zenon
by COPA-DATA

Manuel de zenon

Contrôles

v.8.10

Contenu

1 Bienvenue dans l'aide de COPA-DATA

TUTORIELS VIDÉO DE ZENON.

Des exemples concrets de configurations de projets dans zenon sont disponibles sur notre chaîne YouTube (https://www.copadata.com/tutorial_menu). Les tutoriels sont regroupés par sujet et proposent un aperçu de l'utilisation des différents modules de zenon. Les tutoriels sont disponibles en anglais.

AIDE GÉNÉRALE

Si vous ne trouvez pas certaines informations dans ce chapitre de l'aide ou si vous souhaitez nous suggérer d'intégrer un complément d'information, veuillez nous contacter par e-mail : documentation@copadata.com.

ASSISTANCE PROJET

Si vous avez besoin d'aide dans le cadre d'un projet, n'hésitez pas à adresser un e-mail à notre service d'assistance : support@copadata.com

LICENCES ET MODULES

Si vous vous rendez compte que vous avez besoin de licences ou de modules supplémentaires, veuillezcontacter l'équipe commerciale par e-mail : E-mail sales@copadata.com.

2 Contrôles

Dans zenon, vous pouvez intégrer vos propres contrôles. Pour cela, les possibilités suivantes sont disponibles :

- Contrôles utilisateur .NET (à la page 43) (pour la mise en œuvre dans zenon, voir également la section Contrôles .NET dans le manuel Synoptiques.)
- ActiveX (à la page 10) (pour la mise en œuvre dans zenon, voir également la section ActiveX dans le manuel Synoptiques.)
- WPF

> **Information**
>
> Des informations concernant l'utilisation des interfaces de programmation (PCE, VBA, VSTA) de zenon sont disponibles dans le manuel Interfaces de programmation.

> **Attention**
>
> Les erreurs dans les applications telles que ActiveX, PCE, VBA, VSTA, WPF et les applications externes accédant à zenon via l'API peuvent également influencer la stabilité du Runtime.

3 Général

Les contrôles de zenon peuvent être mis en œuvre via ActiveX, .NET et WPF. Via VBA/VSTA, vous pouvez accéder à l'API de zenon.

3.1 Access zenon API

Dans zenon, vous pouvez améliorer un contrôle ActiveX avec des fonctions spéciales, afin d'accéder à l'API zenon.

ACCÉDEZ À L'API ZENON.

- Sélectionner, dans **Références du projet**, via **Ajouter des références...**, **<Bibliothèque d'objets de Runtime> CD_PRODUCTNAME**
- Ajoutez les fonctions améliorées au code de classe du contrôle.

FONCTIONS ACTIVEX AMÉLIORÉES DE ZENON

```
// Appelé durant l'initialisation du contrôle dans le Runtime zenon.
public bool zenon>Init(zenon.Element dispElement)…
// Appelé durant la destruction du contrôle dans le Runtime zenon.
public bool zenonExit()
// Prend en charge la liaison de variable du contrôle
public short CanUseVariables()…
// Le contrôle COM prend en charge différents types de données.
public short VariableTypes()…
```

```
// Nombre maximal de variables pouvant être liées au contrôle.
```

public short MaxVariables()...

EXEMPLE

L'objet COM d'une variable de zenon est temporairement enregistré dans un membre, afin qu'il soit accessible ultérieurement via l'événement Paint du contrôle.

```
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; // les variables sont prises en charge
}

public short VariableTypes()
{
```

```
return short.MaxValue; // Tous les types de données sont pris en charge
}

public short MaxVariables()
{
return 1; // Au maximum, une variable sera liée au contrôle
}

private void SamplesControl_Paint(object sender, PaintEventArgs e)
{
// Les variables de zenon ont été modifiées
try {
if (m_cVal != null) {
object obRead = m_cVal.get_Value((object)-1);
UserText = obRead.ToString();
}
}catch { }
}
```

3.2 Méthodes

Les contrôles ActiveX et .NET utilisant des variables de zenon nécessitent certaines méthodes spécifiques.

3.2.1 CanUseVariables

Prototype : short CanUseVariables();

Cette méthode renvoie *1* ou *0*

Value	Description
1:	Le contrôle peut utiliser des variables de zenon. Pour l'élément dynamique (via le bouton **Variable**), vous pouvez uniquement déclarer des variables de zenon avec le type déclaré par le biais de la méthode VariableTypes (à la page 9), conformément au nombre défini par la méthode MaxVariables (à la page 9).
0:	Le contrôle ne peut pas utiliser de variables de zenon, ou ne dispose pas de la méthode adéquate. Vous pouvez déclarer des variables de tout type, sans limitation de nombre. Dans le

Valu e	Description
	Runtime, toutefois, ces variables sont uniquement utilisables avec du code VBA.

3.2.2 MaxVariables

Prototype: short MaxVariables();

Dans cette section est défini le nombre de variables pouvant être sélectionnées dans la liste de variables.

Si la valeur *1* est renvoyée, la sélection multiple est désactivée dans la liste de variables. Un avertissement est alors affiché si plusieurs variables sont sélectionnées.

3.2.3 Types de variables

Prototype: short VariableTypes();

La valeur renvoyée par cette méthode est utilisée comme masque pour les types de variable utilisables dans la liste de variables. La valeur est une relation **AND** (et) parmi les valeurs suivantes (définies dans *zenon32/dy_type.h*) :

Valeur 1	Valeur 2	Équivalent
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();

Cette méthode est appelée par le Runtime de zenon lorsque le contrôle ActiveX est fermé.

Ici, tous les pointeurs de répartition renvoyant à des variables doivent être libérés.

3.2.5 zenonExitEd

Équivalent à zenonExit (à la page 9) ; exécuté lors de la fermeture d'ActiveX dans Editor.

Cette méthode permet de réagir aux changements dans ActiveX, par exemple aux modifications de valeurs dans Editor.

Information : actuellement uniquement disponible pour ActiveX.

3.2.6 zenonInit

Prototype: boolean zenonInit(IDispatch*dispElement);

Cette méthode (dans le Runtime) permet au contrôle ActiveX d'obtenir un pointeur vers l'interface de répartition de l'élément dynamique. Ce pointeur permet également d'accéder aux variables de zenon liées à l'élément dynamique.

Vous définissez l'ordre de tri des variables transférées dans la configuration de l'élément ActiveX à l'aide des boutons **Bas** ou **Haut**.
La boîte de dialogue **Entrer les propriétés** apparaît lorsque vous double-cliquez sur l'élément ActiveX ou que vous sélectionnez la propriété **Paramètres ActiveX** dans les propriétés de l'élément, dans le nœud **Affichage**.

3.2.7 zenonInitEd

Équivalent à zenonInit (à la page 10) ; exécuté lors de l'ouverture d'ActiveX dans Editor (double-cliquez sur le contrôle ActiveX).

Information : actuellement uniquement disponible pour ActiveX.

4 ActiveX

ActiveX permet d'améliorer de manière autonome les fonctionnalités du Runtime et d'Editor de zenon.

Dans ce manuel, vous trouverez des informations concernant :

- Développement d'éléments ActiveX (à la page 11)
- Exemple : LatchedSwitch (C++) (à la page 14)
- Exemple : CD SliderCtrl (C++) (à la page 23)
- Exemple : contrôle. NET exécuté en tant que contrôle ActiveX (C#) (à la page 30)

Vous trouverez d'autres informations concernant les éléments dynamiques ActiveX dans le manuel Synoptiques, au chapitre ActiveX.

ACTIVEX POUR WINDOWS CE

Si un contrôle ActiveX doit être exécuté sous Windows CE, le modèle de **apartment** doit être défini sur *Threading*. S'il est défini sur *Free* (Libre), le contrôle ne s'exécutera pas dans le Runtime de zenon.

4.1 Développement d'éléments ActiveX

L'élément dynamique ActiveX dans zenon peut transmettre des variables au contrôle ActiveX sans utiliser VBA pour actionner le contrôle.

Le contrôle définit maintenant automatiquement combien de variables zenon il peut utiliser et de quel type ces variables peuvent être. En outre, les propriétés du contrôle peuvent également être définies par l'élément dynamique.

Pour cela, l'interface (interface de transmission) du contrôle doit être compatible avec un certain nombre de méthodes (à la page 11) .

4.1.1 Méthodes

Chaque contrôle ActiveX pouvant utiliser des variables de zenon doit contenir les méthodes suivantes :

- CanUseVariables (à la page 8)
- MaxVariables (à la page 9)
- Types de variables (à la page 9)
- zenonExit (à la page 9)
- zenonExitEd (à la page 10)
- zenonInit (à la page 10)
- zenonInitEd (à la page 10)

L'ID de répartition des méthodes dans l'interface est sans importance. Lors de l'appel des méthodes, zenon reçoit l'ID de l'interface.

4.1.1.1 CanUseVariables

Prototype : short CanUseVariables();

Cette méthode renvoie *1* ou *0*

Valu e	Description
1:	Le contrôle peut utiliser des variables de zenon. Pour l'élément dynamique (via le bouton **Variable**), vous pouvez uniquement déclarer des variables de zenon avec le type déclaré par le biais de la méthode VariableTypes (à la page 9), conformément au nombre défini par la méthode MaxVariables (à la page 9).
0:	Le contrôle ne peut pas utiliser de variables de zenon, ou ne dispose pas de la méthode adéquate. Vous pouvez déclarer des variables de tout type, sans limitation de nombre. Dans le Runtime, toutefois, ces variables sont uniquement utilisables avec du code VBA.

4.1.1.2 MaxVariables

Prototype: short MaxVariables();

Dans cette section est défini le nombre de variables pouvant être sélectionnées dans la liste de variables.

Si la valeur *1* est renvoyée, la sélection multiple est désactivée dans la liste de variables. Un avertissement est alors affiché si plusieurs variables sont sélectionnées.

4.1.1.3 Types de variables

Prototype: short VariableTypes();

La valeur renvoyée par cette méthode est utilisée comme masque pour les types de variable utilisables dans la liste de variables. La valeur est une relation **AND** (et) parmi les valeurs suivantes (définies dans *zenon32/dy_type.h*) :

Valeur 1	Valeur 2	Équivalent
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

Valeur 1	Valeur 2	Équivalent
IN_OUTPUT	0x8000	Position 15

4.1.1.4 zenonExit

Prototype: boolean zenonExit();

Cette méthode est appelée par le Runtime de zenon lorsque le contrôle ActiveX est fermé.

Ici, tous les pointeurs de répartition renvoyant à des variables doivent être libérés.

4.1.1.5 zenonExitEd

Équivalent à zenonExit (à la page 9) ; exécuté lors de la fermeture d'ActiveX dans Editor.

Cette méthode permet de réagir aux changements dans ActiveX, par exemple aux modifications de valeurs dans Editor.

Information : actuellement uniquement disponible pour ActiveX.

4.1.1.6 zenonInit

Prototype: boolean zenonInit(IDispatch*dispElement);

Cette méthode (dans le Runtime) permet au contrôle ActiveX d'obtenir un pointeur vers l'interface de répartition de l'élément dynamique. Ce pointeur permet également d'accéder aux variables de zenon liées à l'élément dynamique.

Vous définissez l'ordre de tri des variables transférées dans la configuration de l'élément ActiveX à l'aide des boutons **Bas** ou **Haut**.
La boîte de dialogue **Entrer les propriétés** apparaît lorsque vous double-cliquez sur l'élément ActiveX ou que vous sélectionnez la propriété **Paramètres ActiveX** dans les propriétés de l'élément, dans le nœud **Affichage**.

4.1.1.7 zenonInitEd

Équivalent à zenonInit (à la page 10) ; exécuté lors de l'ouverture d'ActiveX dans Editor (double-cliquez sur le contrôle ActiveX).

Information : actuellement uniquement disponible pour 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

Le contrôle LatchedSwitch comporte l'interface de répartition suivante :

```
[ uuid(EB207159-D7C9-11D3-B019-080009FBEAA2),
helpstring(Dispatch interface for LatchedSwitch Control), hidden ]
dispinterface _DLatchedSwitch

{
        properties:
        // NOTE - ClassWizard conserve les informations de méthodes ici.
        // Soyez très prudent si vous modifiez cette section !
        //{{AFX_ODL_PROP(CLatchedSwitchCtrl)
        [id(1)] boolean SollwertDirekt;
        [id(2)] IPictureDisp* SwitchOn; // conteneur pour bitmaps
        [id(3)] IPictureDisp* SwitchOff;
        [id(4)] IPictureDisp* LatchedOn;
        [id(5)] IPictureDisp* LatchedOff;
        //}}AFX_ODL_PROP


methods:
// NOTE - ClassWizard conserve les informations de méthodes ici.
 // Soyez très prudent si vous modifiez cette 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();

};
```

Les propriétés **SwitchOn** à **LatchedOff** contiennent les fichiers bitmap correspondant aux quatre différents états du contrôle. Les fichiers bitmap sont conservés dans des objets de la classe CScreenHolder. La propriété **SollwertDirekt** définit si la saisie de valeurs prescrites doit être effectuée dans une boîte de dialogue, ou directement en cliquant sur le contrôle.

4.2.2 Contrôle

La mise en œuvre du contrôle s'effectue par le biais de la classe **CLatchedSwitchCtrl**. Les membres de cette classe sont les objets**CScreenHolder** destinés au stockage de bitmaps. En outre, trois drivers de répartition sont générés pour l'élément dynamique et les variables :

```
class CLatchedSwitchCtrl : public COleControl
{

DECLARE_DYNCREATE(CLatchedSwitchCtrl)

// Constructeur
public:

CLatchedSwitchCtrl();

// Substitutions

// Substitutions de fonctions virtuelles générées par ClassWizard
//{{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

// Mise en œuvre
protected:

~CLatchedSwitchCtrl();

DECLARE_OLECREATE_EX(CLatchedSwitchCtrl) // Fabrique de classe et guid
DECLARE_OLETYPELIB(CLatchedSwitchCtrl) // GetTypeInfo
DECLARE_PROPPAGEIDS (CLatchedSwitchCtrl) // ID de page de propriété
DECLARE_OLECTLTYPE (CLatchedSwitchCtrl) // Nom de type et informations d'état diverses

// Tables de messages

//{{AFX_MSG(CLatchedSwitchCtrl)
afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
```

```
//}}AFX_MSG
DECLARE_MESSAGE_MAP()

// Tables de répartition

//{{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);

// ID de répartition et d'événement
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 Méthodes

Les méthodes suivantes sont utilisées :

- CanUseVariables (à la page 18)
- Types de variables (à la page 18)
- MaxVariables (à la page 18)
- zenonInit (à la page 18)
- zenonExit (à la page 20)

4.2.3.1 CanUseVariables

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

```
short CLatchedSwitchCtrl::CanUseVariables()
{

return 1;
}
```

4.2.3.2 Types de variables

Le contrôle est uniquement utilisable avec les variables de bit, donc le résultat *0x0004* est renvoyé.

```
short CLatchedSwitchCtrl::VariableTypes()
{

return 0x0004; // Variables de bit uniquement
}
```

4.2.3.3 MaxVariables

Deux variables peuvent être utilisées. La valeur *2* est donc renvoyée.

```
short CLatchedSwitchCtrl::MaxVariables()
{

return 2; // 2 variables
}
```

4.2.3.4 zenonInit

Cette méthode obtient les *Dispatchdriver* des variables via le *Dispatchpointer* de l'élément dynamique. Avec ce pointeur (*Pointer*), les valeurs des variables sont lues et écrites lorsque l'utilisateur clique et lors du traçage du contrôle.

```
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();

Les objets inutilisés sont automatiquement supprimés de la mémoire. Cette opération doit être exécutée par la programmation. Le programmeur détermine (sur la base d'un compteur de références) si un objet peut être supprimé.

COM utilise les méthodes *IUnknow AddRef* et *Release* pour gérer le nombre de références d'interfaces avec un objet.

Les règles générales d'appel de ces méthodes sont les suivantes :

- *AddRef* doit toujours être appelé sur l'interface si le client reçoit un pointeur d'interface.
- Une instruction *Release* doit toujours être appelée sur l'interface si le client met un terme à l'utilisation du pointeur d'interface.

Dans le cadre d'un déploiement simple, un compteur de variables dans l'objet est augmenté avec l'appel d'une instruction *AddRef* . Chaque appel d'une instruction *Release* réduit ce compteur dans l'objet. Si ce compteur atteint à nouveau ZÉRO, l'interface peut être supprimée de la mémoire.

Un compteur de références peut également être mis en œuvre, afin que chaque référence à l'objet (et non à une interface individuelle) soit comptée.

Dans ce cas, chaque substitut d'*AddRef* et de *Release* appellent une mis en œuvre centrale par rapport à l'objet. Une instruction *Release* déverrouille alors l'ensemble de l'objet si le compteur de références a atteint zro.

4.2.3.5 zenonExit

Cette méthode libère le driver de répartition.

```
BOOL CLatchedSwitchCtrl::zenonExit()
{

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

4.2.4 Utilisation et affichage

4.2.4.1 Spécifier valeur prescrite

Une valeur peut être définie en cliquant sur le contrôle avec le bouton gauche de la souris.

Si **m_iSollwertDirekt** est égal à *0*, une boîte de dialogue de sélection de la valeur prescrite s'affiche ; dans le cas contraire, la valeur actuelle de la variable de commutation est inversée.

Si la variable de verrouillage possède la valeur *1*, seule la fonction *MessageBeep* est exécutée. Aucune valeur ne peut être définie par le biais du contrôle.

```
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) // Verrouilllée !!!

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) // Changement

bSwitch = !bSwitch;
else

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

4.2.4.2 Dessin

Lors du tracé du contrôle, les valeurs des variables sont lues depuis les drivers de répartition et l'un des quatre éléments graphiques définis est affiché. Lorsque la valeur d'une variable est modifiée, le contrôle est actualisé par le processus **OnDraw**.

```
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) // La variable existe-t-elle ?
{

vRes = m_dSwitchVar.GetValue();
bVal1 = (BOOL)VariantToDouble(&vRes);
}
if (m_dLatchVar) // La variable existe-t-elle ?
{

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

Les classes résultant de COleDispatchDriver doivent être créées pour l'élément et les variables, afin que l'interface de répartition de zenon puisse être utilisée pour définir les valeurs. La manière la plus simple de créer ces dernières consiste à utiliser l'Assistant Classes de l'environnement de développement

(cliquez sur le bouton **Ajouter une classe**, puis sélectionnez **Depuis une bibliothèque de types** et *zenrt32.tlb*).

Pour notre contrôle, ces classes sont **IElement** et **IVariable**. Elles sont définies dans *zenrt32.h* et *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

Le contrôle **CD_SliderCtrl** comporte l'interface de répartition suivante :

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

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

methods: //*** Méthode du contrôle (pour ActiveX dans zenon)

[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 Contrôle

La mise en œuvre du contrôle s'effectue par le biais de la classe CD_SliderCtrlCtrl. Cette classe comporte un membre **CSliderCtrl** standard de Windows, le contrôle étant sous-classé à cet élément. Les interfaces **IVariable** et **IElement** contiennent des interfaces de zenon devant être intégrées. Celles-ci découlent de **COleDispatchDriver**.

```
class CCD_SliderCtrlCtrl : public COleControl
{

DECLARE_DYNCREATE(CCD_SliderCtrlCtrl)
private: //*** Variables membres

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();
//*** méthodes de conversion de la variante
double VariantToDouble(const VARIANT FAR *vValue);
```

```
DECLARE_OLECREATE_EX(CCD_SliderCtrlCtrl) // Fabrique de classe et guid

DECLARE_OLETYPELIB  (CCD_SliderCtrlCtrl) // GetTypeInfo
DECLARE_PROPPAGEIDS (CCD_SliderCtrlCtrl) // ID de page de propriété
DECLARE_OLECTLTYPE (CCD_SliderCtrlCtrl) // Nom de type et informations d'état diverses


//*** méthodes de fonctionnement de l'élément 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 Méthodes

Les méthodes suivantes sont utilisées :

- CanUseVariables (à la page 26)
- Types de variables (à la page 27)
- MaxVariables (à la page 27)
- zenonInit (à la page 27)
- zenonExit (à la page 28)

4.3.3.1 CanUseVariables

Cette méthode renvoie *1*, ce qui permet d'utiliser des variables de zenon.

```
short CCD_SliderCtrlCtrl::CanUseVariables()
{

return 1 ;
}
```

4.3.3.2 Types de variables

Le contrôle peut utiliser les variables de type mot, octet, double mot et flottante. Vous trouverez une liste des types de données utilisables dans la description (à la page 9) générale de cette méthode.

```
short CCD_SliderCtrlCtrl::VariableTypes()
{

return 0x0001 | // Mot

0x0002 | // Octet
0x0008 | // DWord
0x0010 | // Flottante
0x0020 | // D flottante
}
```

4.3.3.3 MaxVariables

Une seule variable peut être liée à ce contrôle.

```
short CCD_SliderCtrlCtrl::MaxVariables()
{

return 1; // 1 variables
}
```

4.3.3.4 zenonInit

Le paramètre **dispElement** contient l'interface de l'élément dynamique. Cet élément permet de déterminer la variable zenon liée. Si elle est valide, la plage d'affichage du contrôle **SlideCtrl** est définie. En outre, les paramètres d'affichage (nombre de graduations, etc.) sont définis. Si aucune variable n'est liée, la plage d'affichage définie s'étend de *0* à *0*. Le contrôle SliderCtrl ne peut donc pas être modifié. La variable **m_bInitialized** définit la capacité d'ajuster les valeurs.

```
BOOL CCD_SliderCtrlCtrl::zenonInit(LPDISPATCH dispElement)
{
//*** Déterminer la variable à l'aide de l'élément de zenon

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

short nIndex = 0;
```

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

//*** Initialiser la taille du contrôle Slider-Ctrl
if (m_interfaceVariable) {

//*** Définir la plage
m_nRangeStart = (long) VariantToDouble(&m_interfaceVariable.GetRangeMin());
m_nRangeEnd = (long) VariantToDouble(&m_interfaceVariable.GetRangeMax());
m_wndSliderCtrl.SetRange(m_nRangeStart,m_nRangeEnd,TRUE);
//*** Définir les graduations intermédiaires
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

Dans cette méthode, les interfaces de zenon sont à nouveau libérées.

```
BOOL CCD_SliderCtrlCtrl::zenonExit()
{

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

4.3.4 Utilisation et affichage

4.3.4.1 Dessin

DoSuperclassPaint dessine l'élément SliderCtrl (il s'agit d'un contrôle sous-classé). Si le curseur est déplacé lors de l'étape de dessin, la variable **m_bInitialized** reçoit la valeur *FALSE*. Ceci garantit que la valeur peut être modifiée. Normalement, la valeur de la variable est lue et affichée avec la méthode **SetPos** de l'élément SliderCtrl.

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

//*** actualiser l'affichage
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 Écrire valeur prescrite

Dans la méthode **LButtonDown**, la variable **m_bInitialized** est définie sur *FALSE* et, dans l'événement **LbuttonUp**, elle est à nouveau définie sur *TRUE*. Ceci garantit que la valeur peut être modifiée. Dans le cas contraire, la procédure **OnDraw**serait exécutée et l'ancienne valeur serait affichée.

```
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);
}
```

Une valeur est transmise au matériel lorsque le curseur est déplacé. Dans les méthodes **Hscroll** et **Vscroll**, la valeur est transmise au matériel (selon la configuration horizontale ou verticale du curseur).

```
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:{

//*** Définir la valeur sans boîte de dialogue ?
int nValue = m_wndSliderCtrl.GetPos();
COleVariant cValue((short) nValue,VT_I2);
m_interfaceVariable.SetValue(cValue);
}
}
}
```

4.3.5 zenon Interface

Les classes résultant de COleDispatchDriver doivent être créées pour l'élément et les variables, afin que l'interface de répartition de zenon puisse être utilisée pour définir les valeurs. La manière la plus simple de créer ces dernières consiste à utiliser l'Assistant Classes de l'environnement de développement (cliquez sur le bouton **Ajouter une classe**, puis sélectionnez **Depuis une bibliothèque de types** et *zenrt32.tlb*).

Pour notre contrôle, ces classes sont **IElement** et **IVariable**. Elles sont définies dans *zenrt32.h* et *zenrt32.cpp*.

4.4 Exemple : contrôle. NET exécuté en tant que contrôle ActiveX (C#)

L'exemple suivant décrit un contrôle .NET exécuté comme un contrôle ActiveX dans zenon.

La création et l'intégration se déroulent en quatre étapes :

1. Création d'un contrôle de formulaire Windows (à la page 31)
2. Conversion d'un contrôle utilisateur .NET en double contrôle (à la page 34)
3. Utilisation avec ActiveX dans Editor via le code VBA (à la page 38)

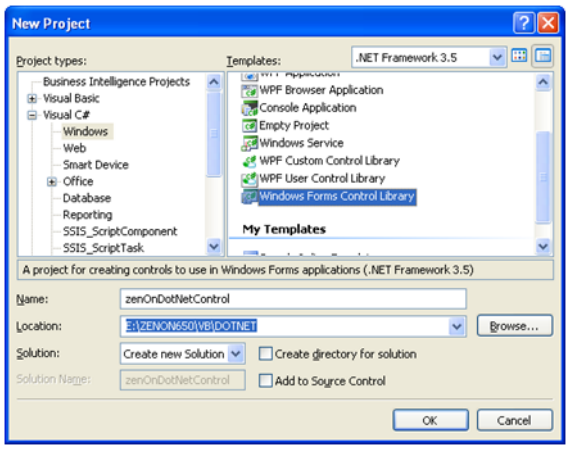
4. <Connexion de variables de CD_PRODUCTNAME> au contrôle utilisateur .NET (à la page 39)

> **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**
>
> Les captures d'écran de cette section sont uniquement disponibles en anglais.

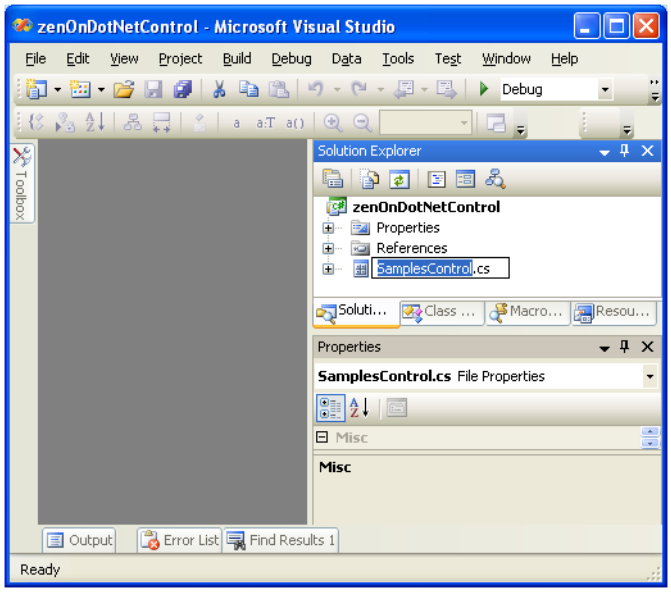
4.4.1 Création d'un contrôle de formulaire Windows

Pour créer un contrôle de formulaire Windows :

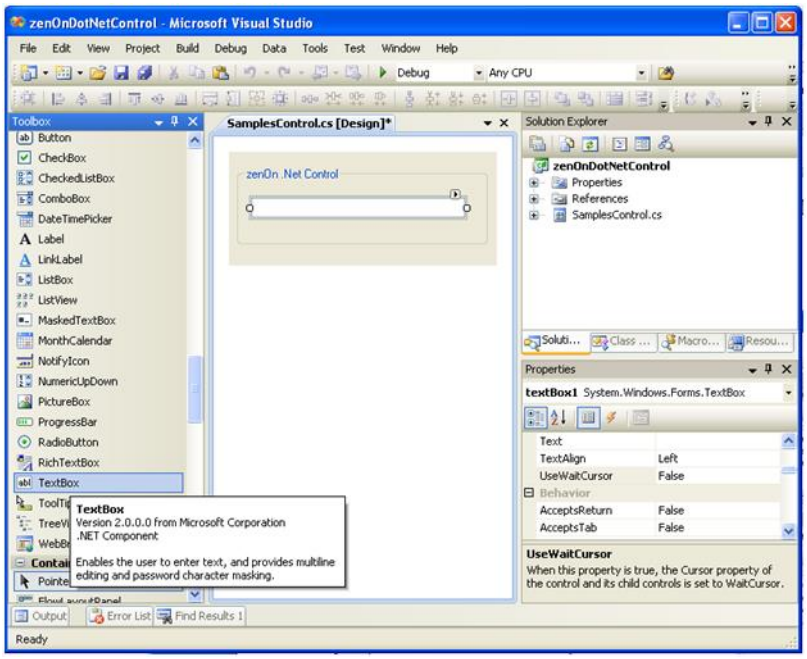
1. Démarrez Visual Studio 2008 et créez un nouveau projet **Windows Form Control Library** (Bibliothèque de contrôles de formulaire Windows) :



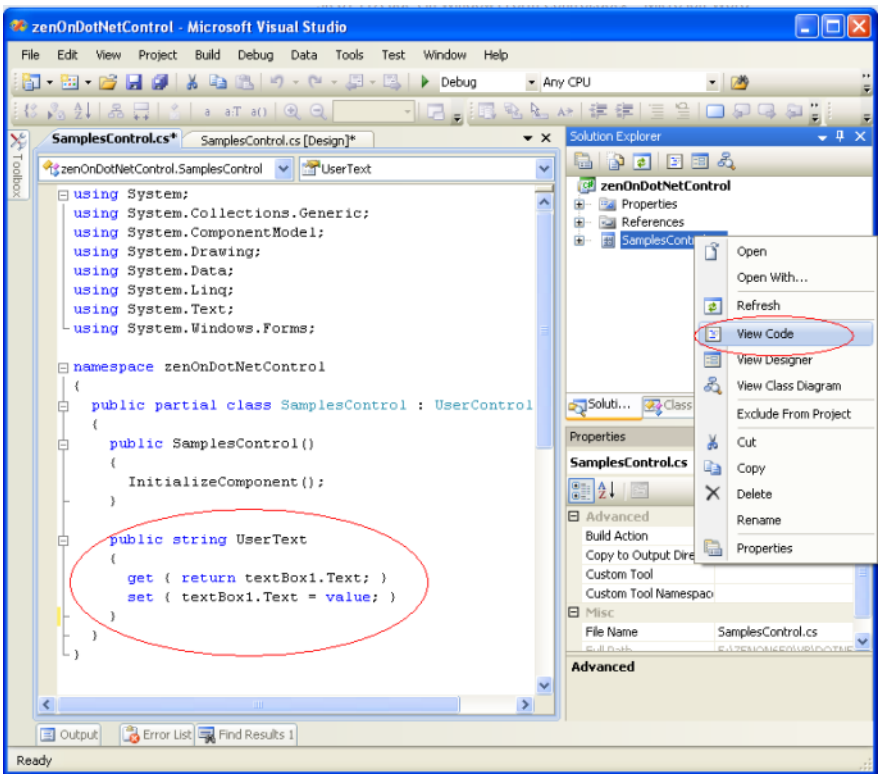
2. Renommez le contrôle par défaut avec le nom de contrôle souhaité.
   Pour cet exemple : **SampesControl.cs**.



3. Ouvrez le concepteur de contrôles et ajoutez le contrôle souhaité (dans notre cas, un champ de texte) :



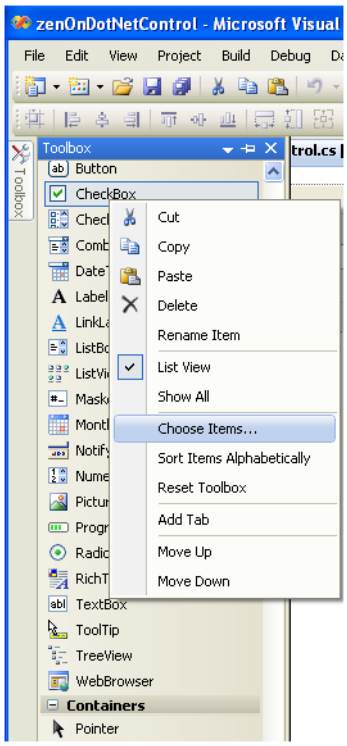
4. Les contrôles comportent normalement des propriétés. Ouvrez le concepteur de code en sélectionnant la fonction **Afficher le code**, puis ajoutez les propriétés devant que vous souhaitez rendre disponibles au niveau externe.
   Pour cet exemple : la propriété visible au niveau externe "**UserText**", avec l'accès **get** et **set**, qui contient le texte du champ de texte :



5. Compilez le projet.

   Le contrôle de formulaire Windows peut maintenant être utilisé dans d'autres projets de formulaires Windows.

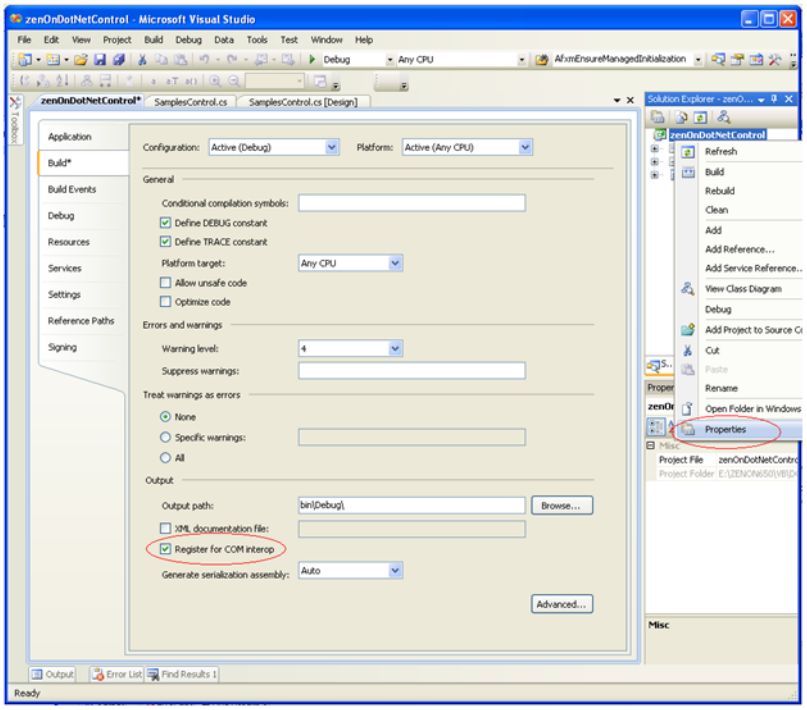
Important : le contrôle doit être inséré manuellement dans la boîte à outils de contrôle par le biais de la fonction **Choisir les éléments**.



4.4.2 Conversion d'un contrôle utilisateur .NET en double contrôle

Pour convertir un contrôle .NET en double contrôle, vous devez d'abord activer l'interface COM pour ActiveX.

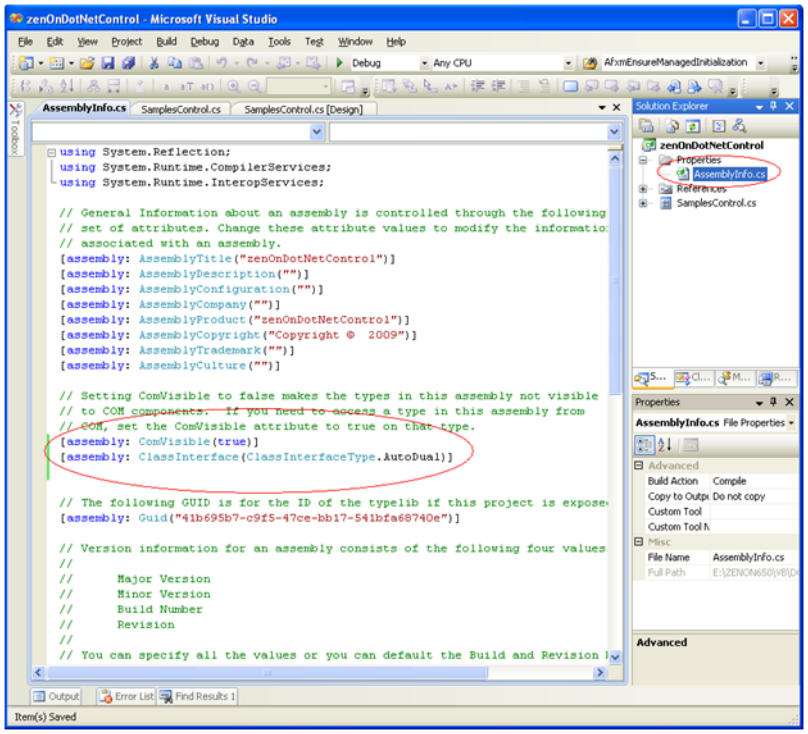
1. Ouvrez le projet et activez la propriété **Inscrire pour COM interop** dans les paramètres **Générer** :



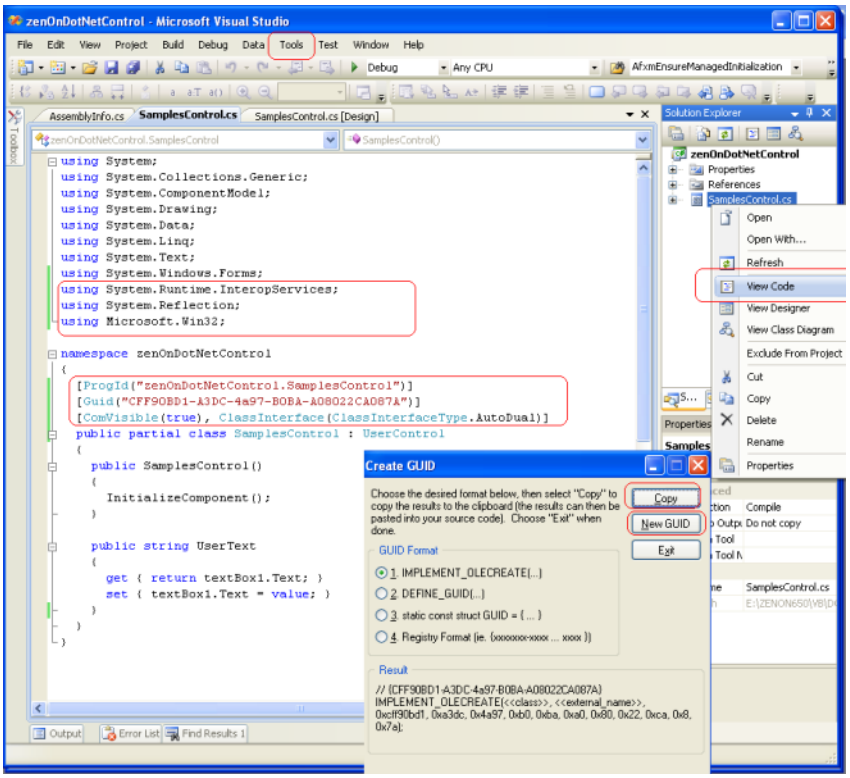
2. Ouvrez le fichier **AssemblyInfo.cs** et
   - définissez l'attribut **ComVisible** sur *true*
   - ajoutez l'attribut **ClassInterface**

   [ensemble : ComVisible(true)]

[ensemble : ClassInterface(ClassInterfaceType.AutoDual)]

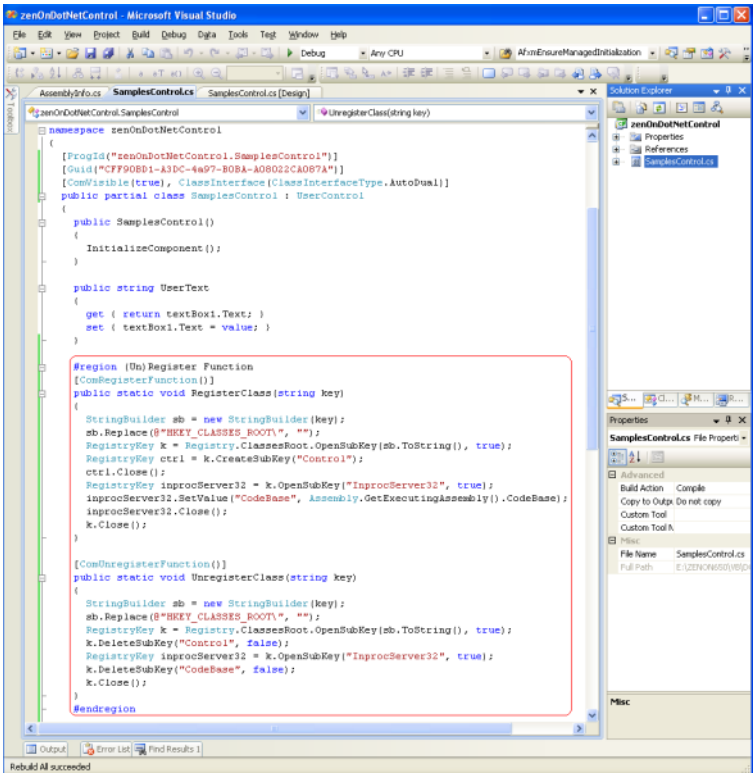


3. Ouvrez le concepteur de code en sélectionnant l'option **Afficher le code**, puis ajoutez les attributs ActiveX et les entrées **using** requis. Dans le menu **Outils/Create GUID**, créez un nouveau GUID pour l'attribut GUID :



4. Pour que le contrôle puisse être sélectionné comme un contrôle d'interface utilisateur Active X, vous devez ajouter des fonctions aux classes de contrôle suivantes :

   - RegisterClass
   - UnregisterClass



   Vous pouvez ensuite inscrire le contrôle dans la base de registres.

5. Recompilez le projet.

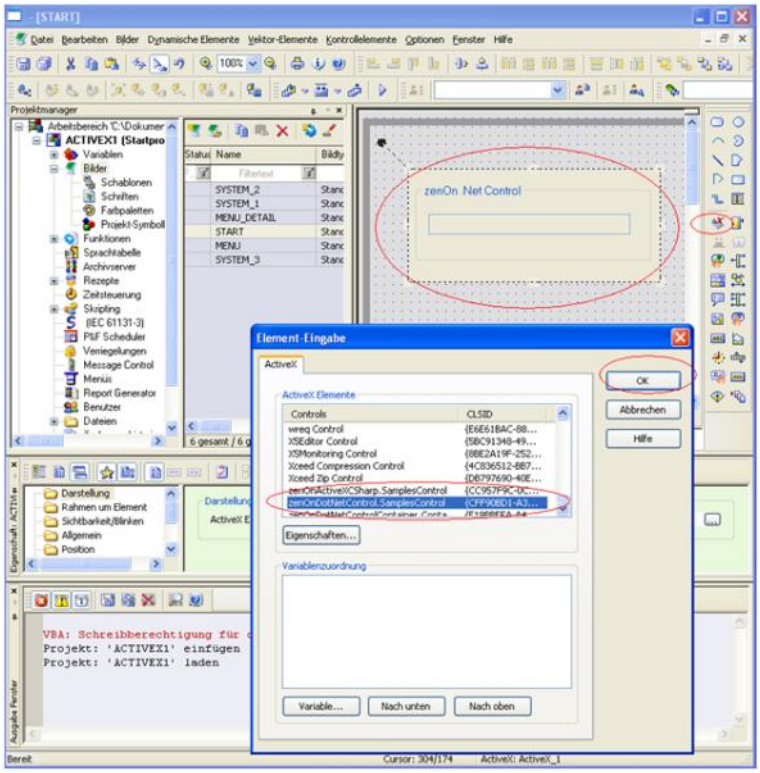
   Le contrôle de formulaire Windows est maintenant utilisable dans ActiveX, et a été inscrit automatiquement durant la création de la nouvelle version. Un fichier typelib supplémentaire, **zenonDotNetControl.tlb**, a été créé dans le répertoire de sortie.

6. Pour utiliser le contrôle sur un autre ordinateur :

   a) Copiez le fichier DLL et le fichier TLB sur l'ordinateur cible

   b) Enregistrez les fichiers en saisissant la ligne de commande :
   %windir%\Microsoft.NET\Framework\v2.0.50727\regasm.exe zenonDotNetControl.dll /tlb:zenonDotNetControl.tlb

7. Ajoutez le contrôle de formulaire étendu de Windows en tant que contrôle ActiveX dans zenon Editor :



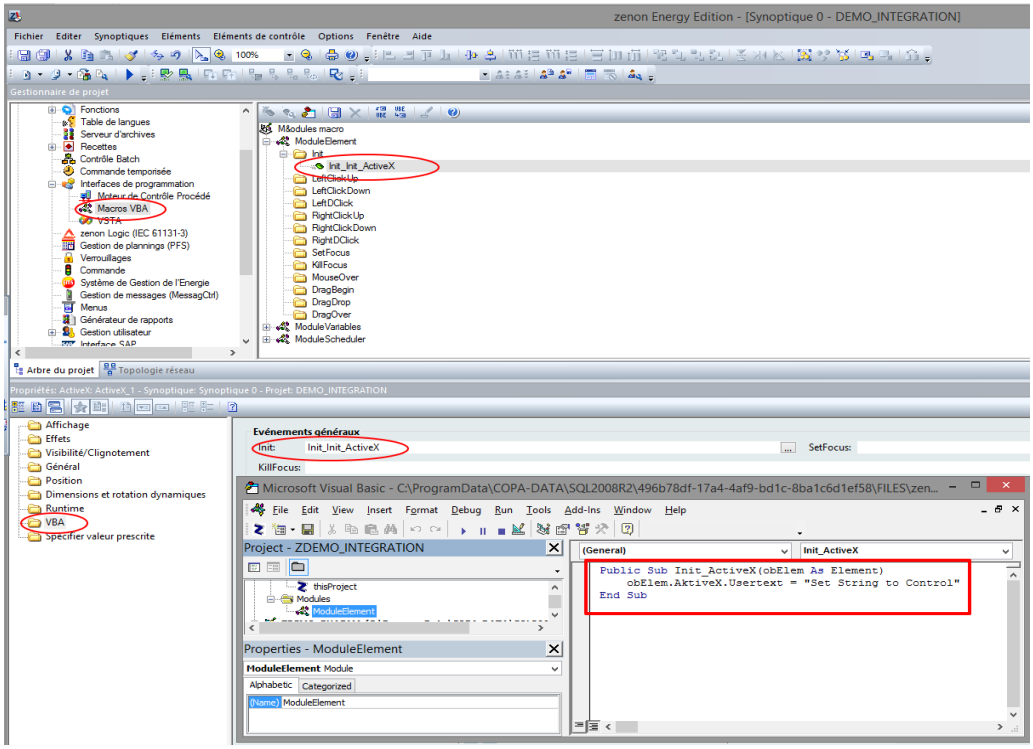
4.4.3 Utilisation avec ActiveX dans Editor via le code VBA

Pour accéder aux propriétés du contrôle dans zenon Editor :

1. Dans zenon Editor, dans le nœud **Interfaces de programmation/Macros VBA**, créez une nouvelle macro **Init** avec le nom **Init_ActiveX**.

   Dans cette macro, vous pouvez accéder à toutes les propriétés externes via **obElem.ActiveX**.

2. Attribuez cette macro au contrôle ActiveX via les propriétés **macros VBA/Init** de l'élément ActiveX.



EXEMPLE DE MACRO INIT

```
Public Sub Init_ActiveX(obElem As Element)
obElem.AktiveX.Usertext = "Attribuez la chaîne au contrôle"
End Sub
```

4.4.4 <Connexion de variables de CD_PRODUCTNAME> au contrôle utilisateur .NET

Dans zenon, vous avez la possibilité d'améliorer un contrôle ActiveX avec des fonctions spéciales pour accéder à l'API zenon.

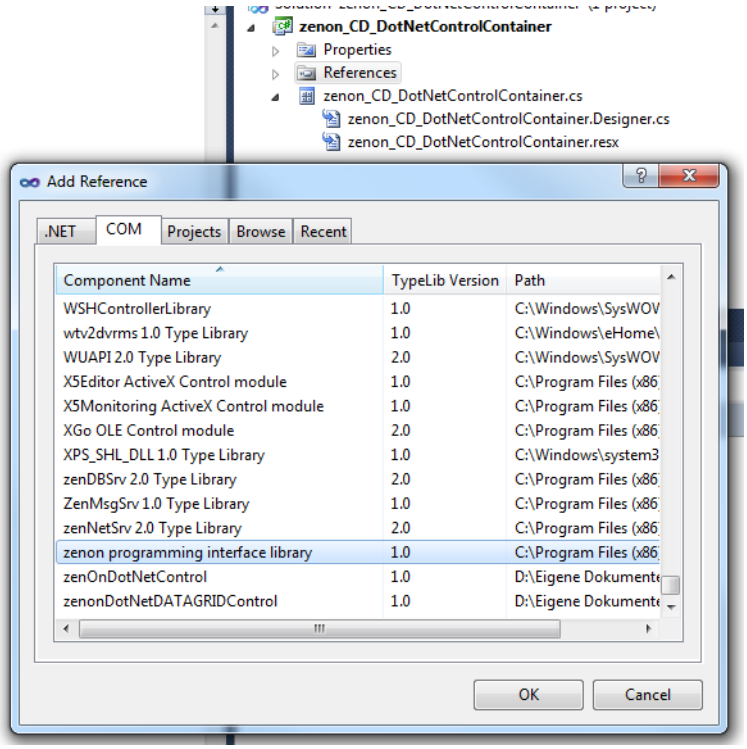
MÉTHODES REQUISES

- public bool zenonInit (à la page 41) (appelé durant l'initialisation du contrôle dans le Runtime de zenon.)
- public bool zenonInitED (à la page 41) (utilisé dans Editor.)
- public bool zenonExit() (à la page 42) (appelé lors de la destruction du contrôle dans le Runtime de zenon.)

- public bool zenonExitED() (à la page 42) (utilisé dans Editor.)
- public short CanUseVariables() (à la page 42) (prend en charge la liaison de variables.)
- public short VariableTypes() (à la page 42) (types de données pris en charge par le contrôle)
- public MaxVariables() (à la page 43) (nombre maximal de variables pouvant être liées au contrôle.)

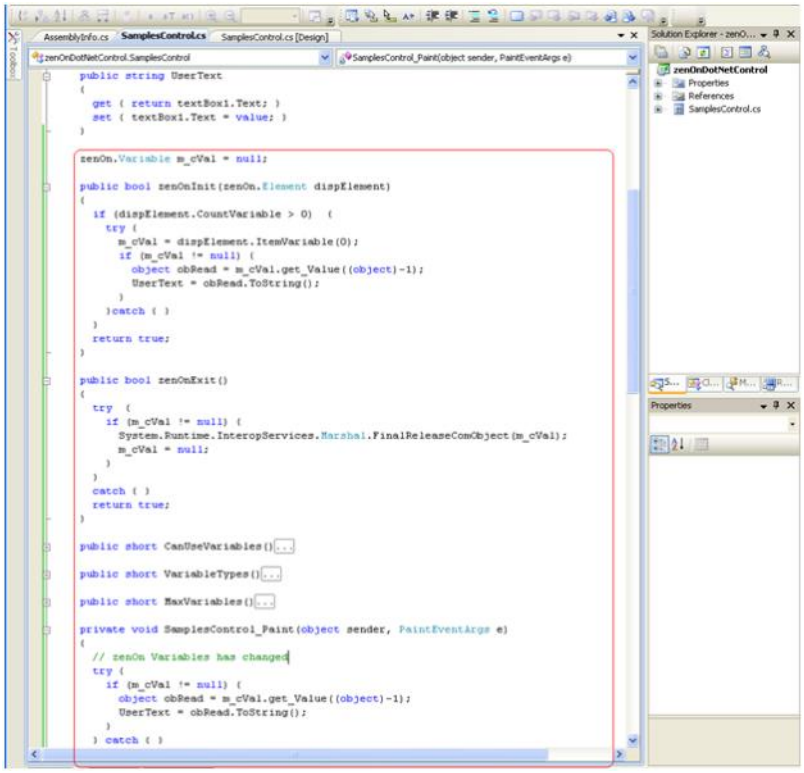
AJOUT DE RÉFÉRENCES

1. Dans Microsoft Visual Studio, sous **Ajouter des références**, sélectionnez la bibliothèque d'objets de zenon pour pouvoir accéder à l'API de zenon dans le contrôle.



2. Ajoutez les fonctions améliorées au code de classe du contrôle pour accéder à l'ensemble de l'API de zenon.

Dans notre exemple, l'objet COM d'une variable de zenon est temporairement enregistré dans un **membre**, afin d'être accessible ultérieurement via l'événement **Paint** du contrôle.



4.4.4.1 public bool zenOnInit(zenOn.Element dispElement)

Cette méthode (dans le Runtime) permet au contrôle ActiveX d'obtenir un pointeur vers l'interface de répartition de l'élément dynamique. Ce pointeur permet également d'accéder aux variables de zenon liées à l'élément dynamique.

Vous pouvez configurer l'ordre de transmission des variables dans la boîte de dialogue Entrer les propriétés, à l'aides des boutons **Bas** et **Haut**. La boîte de dialogue Entrer les propriétés s'affiche si :

- Vous double-cliquez sur l'élément ActiveX, ou
- Vous sélectionnez **Propriétés** dans le menu contextuel, ou
- Vous sélectionnez la propriété **Paramètres ActiveX** dans le nœud **Affichage** de la fenêtre de propriétés

4.4.4.2 public bool zenonInitED(zenon.Element dispElement)

Équivalent à public bool zenonInit (à la page 41) ; exécuté lors de l'ouverture d'ActiveX dans Editor (double-cliquez sur le contrôle ActiveX).

4.4.4.3 public bool zenOnExit()

Cette méthode est appelée par le Runtime de zenon lorsque le contrôle ActiveX est fermé. Ici, tous les pointeurs de répartition renvoyant à des variables doivent être libérés.

4.4.4.4 public bool zenOnExitED()

Equals public bool zenOnExit() (à la page 42) 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()

Cette méthode renvoie *1* si le contrôle peut utiliser les variables de zenon, et *0* dans le cas contraire.

- *1* : Pour l'élément dynamique (via le bouton **Variable**), vous pouvez uniquement déclarer des variables de zenon avec le type déclaré par le biais de la méthode VariableTypes, conformément au nombre défini par la méthode MaxVariables.
- *0* : si CanUseVariables renvoie *0* ou le contrôle ne comporte pas cette méthode, n'importe quel nombre de variables de tout type peut être défini, sans limitation aucune. Dans le Runtime, toutefois, ces variables sont uniquement utilisables avec du code 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()

Dans cette section est défini le nombre de variables pouvant être sélectionnées dans la liste de variables.

1 : la sélection multiple est désactivée dans la liste de variables. Un avertissement est alors affiché si plusieurs variables sont sélectionnées.

5 Contrôles utilisateur .NET

Le contrôle .NET permet d'améliorer de manière autonome les fonctionnalités du Runtime de zenon et d'Editor.

Dans ce manuel, vous trouverez des informations concernant :

- La différence entre un conteneur de contrôle et ActiveX (à la page 43)
- Exemple de conteneur de contrôle .NET (à la page 44)
- Exemple : contrôle. NET exécuté en tant que contrôle ActiveX (C#) (à la page 30)

Vous trouverez d'autres informations concernant les contrôles .NET dans ActiveX dans le manuel Synoptiques, au chapitre Contrôles .NET.

> **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).

CD_DotNetControlContainer control	ActiveX control
▸ 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.

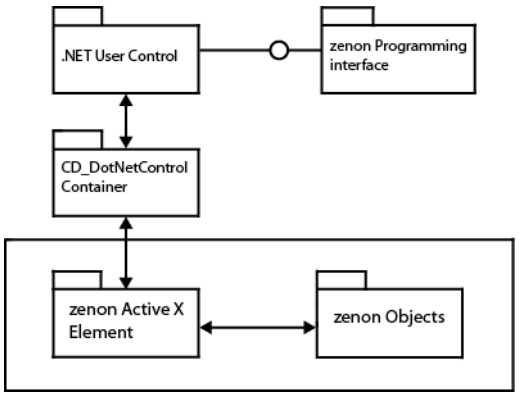
5.2 Exemple de conteneur de contrôle .NET

Dans ce didacticiel, vous allez découvrir comment créer un contrôle utilisateur .NET simple dans Visual Studio 2010 (langage de programmation **C#**) et comment l'intégrer, à l'aide du contrôle **CD_DotNetControlContainer** de zenon, sous forme de contrôle ActiveX dans un élément ActiveX de zenon.

5.2.1 Général

Le contrôle CD_DotNetControlContainer agit donc comme un wrapper entre le contrôle utilisateur et l'élément ActiveX de zenon. Toutes les méthodes utilisées dans l'exemple suivant, et toutes les méthodes publiques et propriétés sont transmises, via le contrôle CD_DotNetControlContainer, du contrôle utilisateur au contrôle ActiveX, et peuvent être utilisés par zenon (dans VBA et VSTA également).

Si une référence à l'interface de programmation de zenon est présente dans le contrôle utilisateur, vous pouvez directement accéder aux objets de >CD_PRODUCTNAME<.



Dans l'exemple suivant, nous allons :

- Créer un contrôle utilisateur .NET (à la page 46)
- Ajouter un conteneur CD_DotNetControlContainer et un contrôle utilisateur .NET (à la page 55)
- Autoriser l'accès au contrôle utilisateur via VSTA (VBA) (à la page 60)

CHEMIN DE LA DLL DANS EDITOR ET LE RUNTIME

Le chemin d'accès à **.Net DLL** sélectionné dans Editor est également utilisé dans le Runtime. Il est défini comme un chemin absolu, et ne peut pas être modifié.

Assurez-vous d'utiliser le même chemin sur tous les ordinateurs sur le réseau zenon pour Editor et le Runtime.
Remarque : Sélectionnez un chemin absolu, par exemple : *C:\Controls*. Saisissez le chemin tel qu'il est défini dans **Remote-Transport** et dans **.NET Control Container**. Utilisez **Remote-Transport** pour harmoniser ce chemin avec tous les ordinateurs.

5.2.1.1 public bool zenOnInit(zenOn.Element dispElement)

Cette méthode (dans le Runtime) permet au contrôle ActiveX d'obtenir un pointeur vers l'interface de répartition de l'élément dynamique. Ce pointeur permet également d'accéder aux variables de zenon liées à l'élément dynamique.

Vous pouvez configurer l'ordre de transmission des variables dans la boîte de dialogue Entrer les propriétés, à l'aides des boutons **Bas** et **Haut**. La boîte de dialogue Entrer les propriétés s'affiche si :

- Vous double-cliquez sur l'élément ActiveX, ou
- Vous sélectionnez **Propriétés** dans le menu contextuel, ou
- Vous sélectionnez la propriété **Paramètres ActiveX** dans le nœud **Affichage** de la fenêtre de propriétés

5.2.1.2 public bool zenOnExit()

Cette méthode est appelée par le Runtime de zenon lorsque le contrôle ActiveX est fermé. Ici, tous les pointeurs de répartition renvoyant à des variables doivent être libérés.

5.2.1.3 public short CanUseVariables()

Cette méthode renvoie *1* si le contrôle peut utiliser les variables de zenon, et *0* dans le cas contraire.

- *1* : Pour l'élément dynamique (via le bouton **Variable**), vous pouvez uniquement déclarer des variables de zenon avec le type déclaré par le biais de la méthode VariableTypes, conformément au nombre défini par la méthode MaxVariables.
- *0* : si CanUseVariables renvoie *0* ou le contrôle ne comporte pas cette méthode, n'importe quel nombre de variables de tout type peut être défini, sans limitation aucune. Dans le Runtime, toutefois, ces variables sont uniquement utilisables avec du code 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*
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()

Dans cette section est défini le nombre de variables pouvant être sélectionnées dans la liste de variables.

1 : la sélection multiple est désactivée dans la liste de variables. Un avertissement est alors affiché si plusieurs variables sont sélectionnées.

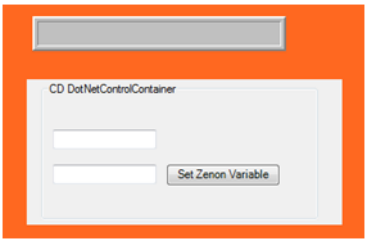
5.2.2 Créer un contrôle utilisateur .NET

Un contrôle utilisateur est un contrôle simple, permettant de définir une nouvelle valeur par le biais d'un champ de saisie (champ de texte). Lorsque vous cliquez sur le bouton, la valeur est écrite dans la variable zenon de votre choix.

Une fonction supplémentaire doit automatiquement détecter tout changement de valeur de la variable dans zenon et afficher automatiquement la nouvelle valeur dans le contrôle.

Information

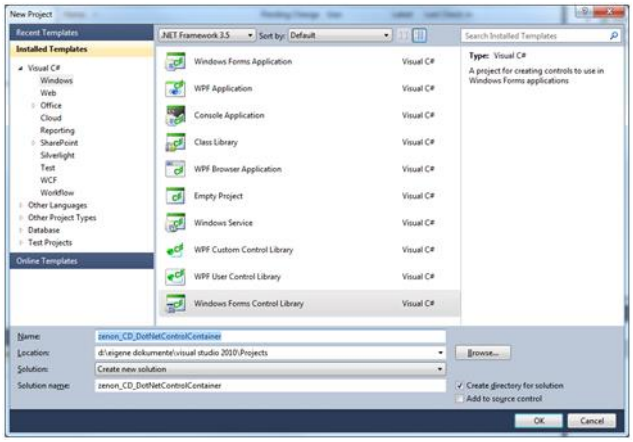
Les captures d'écran de cette section sont uniquement disponibles en anglais.



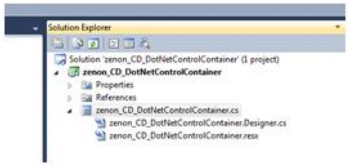
PROCÉDURE

1. Créez d'abord un nouveau projet dans VS, et utilisez le type de projet **Bibliothèque de contrôles Windows Forms**

   Important : spécifiez la version 3.5 de .NET Framework !



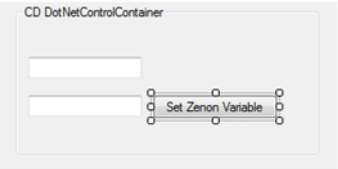
2. Renommez ensuite le fichier CS de "**UserControl**" en "**zenon_CD_DotNetControlContainer.cs**". Les fichiers **Designer.cs** et **.resx** sont renommés automatiquement.



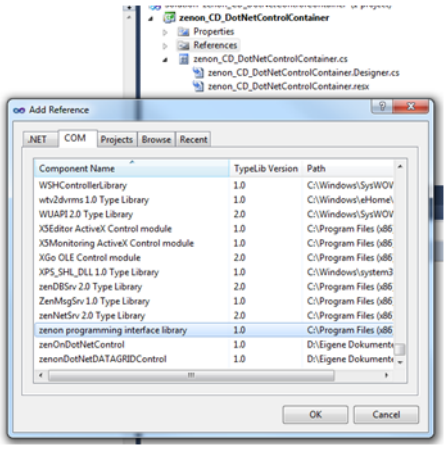
3. À l'étape suivante, vous créez le contrôle utilisateur. Pour cela, utilisez deux champs de texte pour l'entrée et la sortie, respectivement, et un bouton permettant d'écrire de nouvelles valeurs dans la variable zenon.
   Nom :

   - Premier champ de texte : "**txtGetzenonVariable**"
   - Deuxième champ de texte : "**txtSetzenonVariable**"

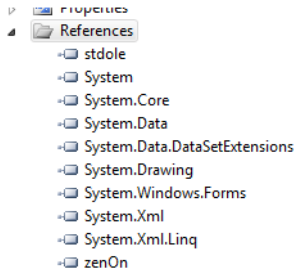
- Bouton : "**btnSetzenonVariable**"



4. Pour accéder aux objets de zenon, vous devez disposer d'une référence à l'interface de programmation de zenon. Pour cela :
   - Cliquez sur le nœud **Références** dans l'Explorateur de solutions
   - Ouvrez menu contextuel
   - Sélectionnez **Ajouter des références...**
   - Sélectionnez l'onglet **COM**
   - Sélectionnez **zenon programming interface library** (Bibliothèque d'interfaces de programmation de zenon)



La référence **zenon** devrait alors être visible dans la liste de références.



5. À l'étape suivante, créez une variable globale du type *zenon.variable* dans le code de **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. Cette variable est initialisée par le biais de la méthode publique **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                     txtGetZenonVariable.Text = m_cVal.get_Value(0).ToString();
42                 }
43                 catch { }
44             }
45             return true;
46         }
```

et activée via la méthode publique **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         }
```

Dans les méthodes suivantes, nous définissons si des variables et des types de données de zenon sont utilisés, ainsi que le nombre de variables pouvant être transmis :

```
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. À l'étape suivante, dans le paramètre **Click-Event** du bouton **btnSetzenonVariable**, définissez qu'un clic sur le bouton écrit la valeur du champ de texte **txtSetzenonVariable** dans la variable zenon, avant d'effacer le contenu du champ de texte.

```
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. Pour réagir à un changement de valeur de la variable, vous devez avoir accès à l'événement **Paint** du contrôle. L'événement **Paint** est également déclenché si la valeur de la variable initialisée de zenon change, et peut donc être utilisé pour actualiser des valeurs. Puisque les variables référencées dans l'élément ActiveX de zenon sont automatiquement *suggérées*, vous n'avez généralement pas besoin d'utiliser le conteneur **zenon.OnlineVariable** dans le contrôle.

```
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     }
```

APERÇU DU CODE

Voici un récapitulatif de l'ensemble du code :

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

```
public partial class zenon_CD_DotNetControlContainer : UserControl
{
  //Cette ligne est nécessaire pour accéder au conteneur de variable de zenon
zenOn.Variable m_cVal = null;
public zenon_CD_DotNetControlContainer()
{
InitializeComponent();
}
  /// <summary>
  /// Cette méthode publique sera appelée par l'initialisation du contrôle dans
  /// le Runtime de zenon.
  /// </summary>
  /// <param name="dispElement"></param>
  /// <returns></returns>
public bool zenOnInit(zenOn.Element dispElement)
{
//Vérifier si des variables de zenon ont été ajoutées au
  //Contrôle
if (dispElement.CountVariable > 0)
{
try
{
  //Prendre la première variable de zenon et l'ajouter
  //à la variable globale
m_cVal = dispElement.ItemVariable(0);
}
catch { }
}
```

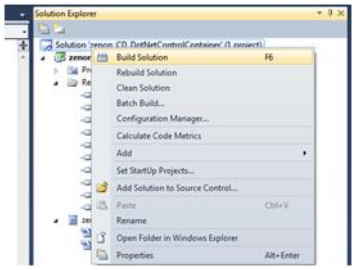
```
return true;
}
  // <summary>
  /// Cette méthode publique sera appelée par la libération du contrôle dans
  /// le Runtime de zenon.
  /// </summary>
  /// <returns></returns>
public bool zenOnExit()
{
try
{
if (m_cVal != null)
{
//Libérer la variable de zenon (objet Com)
System.Runtime.InteropServices.Marshal.ReleaseComObject(m_cVal);
m_cVal = null;
}
}
catch { }
return true;
}
  /// <summary>
  /// Cette méthode publique est nécessaire pour lier les variables de zenon
  /// au contrôle.
  /// </summary>
  /// <returns></returns>
public short CanUseVariables()
{
```

```
  return 1; // Only this Variable is supported
}
/// <summary>
/// Cette méthode publique renvoie le type des
/// variables de zenon prises en charge
/// </summary>
/// <returns></returns>
public short VariableTypes()
{
return short.MaxValue; // Tous les types de données pris en charge
}
/// <summary>
/// Cette méthode publique renvoie le nombre de
/// variables de zenon prises en charge
/// </summary>
/// <returns></returns>
public short MaxVariables()
{
return 1; // Une seule variable doit être liée au contrôle
}
/// <summary>
/// Celle-ci sera déclenchée en cliquant sur le bouton. La nouvelle valeur sera
/// définie sur la variable de zenon
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btnSetZenonVariable_Click(object sender, EventArgs e)
{
```

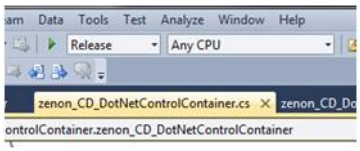
```
//Valeur prescrite de TextBox pour la variable de zenon
m_cVal.set_Value(0,txtSetZenonVariable.Text.ToString());
this.txtSetZenonVariable.Text = string.Empty;
}
/// <summary>
/// Celle-ci sera déclenchée par le tracé du contrôle utilisateur ou la modification de la valeur de la variable.
/// Après la modification de la variable, le contrôle est tracé une nouvelle fois et la nouvelle valeur
/// est écrite dans le champ de texte.
/// </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;
}
}
}
}
```

CRÉATION DE LA VERSION

Pour finir, créez une version pour intégrer la DLL finalisée dans zenon ou le conteneur **CD_DotNetControlContainer**.



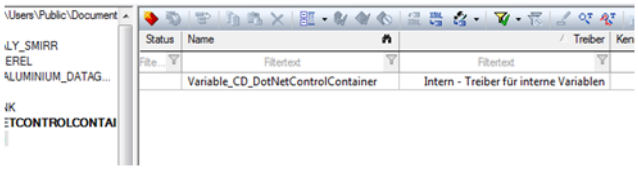
Pour cela, vous devez basculer du mode **Débogage** au mode **Version finale** dans les paramètres.



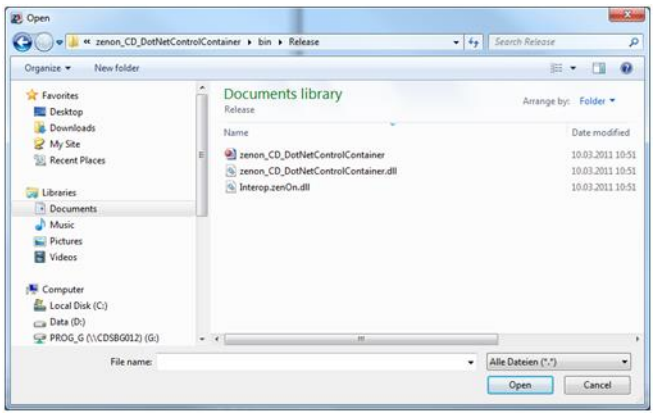
5.2.3 Ajouter un conteneur CD_DotNetControlContainer et un contrôle utilisateur .NET

Pour préparer le projet zenon et ajouter le conteneur **CD_DotNetControlContainer** et le **contrôle utilisateur .NET**, effectuez les étapes suivantes :

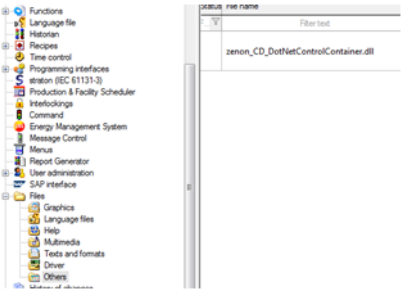
1. Créez une variable interne de type *String* (Chaîne), et définissez la longueur de la chaîne sur *30*.



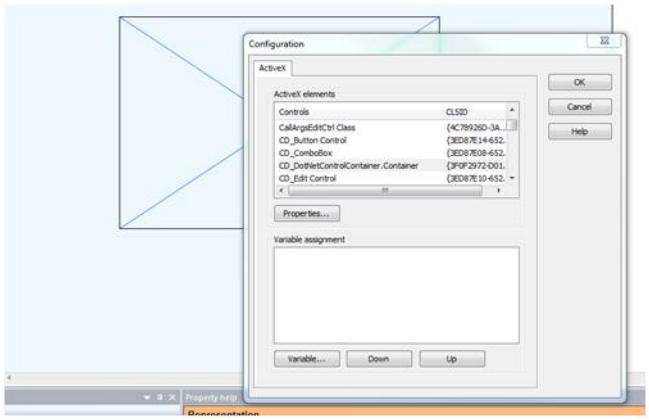
2. Dans le nœud *Projet/Fichiers/Autres* du projet zenon, ajoutez le fichier DLL du contrôle utilisateur .NET que vous avez créé. Ici, le fichier est copié vers le fichier **Additional** au niveau du système de fichiers.

3. 

Le fichier DLL se trouve dans le dossier de sortie du projet de Visual Studio, sous *bin\Release\zenon_CD_DotNetControlContainer.dll*.

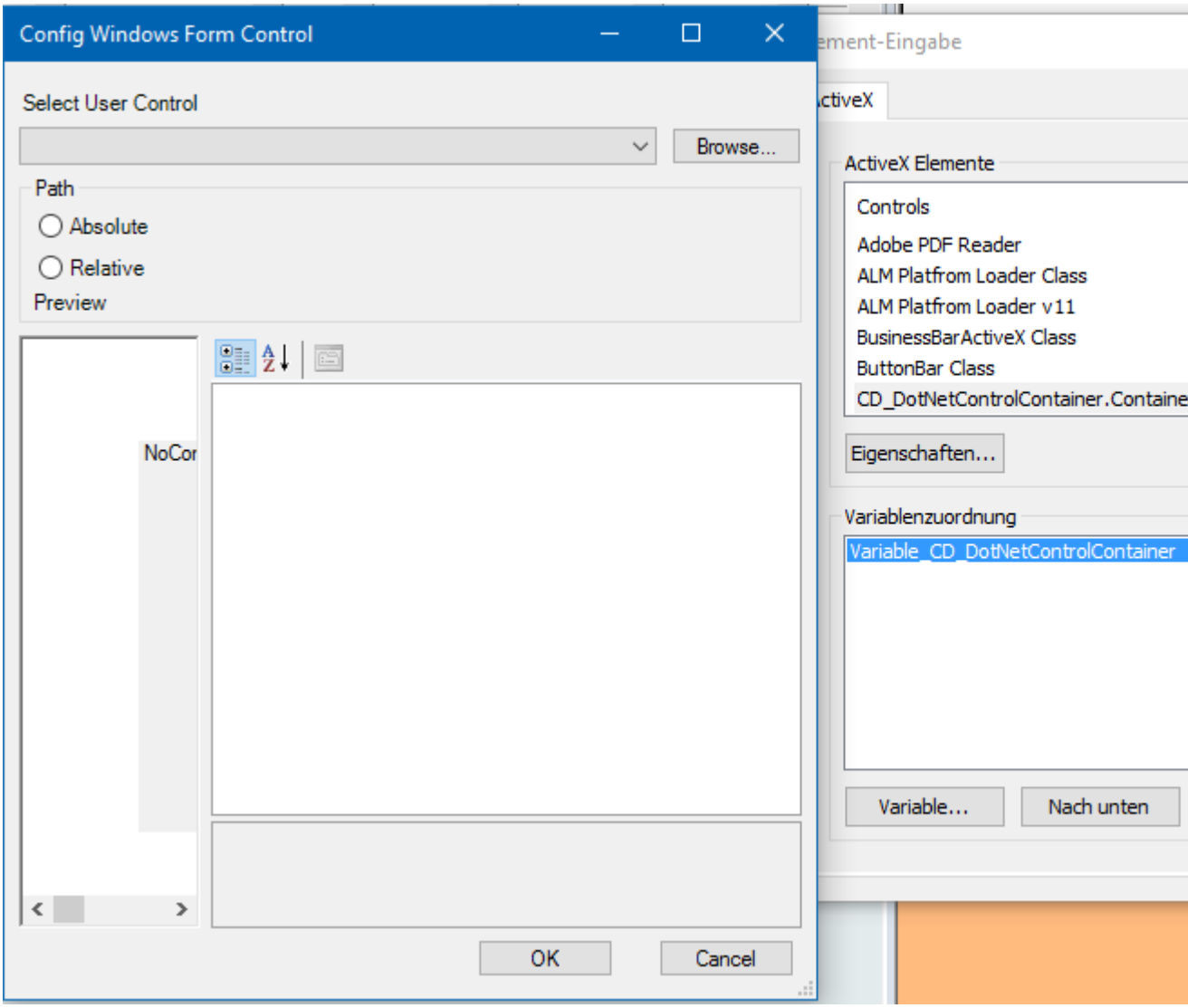


4. Dans le projet, sélectionnez l'élément ActiveX et faites-le glisser vers un synoptique zenon.
   - La boîte de dialogue **Configuration** s'affiche à l'écran.
   - Sélectionnez le contrôle **CD_DotNetControlContainer.Container**.



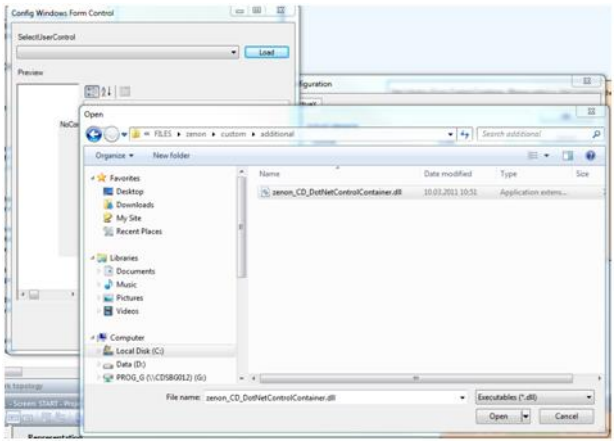
5. Pour incorporer le contrôle utilisateur .NET au contrôle **CD_DotNetControlContainer** :
   - Cliquez sur le bouton **Propriétés**.

- Une nouvelle boîte de dialogue s'affiche à l'écran.



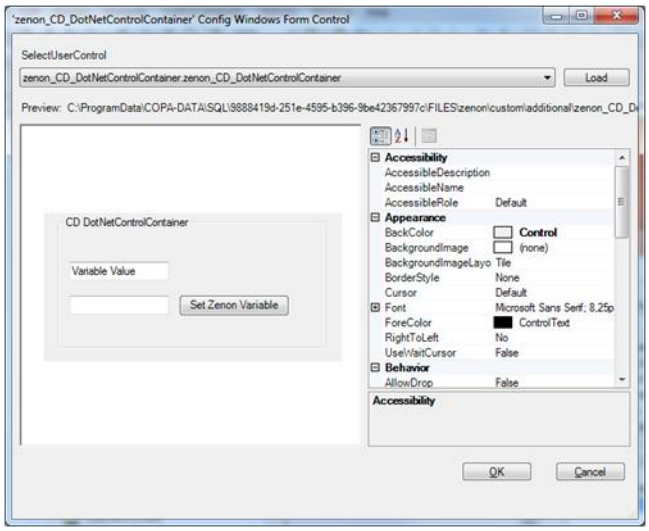
- Cliquez sur le bouton **Charger** pour sélectionner le chemin du dossier du projet, par exemple : *C:\ProgramData\COPA-DATA\SQL\9888419d-251e-4595-b396-9be42367997c\FILES\zenon\custom\additional\zenon_CD_DotNetControlContainer.dll*

Remarque : Les contrôles doivent toujours être enregistrés dans le dossier **Additional** du projet. Ils sont ainsi pris en compte pendant les sauvegardes et les transferts. Le chemin doit alors être défini sur un emplacement relativement proche de l'emplacement de sauvegarde. Vous pouvez également, pour le **Path**, utiliser l'option *Absolute* ; dans ce cas, toutefois, vous devez vous assurer que la même structure de répertoire est présente sur le système cible.

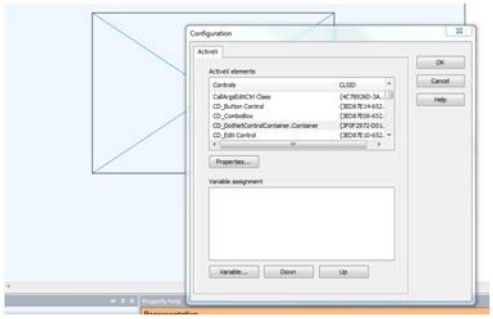


Le contrôle utilisateur .NET devrait maintenant être affiché.

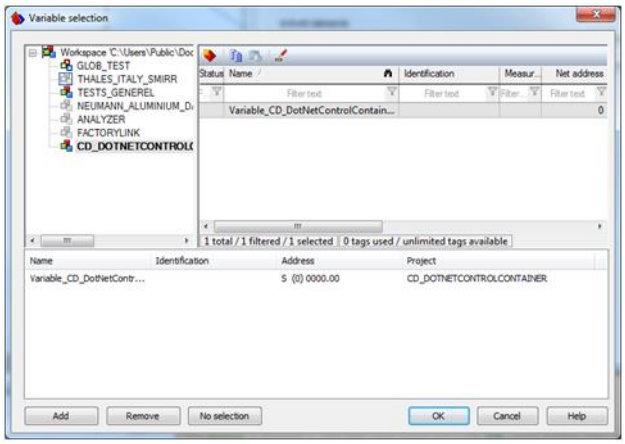
Fermez la boîte de dialogue en cliquant sur **OK**.



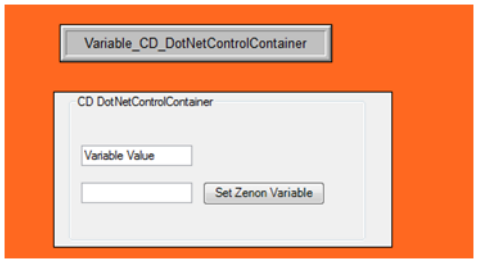
6. À la dernière étape, liez une variable au contrôle en cliquant sur le bouton **Variables**.



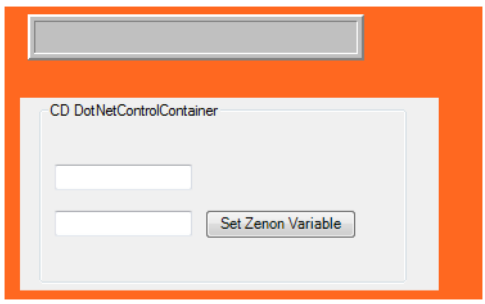
La variable sélectionnée est d'abord automatiquement liée avec notre variable globalement définie (.NET UserControl) via la méthode publique `zenonInit`. La liaison avec le contrôle est effectuée après le démarrage du Runtime.



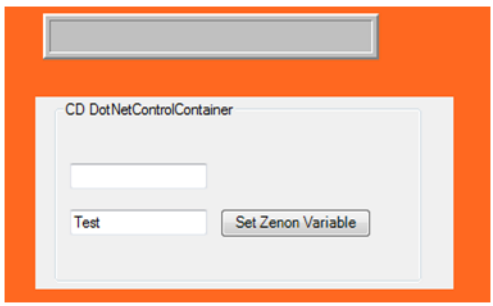
Liez ensuite la variable interne à un élément de texte.



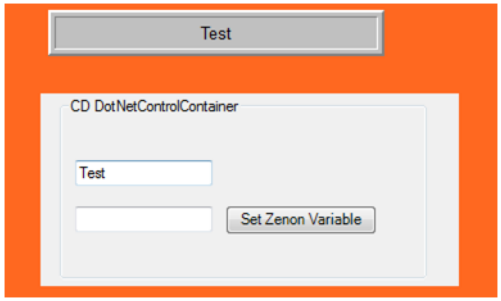
7. Après le démarrage du Runtime, le contrôle est initialement vide.



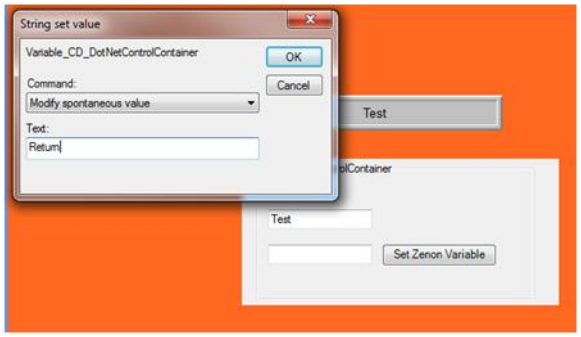
Si vous saisissez une valeur dans le deuxième champ de texte et la validez ensuite en cliquant sur le bouton **Set zenon variable** (Définir la variable de zenon), la valeur est écrite dans la variable zenon. (L'événement **btnSetzenonVariable_Click** est exécuté.)



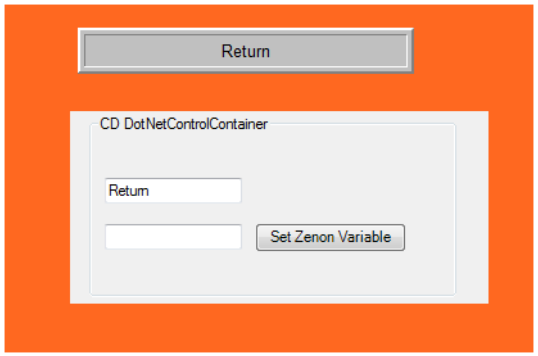
Cette indication est également affichée dans l'élément de texte de zenon.



Si la valeur est directement modifiée dans l'élément de texte de zenon,



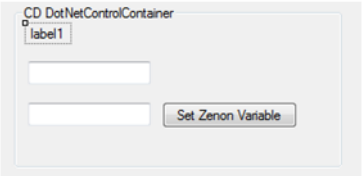
la valeur est directement écrite dans le premier champ de texte par le biais de l'événement **Paint** du contrôle .NET.



5.2.4 Accès au contrôle utilisateur via VSTA ou VBA

Ces exemples présentent l'accès via VSTA. La procédure est la même que pour VBA.

1. Améliorez le contrôle en spécifiant un intitulé (**label**) et nommez-le **lblzenonInfo**. Dans cet intitulé, la valeur d'une autre variable de zenon doit être affichée. La nouvelle valeur doit être définie par le biais d'une macro VSTA.

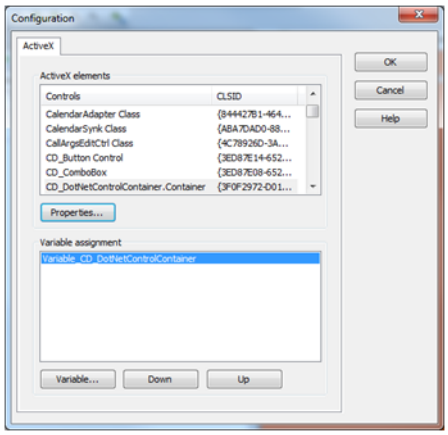


2. Améliorez le code avec une propriété (**Information**), et ajoutez à celle-ci les propriétés **get** et **set**. Elles vous permettent de lire et d'écrire le texte de l'intitulé.

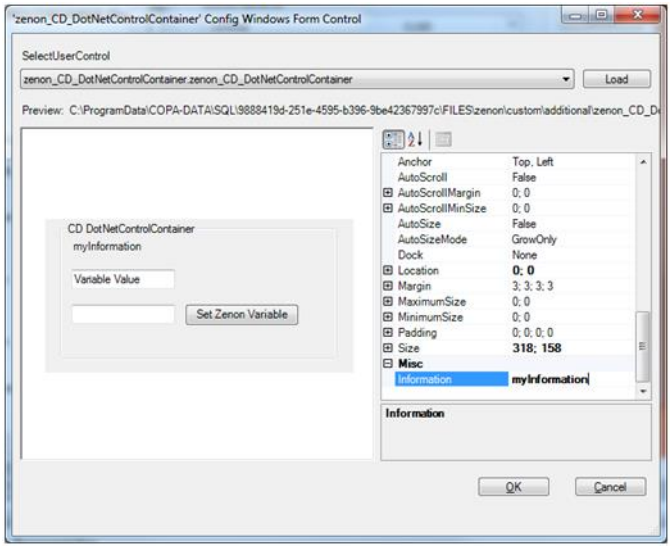
```
13  public partial class zenon_CD_DotNetControlContainer : UserControl
14  {
15      //This will be needed to get the zenon Variable Container
16      zenOn.Variable m_cVal = null;
17
18      public zenon_CD_DotNetControlContainer()
19      {
20          InitializeComponent();
21      }
22
23   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

3. Créez une nouvelle version du contrôle utilisateur et copiez-la vers le dossier *additional* du projet de zenon.
   N'oubliez pas : Fermez zenon Editor avant d'effectuer cette opération !
   Supprimez l'ancien fichier DLL, puis redémarrez zenon Editor. Si la DLL se trouve encore dans le dossier, supprimez-la simplement une deuxième fois. Vous pouvez maintenant importer la DLL modifiée. Les éléments **CD_DotNetContainerControl** et ActiveX sont mis à jour automatiquement.

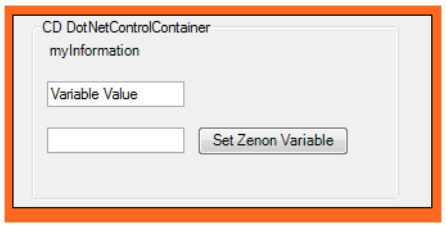
4. Dans zenon Editor, cliquez sur ActiveX et ouvrez la fenêtre de la propriété.



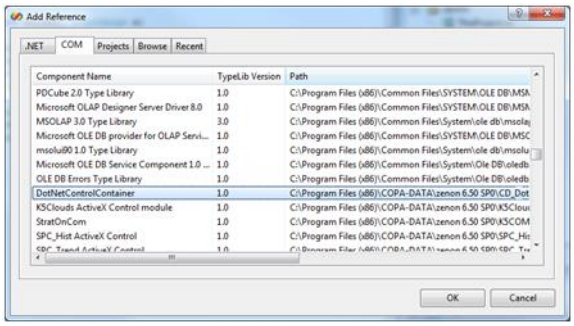
Vous pouvez maintenant vois la nouvelle propriété **Information** dans la fenêtre de sélection du contrôle, et vous pouvez également définir une valeur.



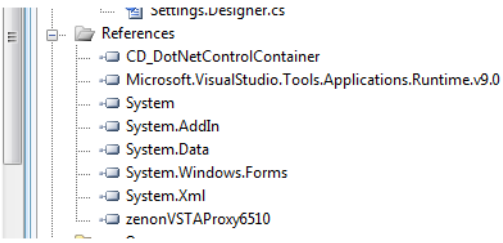
Cette valeur est également définie dans le contrôle ("myInformation").



5. Pour pouvoir utiliser **CD_DotNetControlContainer** dans VSTA ou VBA, vous devez d'abord disposer de la référence du contrôle. Après avoir ouvert VSTA pour le projet (**ProjectAddin**), vous devez ajouter la référence de **CD_DotNetControlContainer**.



Vous devez en outre ajouter l'Assembly **System.Windows.Forms**.



6. Le code suivant permet de redéfinir la valeur de la propriété **Information**.

```
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. Pour finir :
   - Créez une nouvelle fonction de zenon, intitulée **Exécuter macro VSTA**.
   - Liez la fonction à un bouton.

   Dans le Runtime, l'intitulé change de **myInformation** en **New Information** lorsque vous cliquez sur le bouton.

   Cliquez à nouveau sur le bouton pour effectuer le changement inverse.

5.3 Exemple : contrôle. NET exécuté en tant que contrôle ActiveX (C#)

L'exemple suivant décrit un contrôle .NET exécuté comme un contrôle ActiveX dans zenon.

La création et l'intégration se déroulent en quatre étapes :

1. Création d'un contrôle de formulaire Windows (à la page 31)
2. Conversion d'un contrôle utilisateur .NET en double contrôle (à la page 34)
3. Utilisation avec ActiveX dans Editor via le code VBA (à la page 38)

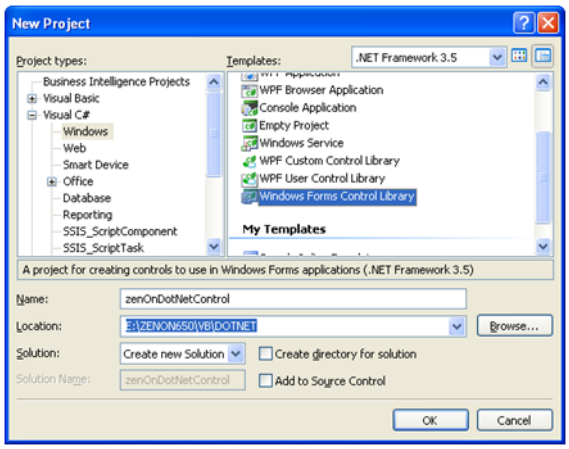
4. <Connexion de variables de CD_PRODUCTNAME> au contrôle utilisateur .NET (à la page 39)

> **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**
>
> Les captures d'écran de cette section sont uniquement disponibles en anglais.

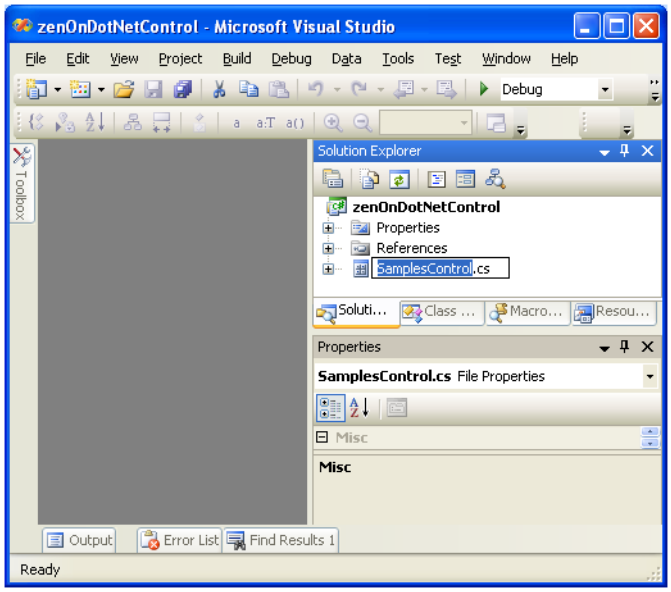
5.3.1 Création d'un contrôle de formulaire Windows

Pour créer un contrôle de formulaire Windows :

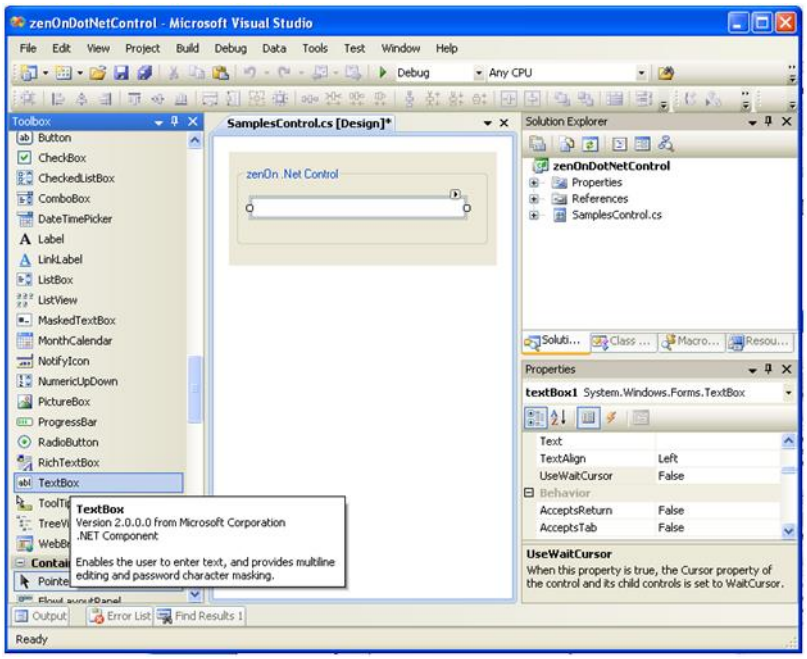
1. Démarrez Visual Studio 2008 et créez un nouveau projet **Windows Form Control Library** (Bibliothèque de contrôles de formulaire Windows) :



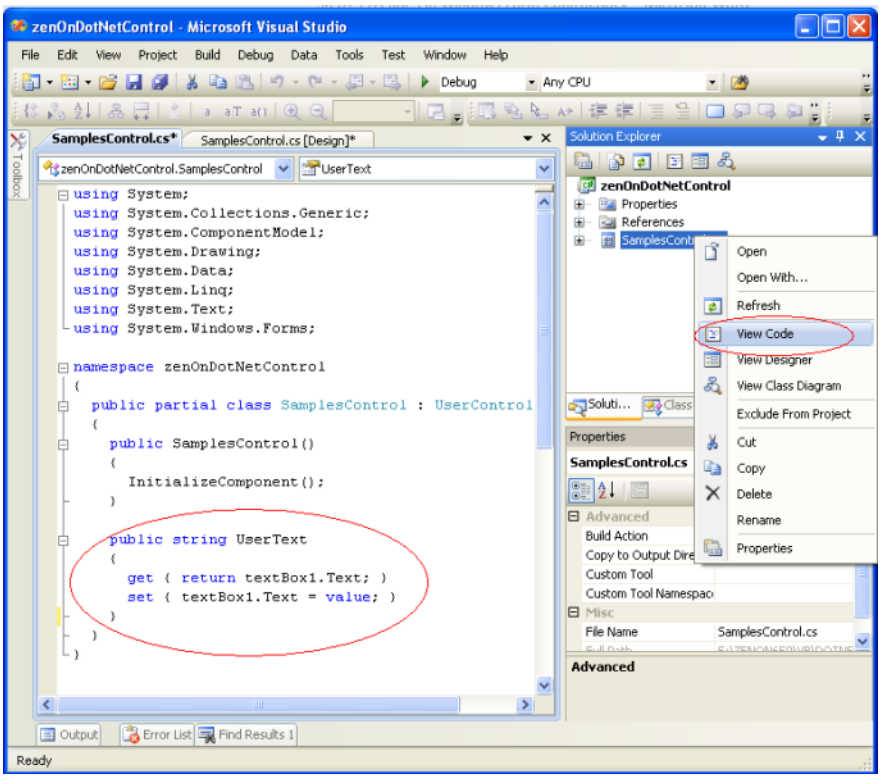
2. Renommez le contrôle par défaut avec le nom de contrôle souhaité.
   Pour cet exemple : **SampesControl.cs**.



3. Ouvrez le concepteur de contrôles et ajoutez le contrôle souhaité (dans notre cas, un champ de texte) :



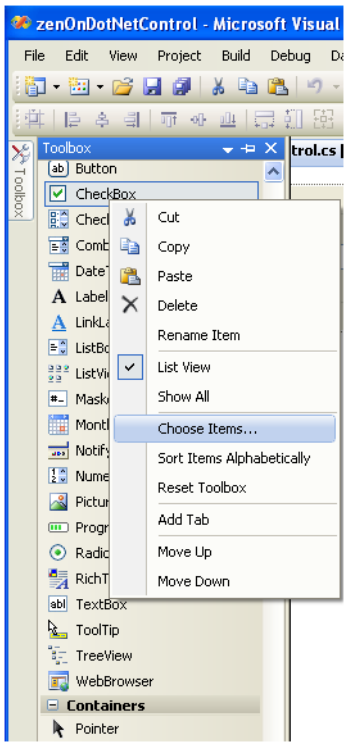
4. Les contrôles comportent normalement des propriétés. Ouvrez le concepteur de code en sélectionnant la fonction **Afficher le code**, puis ajoutez les propriétés devant que vous souhaitez rendre disponibles au niveau externe.
   Pour cet exemple : la propriété visible au niveau externe "**UserText**", avec l'accès **get** et **set**, qui contient le texte du champ de texte :



5. Compilez le projet.

   Le contrôle de formulaire Windows peut maintenant être utilisé dans d'autres projets de formulaires Windows.

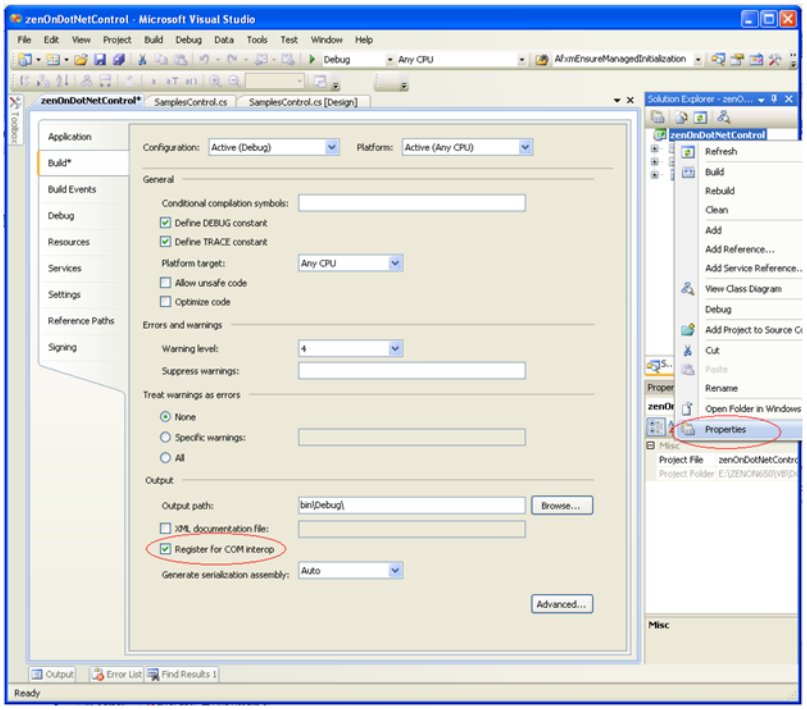
Important : le contrôle doit être inséré manuellement dans la boîte à outils de contrôle par le biais de la fonction **Choisir les éléments**.



5.3.2 Conversion d'un contrôle utilisateur .NET en double contrôle

Pour convertir un contrôle .NET en double contrôle, vous devez d'abord activer l'interface COM pour ActiveX.

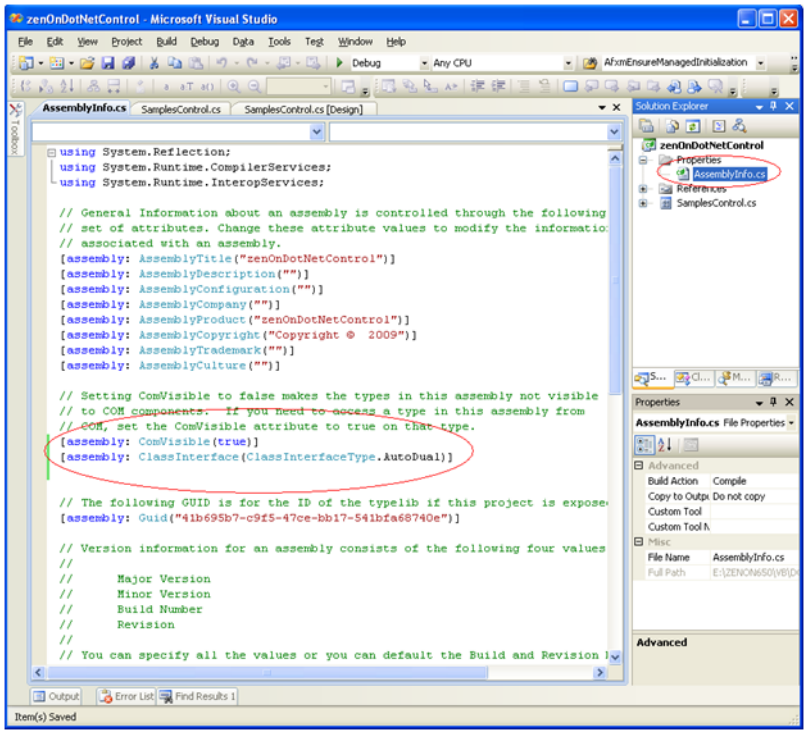
1. Ouvrez le projet et activez la propriété **Inscrire pour COM interop** dans les paramètres **Générer** :



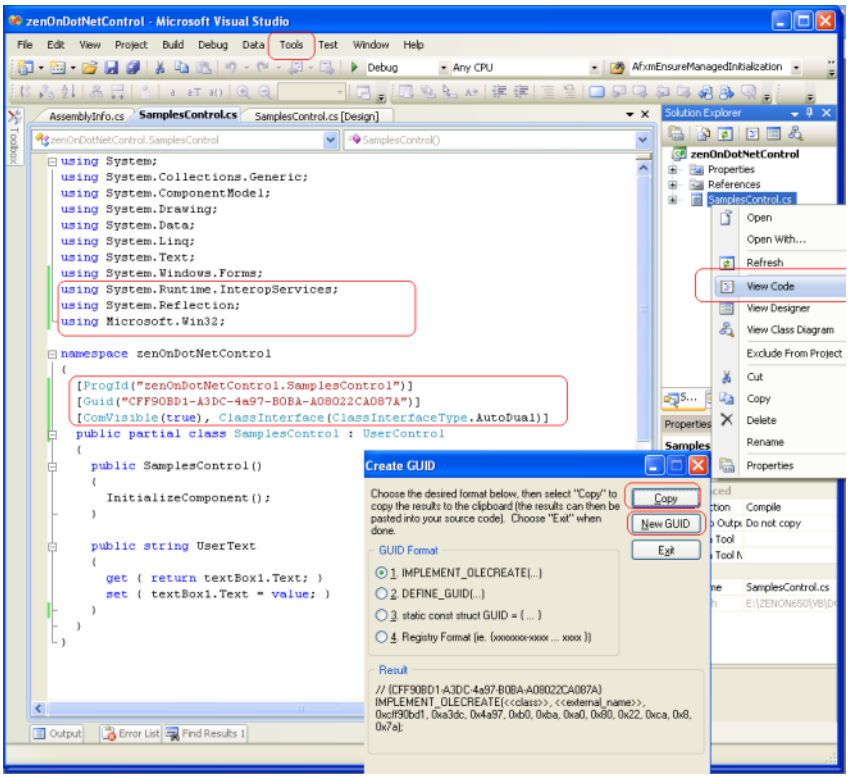
2. Ouvrez le fichier **AssemblyInfo.cs** et
   - définissez l'attribut **ComVisible** sur *true*
   - ajoutez l'attribut **ClassInterface**

   [ensemble : ComVisible(true)]

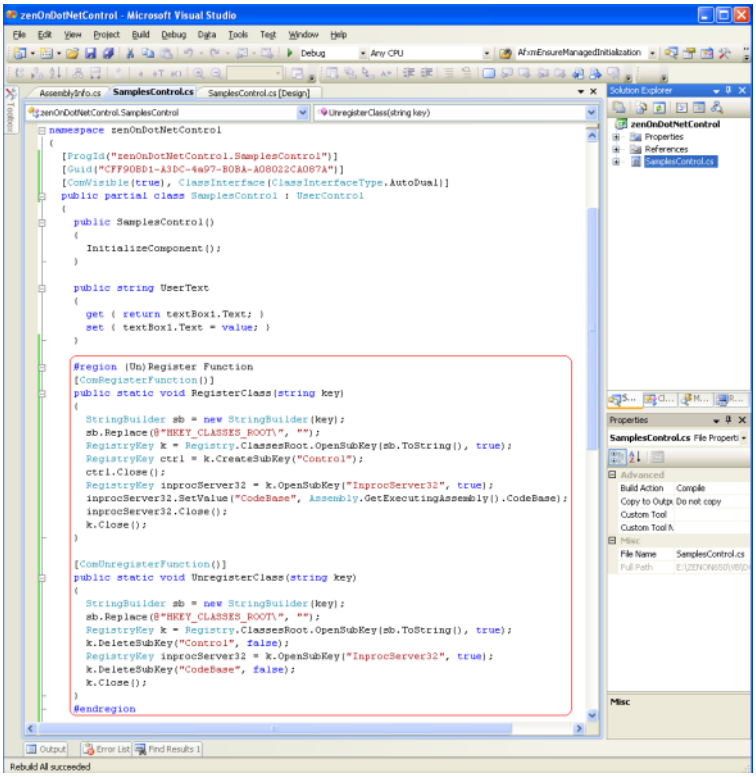
[ensemble : ClassInterface(ClassInterfaceType.AutoDual)]



3. Ouvrez le concepteur de code en sélectionnant l'option **Afficher le code**, puis ajoutez les attributs ActiveX et les entrées **using** requis. Dans le menu **Outils/Create GUID**, créez un nouveau GUID pour l'attribut GUID :



4. Pour que le contrôle puisse être sélectionné comme un contrôle d'interface utilisateur Active X, vous devez ajouter des fonctions aux classes de contrôle suivantes :
   - RegisterClass
   - UnregisterClass



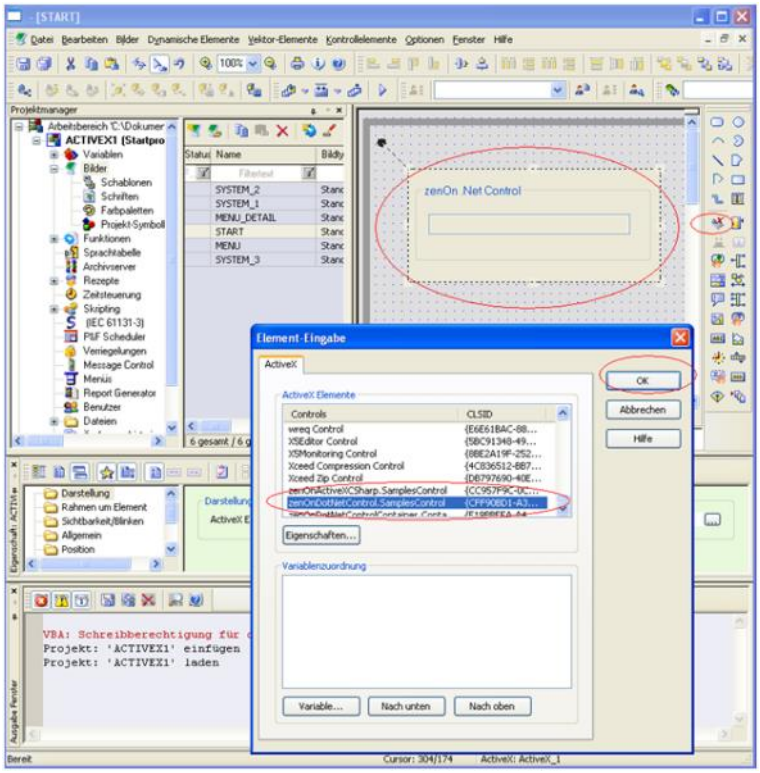
   Vous pouvez ensuite inscrire le contrôle dans la base de registres.

5. Recompilez le projet.

   Le contrôle de formulaire Windows est maintenant utilisable dans ActiveX, et a été inscrit automatiquement durant la création de la nouvelle version. Un fichier typelib supplémentaire, **zenonDotNetControl.tlb**, a été créé dans le répertoire de sortie.

6. Pour utiliser le contrôle sur un autre ordinateur :
   a) Copiez le fichier DLL et le fichier TLB sur l'ordinateur cible
   b) Enregistrez les fichiers en saisissant la ligne de commande :
      %windir%\Microsoft.NET\Framework\v2.0.50727\regasm.exe zenonDotNetControl.dll /tlb:zenonDotNetControl.tlb

7. Ajoutez le contrôle de formulaire étendu de Windows en tant que contrôle ActiveX dans zenon Editor :



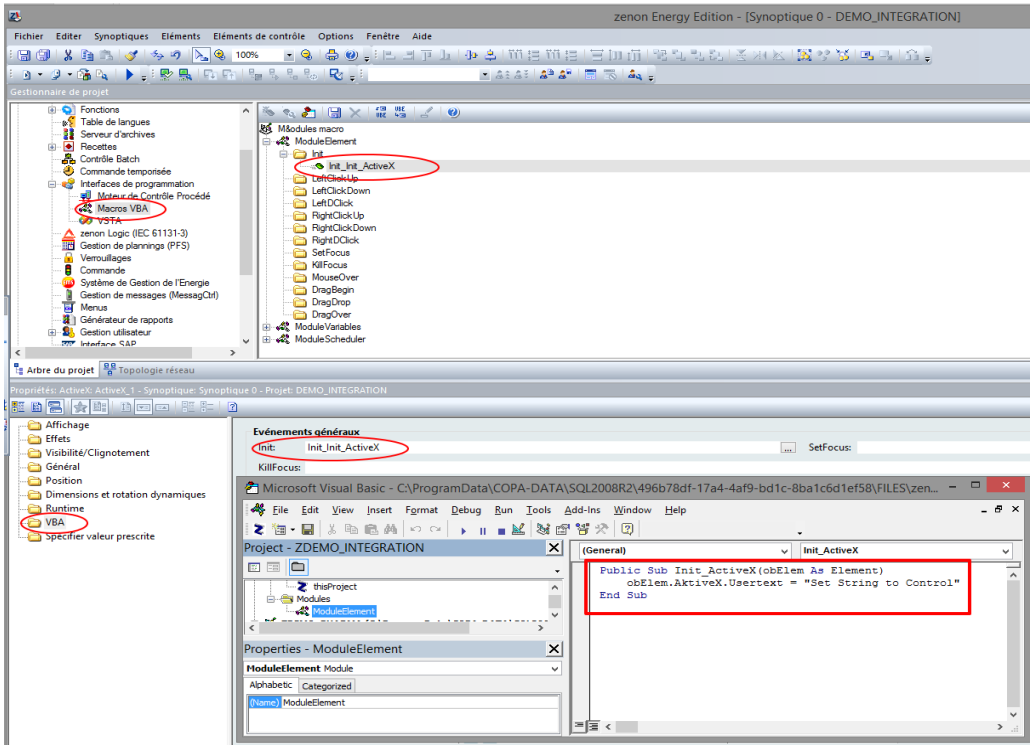
5.3.3 Utilisation avec ActiveX dans Editor via le code VBA

Pour accéder aux propriétés du contrôle dans zenon Editor :

1. Dans zenon Editor, dans le nœud **Interfaces de programmation/Macros VBA**, créez une nouvelle macro **Init** avec le nom **Init_ActiveX**.

   Dans cette macro, vous pouvez accéder à toutes les propriétés externes via **obElem.ActiveX**.

2. Attribuez cette macro au contrôle ActiveX via les propriétés **macros VBA/Init** de l'élément ActiveX.



EXEMPLE DE MACRO INIT

```
Public Sub Init_ActiveX(obElem As Element)
obElem.AktiveX.Usertext = "Attribuez la chaîne au contrôle"
End Sub
```

5.3.4 <Connexion de variables de CD_PRODUCTNAME> au contrôle utilisateur .NET

Dans zenon, vous avez la possibilité d'améliorer un contrôle ActiveX avec des fonctions spéciales pour accéder à l'API zenon.

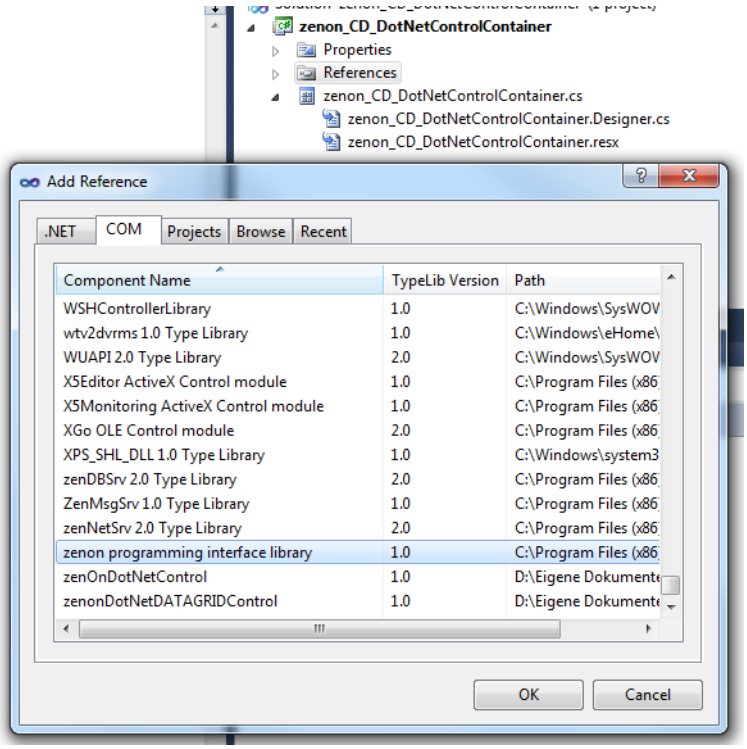
MÉTHODES REQUISES

- public bool zenonInit (à la page 41) (appelé durant l'initialisation du contrôle dans le Runtime de zenon.)
- public bool zenonInitED (à la page 41) (utilisé dans Editor.)
- public bool zenonExit() (à la page 42) (appelé lors de la destruction du contrôle dans le Runtime de zenon.)

- public bool zenonExitED() (à la page 42) (utilisé dans Editor.)
- public short CanUseVariables() (à la page 42) (prend en charge la liaison de variables.)
- public short VariableTypes() (à la page 42) (types de données pris en charge par le contrôle)
- public MaxVariables() (à la page 43) (nombre maximal de variables pouvant être liées au contrôle.)

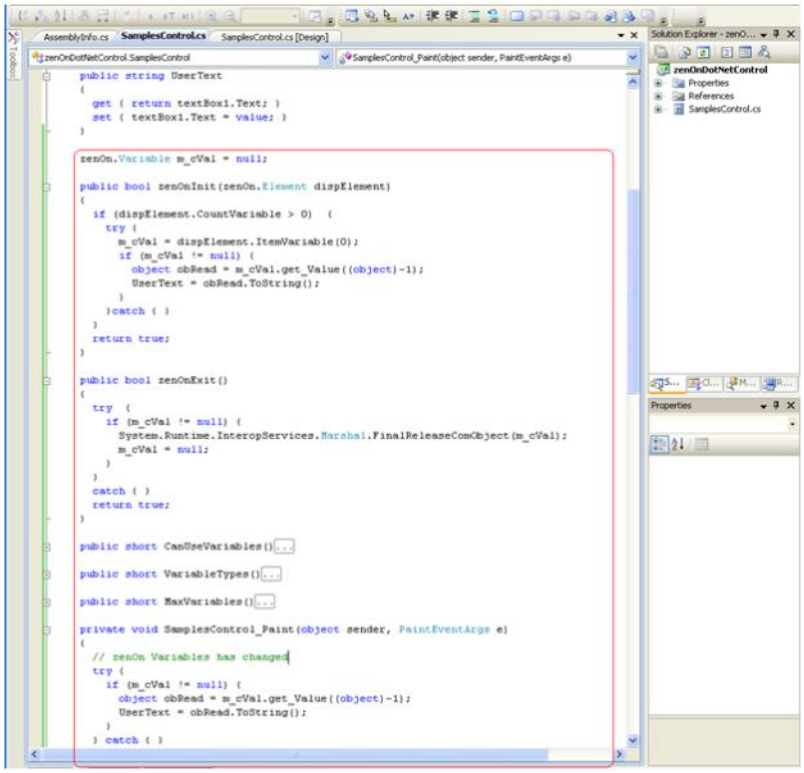
AJOUT DE RÉFÉRENCES

1. Dans Microsoft Visual Studio, sous **Ajouter des références**, sélectionnez la bibliothèque d'objets de zenon pour pouvoir accéder à l'API de zenon dans le contrôle.



2. Ajoutez les fonctions améliorées au code de classe du contrôle pour accéder à l'ensemble de l'API de zenon.

Dans notre exemple, l'objet COM d'une variable de zenon est temporairement enregistré dans un **membre**, afin d'être accessible ultérieurement via l'événement **Paint** du contrôle.



5.3.4.1 public bool zenOnInit(zenOn.Element dispElement)

Cette méthode (dans le Runtime) permet au contrôle ActiveX d'obtenir un pointeur vers l'interface de répartition de l'élément dynamique. Ce pointeur permet également d'accéder aux variables de zenon liées à l'élément dynamique.

Vous pouvez configurer l'ordre de transmission des variables dans la boîte de dialogue Entrer les propriétés, à l'aides des boutons **Bas** et **Haut**. La boîte de dialogue Entrer les propriétés s'affiche si :

- Vous double-cliquez sur l'élément ActiveX, ou
- Vous sélectionnez **Propriétés** dans le menu contextuel, ou
- Vous sélectionnez la propriété **Paramètres ActiveX** dans le nœud **Affichage** de la fenêtre de propriétés

5.3.4.2 public bool zenonInitED(zenon.Element dispElement)

Équivalent à public bool zenonInit (à la page 41) ; exécuté lors de l'ouverture d'ActiveX dans Editor (double-cliquez sur le contrôle ActiveX).

5.3.4.3 public bool zenOnExit()

Cette méthode est appelée par le Runtime de zenon lorsque le contrôle ActiveX est fermé. Ici, tous les pointeurs de répartition renvoyant à des variables doivent être libérés.

5.3.4.4 public bool zenOnExitED()

Equals public bool zenOnExit() (à la page 42) 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()

Cette méthode renvoie *1* si le contrôle peut utiliser les variables de zenon, et *0* dans le cas contraire.

- *1* : Pour l'élément dynamique (via le bouton **Variable**), vous pouvez uniquement déclarer des variables de zenon avec le type déclaré par le biais de la méthode VariableTypes, conformément au nombre défini par la méthode MaxVariables.
- *0* : si CanUseVariables renvoie *0* ou le contrôle ne comporte pas cette méthode, n'importe quel nombre de variables de tout type peut être défini, sans limitation aucune. Dans le Runtime, toutefois, ces variables sont uniquement utilisables avec du code 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()

Dans cette section est défini le nombre de variables pouvant être sélectionnées dans la liste de variables.

1 : la sélection multiple est désactivée dans la liste de variables. Un avertissement est alors affiché si plusieurs variables sont sélectionnées.

6 WPF

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

Une document détaillée concernant les éléments WPF dans zenon, ainsi que des tutoriels destinés aux concepteurs, développeurs et utilisateurs configurant des projets sont disponibles dans le manuel consacré aux éléments WPF dans zenon.

Ce manuel aborde les sujets suivants :

- Notions fondamentales
- Directives pour les concepteurs
- Directives pour développeurs
- Développement dans zenon