable should be displayed. The new value should be set via a VSTA macro.



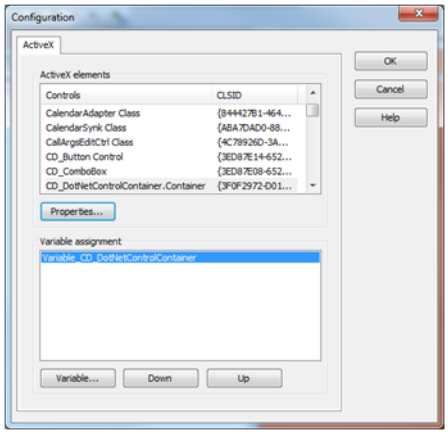
2. Enhance the code by a property (**Information**) and add the properties **get** and **set** to the property. They allow you to read and write the text of the label.

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

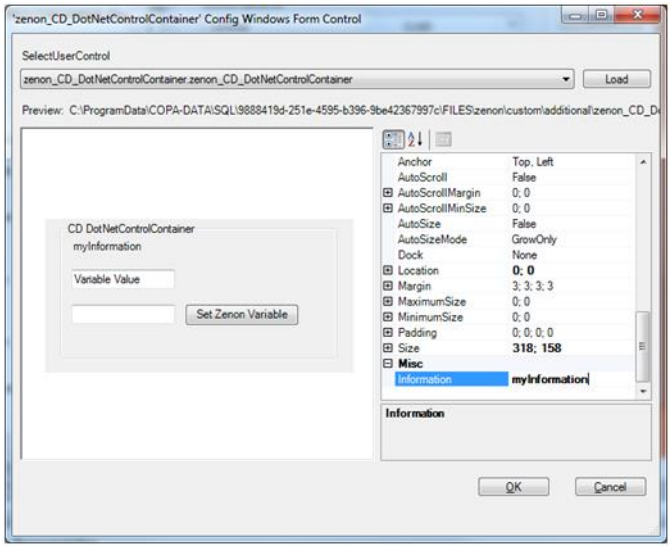
3. Create a new release for our user control and copy it to folder *additional* of the zenon project. **Do not forget:** zenon Editor must be closed before you do this!

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.

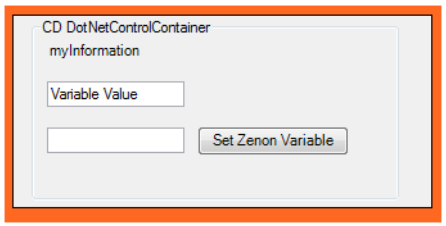
4. In the zenon Editor click on the ActiveX and open the property window.



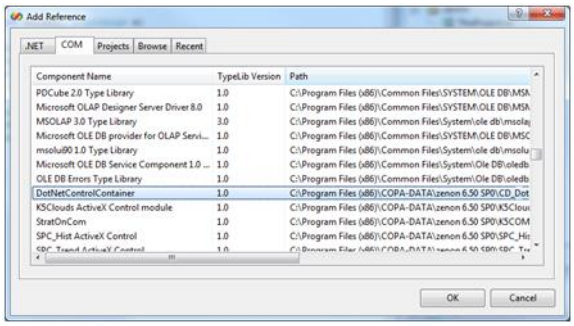
Now you can see the new property **Information** in the selection window of the control and you can also set a value.



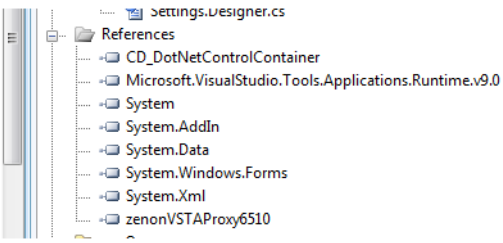
This value is also set in the control ("myInformation")



5. In order to 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 AktiveX Element
        // with a cast
        CD_DotNetControlContainer.Container zAktiveX = (CD_DotNetControlContainer.Container)zElement.AktiveX();

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

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

7. 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 30)
2. Change .NET User Control to dual control (on page 33)
3. Work via VBA with ActiveX in the Editor (on page 37)
4. <Connect CD_PRODUCTNAME> variables with the .NET user control (on page 38)

Attention

When using zenon COM objects with self-created user controls or external applications, they must be enabled using the **Marshal.ReleaseComObject** method. Enabling by means of the **Marshal.FinalReleaseComObject** method must not be used, because this leads to a malfunction of zenon Add-ins.

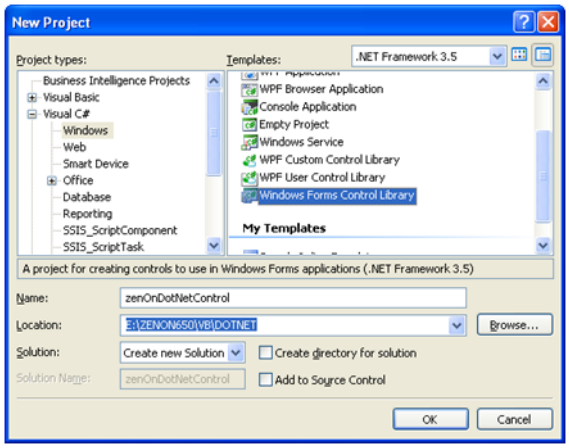
Information

The screenshots for this theme are only available in English.

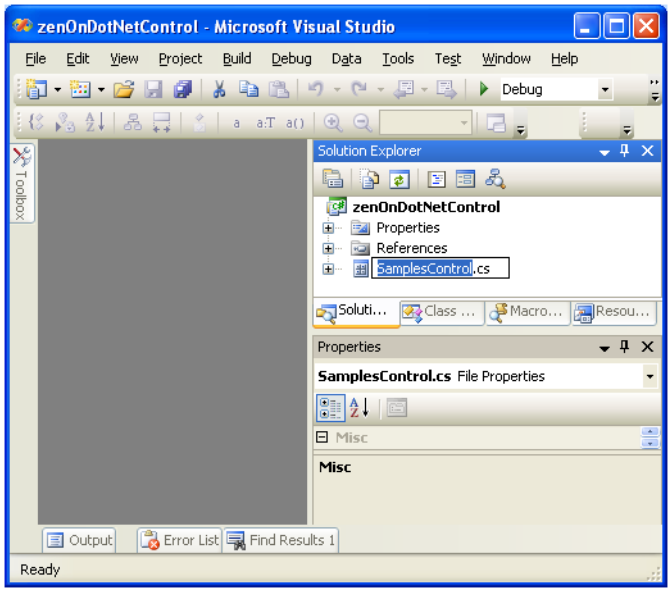
5.3.1 Creat Windows Form Control

To create a Windows Form Control:

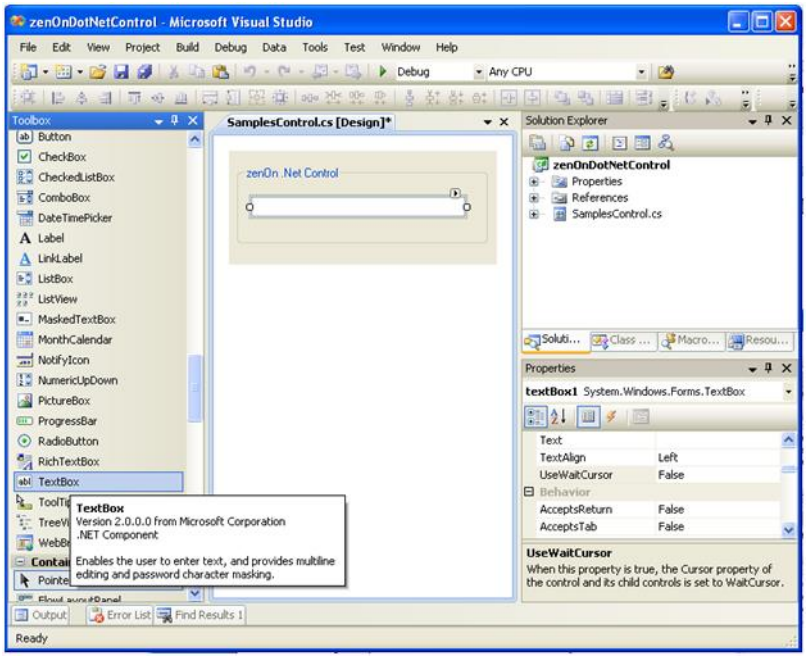
1. Start Visual Studio 2008 and create a new Windows **Form Control Library** project:



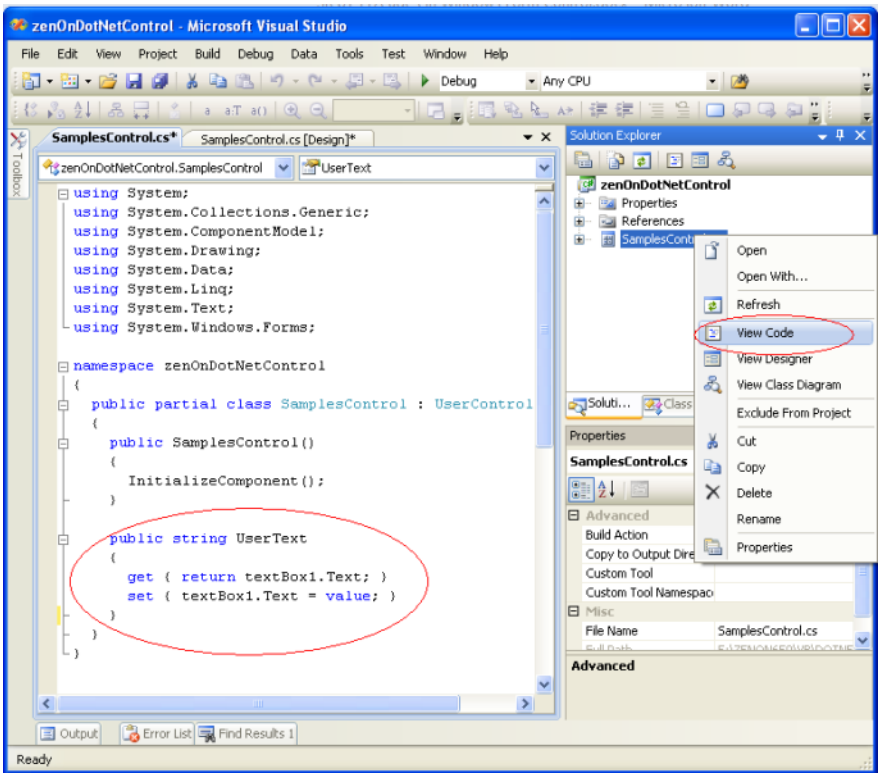
2. Rename the default control to the desired control name.
   In our example: **SampesControl.cs**.



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



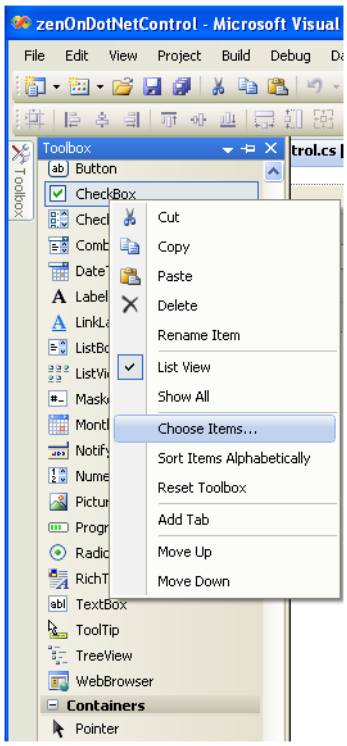
4. Normally controls have properties. Open the Code Designer via **View Code** and ass the desired properties which should be available externally.
   In our example: Externally visible property „**UserText**" with **get** and **set** access which contains the text of the text box:



5. Compile the project.

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

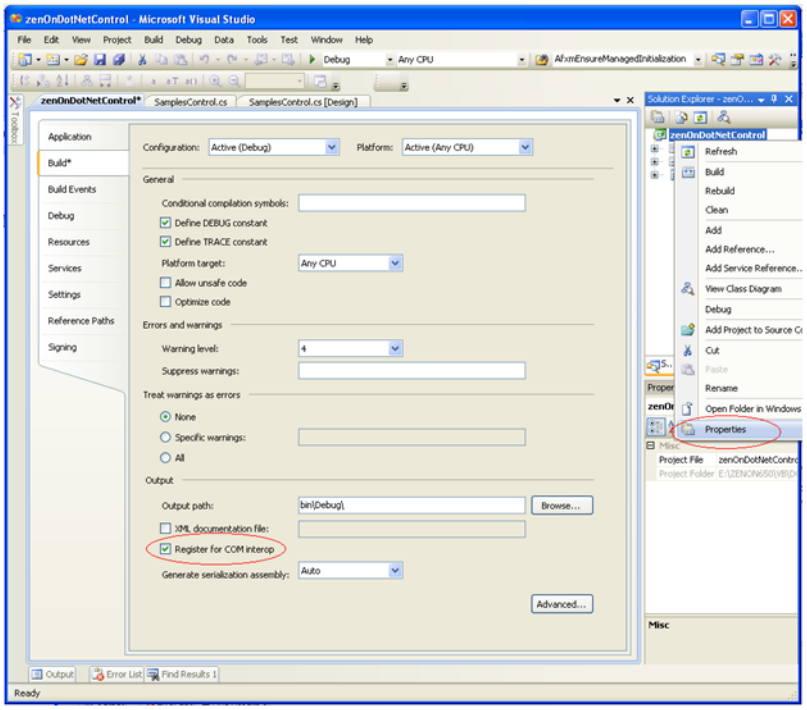
Important: The control must be inserted manually in the control tool box via **Choose Items**.



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.

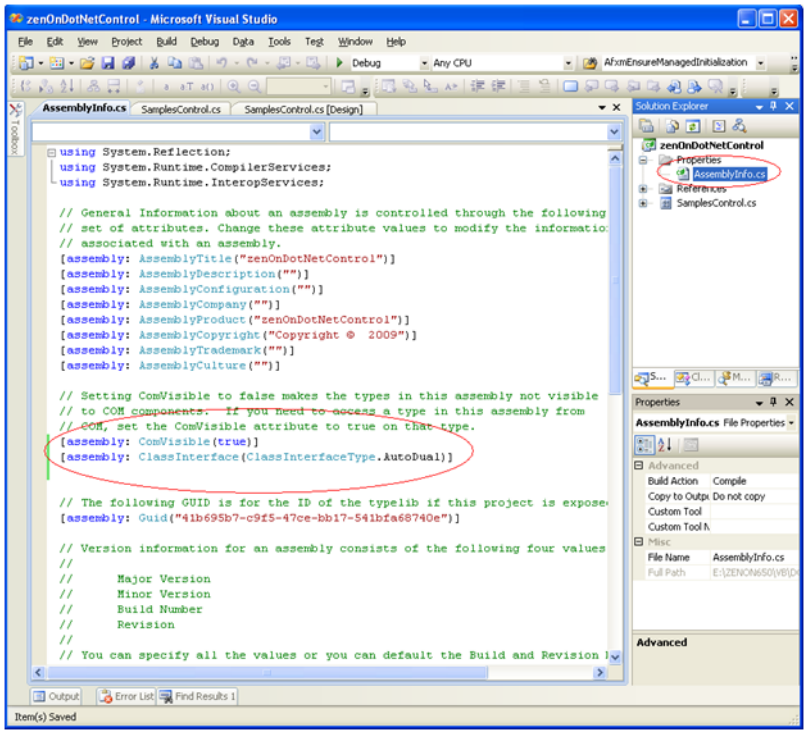
1. Open the project and activate property **Register for COM interop** in the **Build** settings:



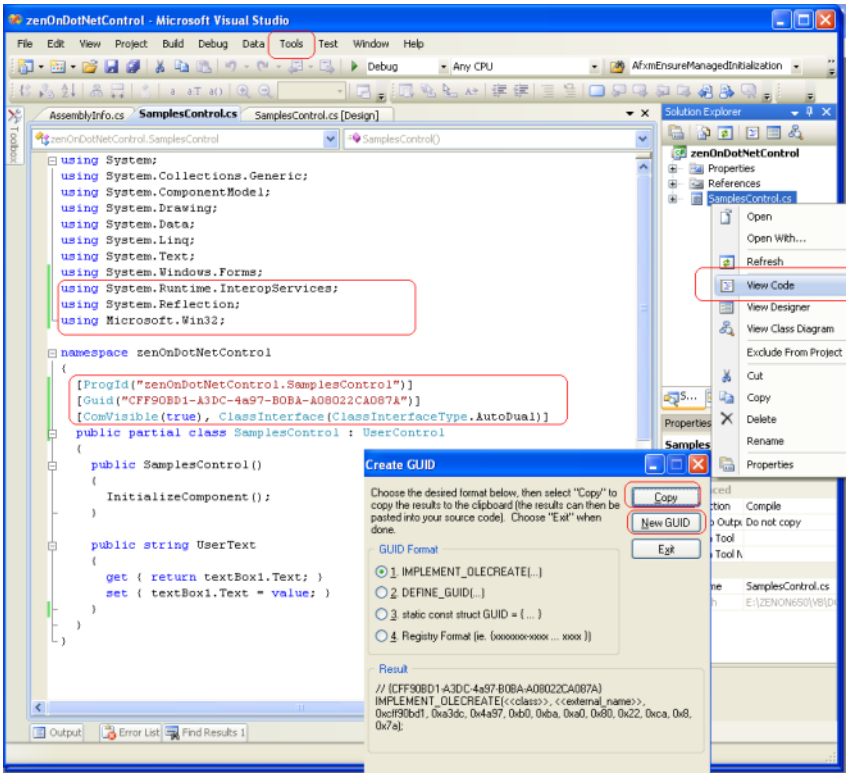
2. Open the file **AssemblyInfo.cs** and
   - set attribute **ComVisible** to *true*
   - add attribute **ClassInterface**

   [assembly: ComVisible(true)]

[assembly: ClassInterface(ClassInterfaceType.AutoDual)]

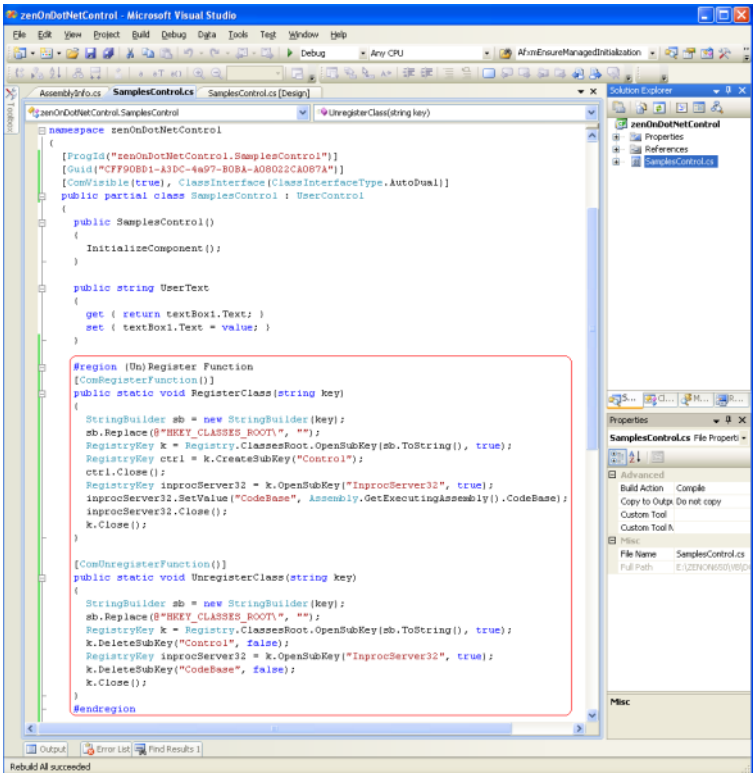


3. Open the code designer via **View Code** and add the necessary ActiveX attributes and **using** entries. Via menu **Tools/Create GUID** create a new GUID for the GUID attribute:



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.

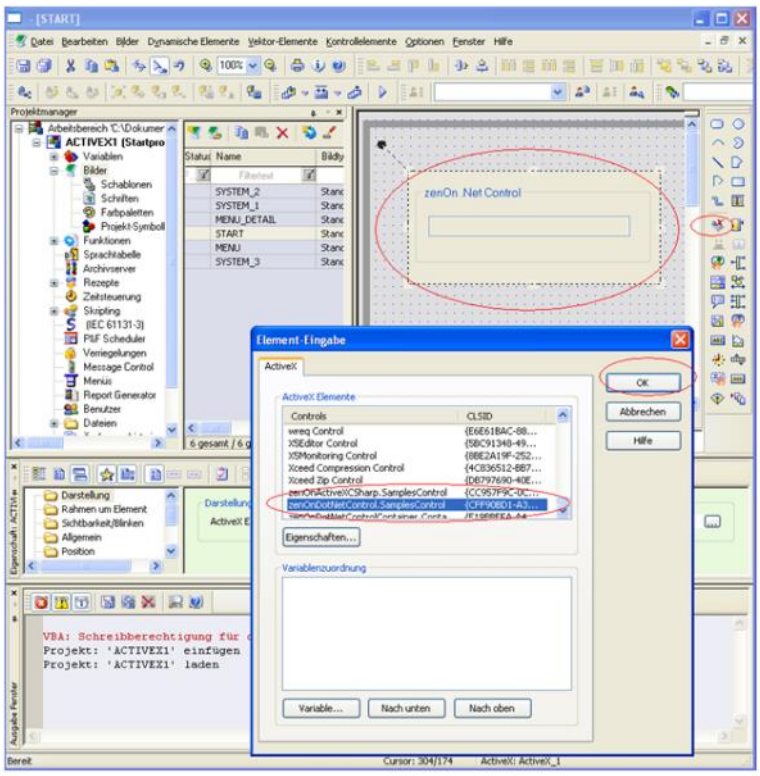
   The Windows Form Control is now ActiveX-able and was registered automatically during the rebuild. An additional typelib file **zenOnDotNetControl.tlb** was created in the output directory.

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:



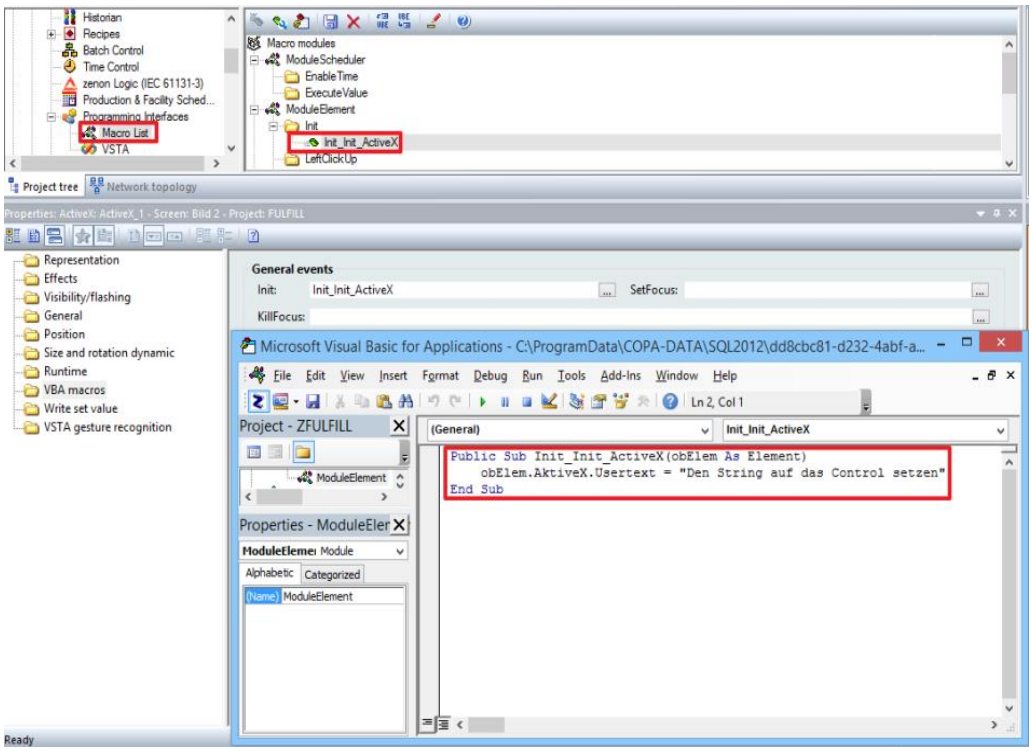
5.3.3 Work via VBA with ActiveX in the Editor

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.AktiveX.Usertext = "Set the string to the control"
End Sub
```

5.3.4 <Connect CD_PRODUCTNAME> 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.

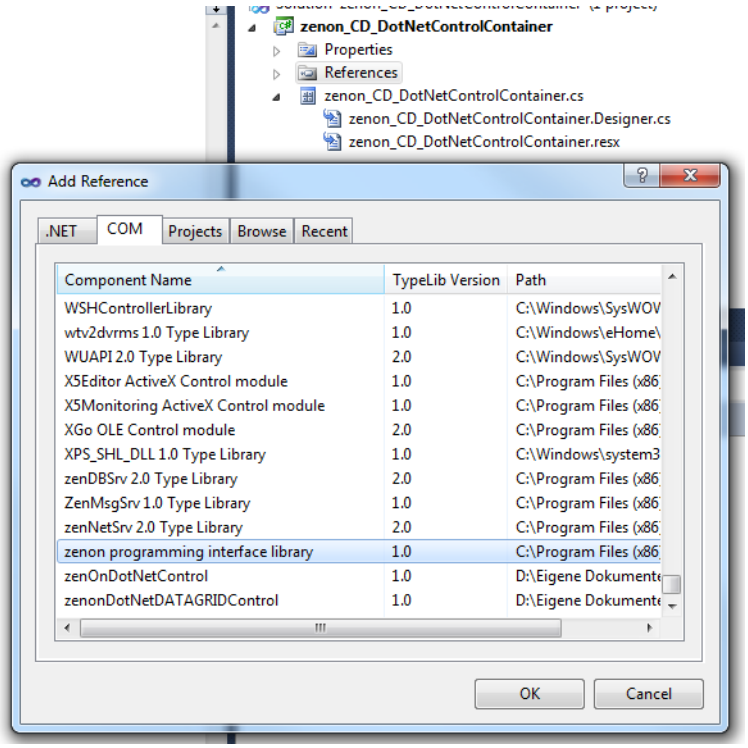
NECESSARY METHODS

- public bool zenOnInit (on page 40) (Is called up during control initializing in the zenon Runtime.)
- public bool zenOnInitED (on page 40) (Is used in the Editor.)
- public bool zenOnExit() (on page 41) (Is called up during control destruction in the zenon Runtime.)
- public bool zenOnExitED() (on page 41) (Is used in the Editor.)
- public short CanUseVariables() (on page 41) (Supports linking variables.)

- public short VariableTypes() (on page 41) (Supported data types by the control)
- public MaxVariables() (on page 42)(Maximum number of variables which can be linked to the control.)

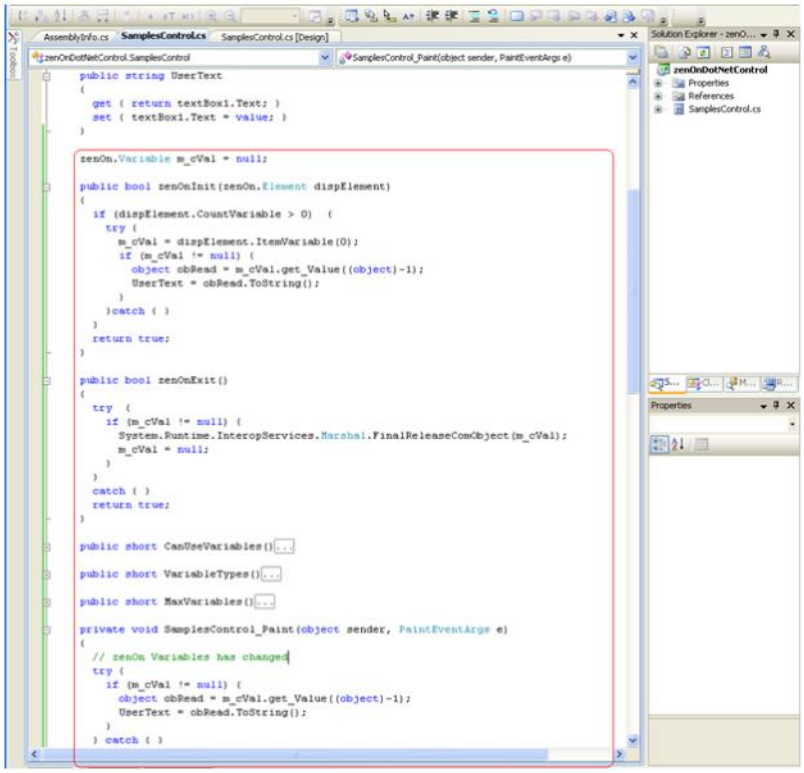
ADD REFERENCE

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.



5.3.4.1 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

5.3.4.2 public bool zenOnInitED(zenOn.Element dispElement)

Equals public bool zenOnInit (on page 40) and is executed when opening the ActiveX in the Editor (double click on ActiveX).

5.3.4.3 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.

5.3.4.4 public bool zenOnExitED()

Equals public bool zenOnExit() (on page 41) and is executed in closing the ActiveX in the Editor. With this you can react to changes, e.g. value changes, in the Editor.

5.3.4.5 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.

5.3.4.6 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*

5.3.4.7 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

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

You can find extensive documentation about WPF elements in zenon and tutorials for designers, developers and people configuring projects in the zenon WPF element manual.

This manual looks at the following topics:

- Basics
- Guidelines for designers
- Guidelines for developers
- Engineering in zenon