Intégrer des Winforms .Net dans Visual FoxPro

3M
3925 South 700 West #22
Murray, UT 84123
Voice: 801-699-8782
www.craigberntson.com
Email: craig@craigberntson.com

Résumé

Cet article vous propose un aperçu de l'Interop Forms Toolkit ; cet outil est fourni par l'équipe VB de Microsoft pour faciliter l'intégration des Winforms dans un client COM tel que Visual FoxPro. Cet outil contient un assistant pour rendre les Winforms et les contrôles compatibles COM, afin de pouvoir les déposer sur des forms VFP.

Pourquoi l'Interopérabilité

Depuis l'annonce de l'arrêt du développement de Visual FoxPro, les entreprises doivent faire un choix, quant à la façon dont elles migreront leurs applications sur une autre plateforme. Toutes les options peuvent être défendues par des arguments corrects, que ce soit continuer d'utiliser une application VFP et développer de nouvelles fonctionnalités en VFP, migrer en une seule fois la totalité de l'application sur une autre technologie, ou bien migrer une application par morceaux. Cet article se réfère à la dernière option, la migration d'une application vers VB, par petits bouts.

Heureusement, l'équipe VB de Microsoft a développé un outil qui va nous aider. Au départ, cet outil était conçu pour aider les développeurs VB6 à passer à .Net, mais nous en profiterons, puisqu'il fonctionne pour beaucoup d'autres langages compatibles COM, y compris Visual FoxPro.

Oui, mais si vous avez choisi C# comme langage .Net, vous allez immédiatement demander pourquoi cet outil ne fonctionne pas en C#? Eh bien, comme C# a été développé spécifiquement pour .Net, il n'y a pas de langage préexistant qui nécessite d'intéropérer avec C#. C'est l'explication donnée par Microsoft. Vous pouvez vous servir de cet outil pour apprendre ce qu'il vous faut coder vous-même et continuer d'utiliser C#. Vous trouverez également sur le web des modèles de projets proposés par des tierces parties ; voici par exemple un Interop Toolkit en C# sur http://www.codeproject.com/KB/vb-interop/VB6InteropToolkit2.aspx.

L'Interop Toolkit

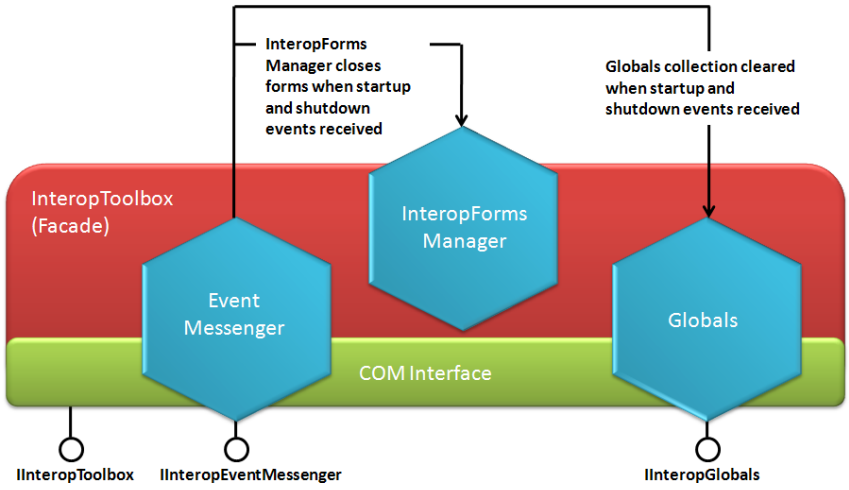
La première chose à faire est de télécharger l'Interop Forms Toolkit sur http://msdn.microsoft.com/en-us/vbasic/bb419144.aspx. Sur mon PC sous Vista, j'ai eu des problèmes en essayant de lancer le fichier .msi, il vaut mieux prendre aussi le Setup.exe (et bien sur le fichier de runtime .msi pour vous distributions). Il faut également Visual Studio. Attention, la version Express n'accepte pas les compléments, il vous faut une autre version. Le setup installe l'assistant dans Visual Studio, et ajoute plusieurs applications d'exemple, les fichiers d'aide et d'autres fichiers, dans le dossier C:\Program Files\Microsoft Visual Basic 2005 Power Packs\Interop Forms Toolkit 2.0. Les exemples dont en VB6 et en VB.Net. Toutefois dans cet article, je les reproduirai et les développerai en VFP.

L'installation ajoute aussi Microsoft.InteropFormTools comme seul assembly dans le Global Assembly Cache (le GAC). Vous y accéderez depuis une application VB.NET par le NameSpace My.InteropToolbox.

Les 3 parties principales de l'InteropToolBox sont :

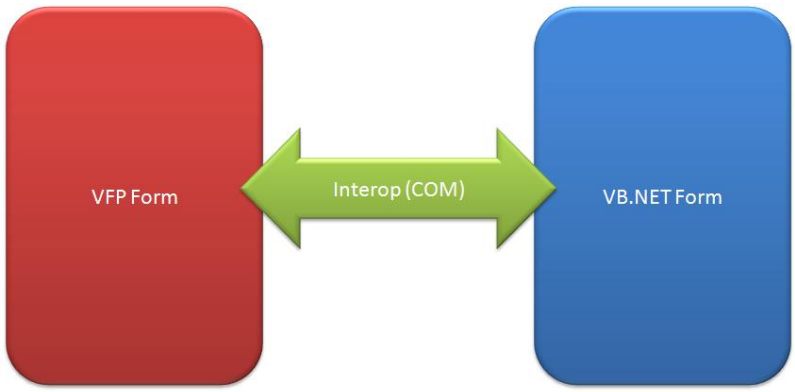
- Event Messenger – qui sert à déclencher et à recevoir les notifications d'événements au niveau de l'application
- InteropFormManager – qui conserve la collection de tous les forms Interop créés
- Globals – un objet collection pour partager les variables « globales » entre VFP et .NET.

Le diagramme ci-dessous montre ces 3 parties.



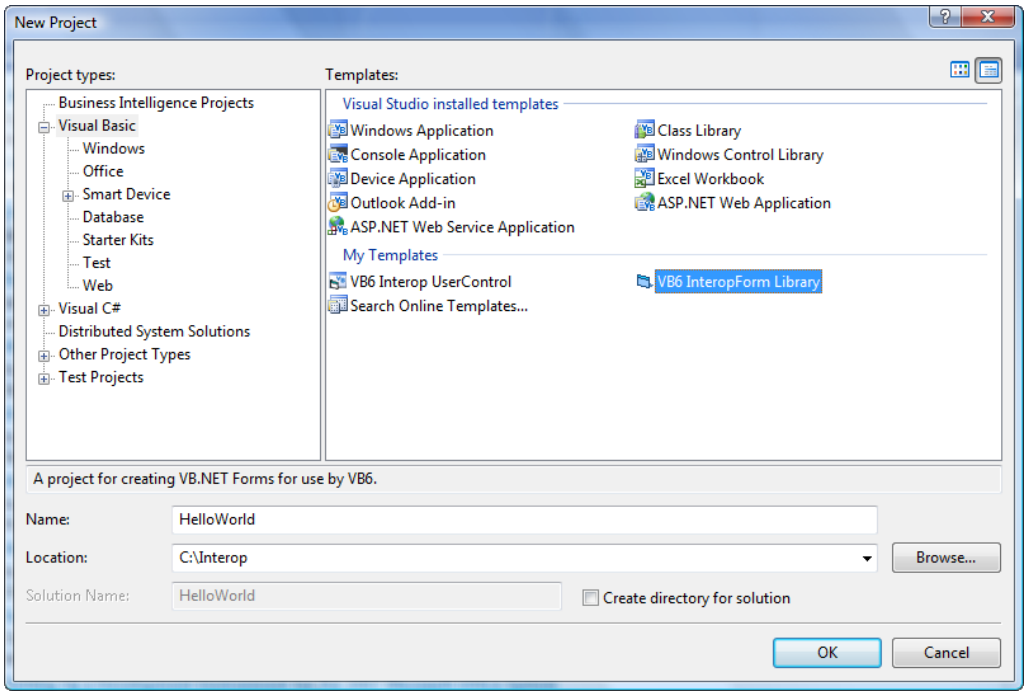
Pour Commencer

Comme tous les bons exemples de programmation débutent par une petite application « Hello World », faisons de même ici. Nous allons donc avoir 2 forms simples, l'un en VFP et l'autre en VB.Net, qui communiqueront entre eux par l'intermédiaire d'un COM Intérop.



Voyons donc le code.

1. Démarrez Visual Studio et créez un nouveau projet VB de Bibliothèque InteropForm ; appelez-le HelloWorld



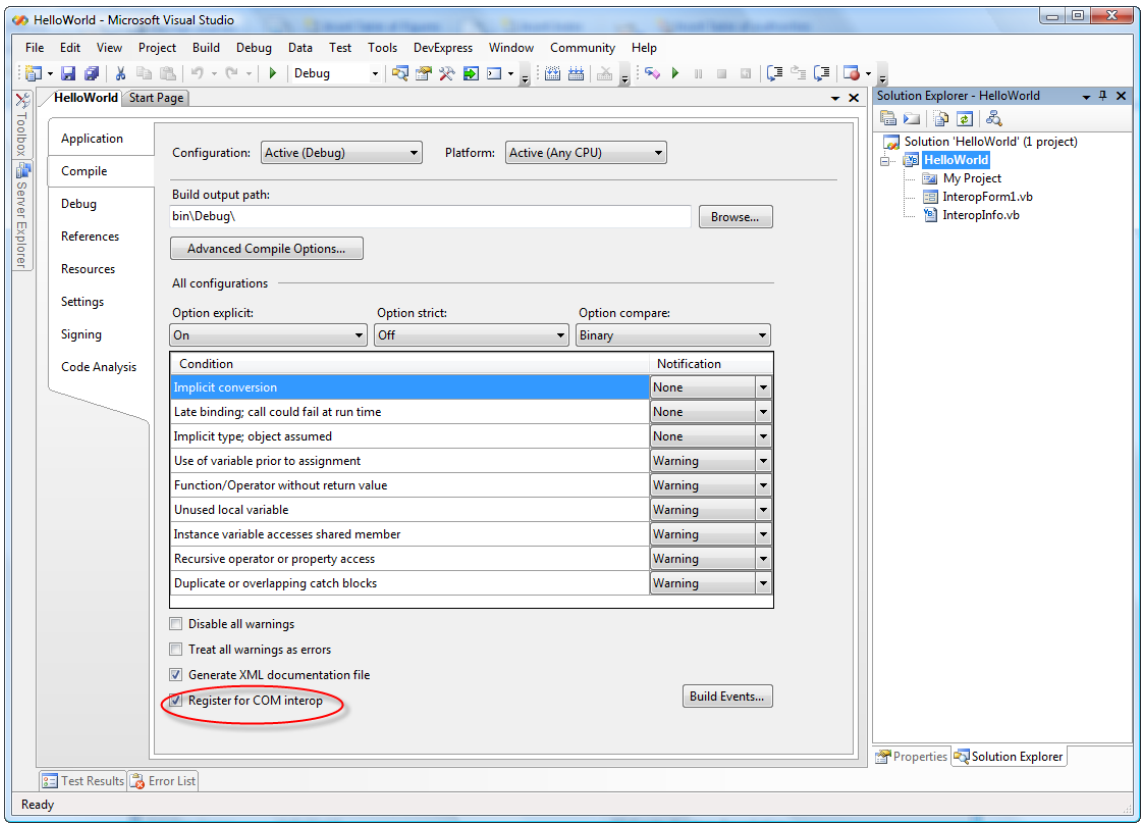
2. Le modèle ajoute une référence à la dll Microsoft.InteropFormsTools et crée deux fichiers, InteropForm1.vb et InteropInfo.vb. Regardons le contenu de InteropInfo.vb, étant donné que le form ne contient rien d'intéressant.

```
Imports Microsoft.InteropFormTools

' This ComVisible attribute is placed outside of the
' AssemblyInfo.vb file since the Project Properties
' Designer will change the attribute in that file when
' the "Register For COM Interop" option is checked.
' If the attribute is added to that AssemblyInfo.vb
' it will cause a compile error so simply delete it.
<Assembly: System.Runtime.InteropServices.ComVisible(False)>
```

```
' This property is included in the My namespace, so it will show up in IntelliSense after referencing
My.
Namespace My
    ' The HideModuleNameAttribute hides the module name MyInteropToolbox so the syntax becomes
My.InteropToolbox.
    <Global.Microsoft.VisualBasic.HideModuleName()> _
    Module MyInteropToolbox

        Private _toolbox As New InteropToolbox

        Public ReadOnly Property InteropToolbox() As InteropToolbox
            Get
                Return _toolbox
            End Get
        End Property
    End Module
End Namespace
```

Ce code importe l'espace de nom Microsoft.InteropFormTools et crée une nouvelle propriété nommée InteropToolbox. Le code utilise également un attribut qui masque le nom du module, pour simplifier l'utilisation de la syntaxe.

Regardez les propriétés du projet : dans l'onglet « Compiler », vous constatez que le projet est d'ores et déjà configuré pour COM Interop.



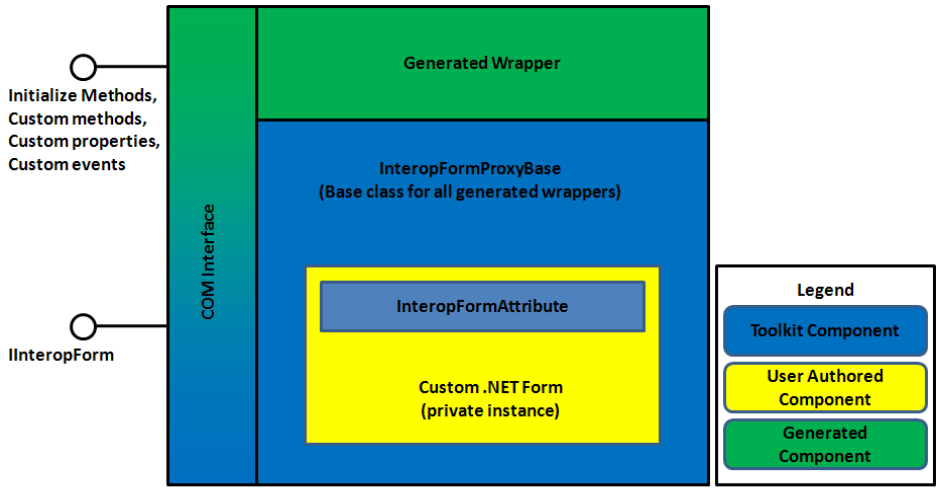
3. Dans l'explorateur de solutions, cliquez droit sur le form et renommez-le HelloWorldForm.vb.
4. Double-cliquez sur le form dans l'Explorateur de solutions.
5. Définissez le Orange comme backcolor, et « Bonjour .Net » pour la propriété Text.
6. Déposez une étiquette (label) sur le form
7. Définissez son nom comme lblHelloText, son backcolor à Cyan, la FontSize à 24, et la propriété Text à « Bonjour depuis VB.Net ».
8. Déposez un LinkLabel sur le form.
9. Réglez sa FontSize à 16 et définissez sa propriété Text comme « Cliquez ici pour déclencher l'évènement ».

10. Double-Cliquez sur le LinkLabel sur le form pour ouvrir la fenêtre de code ; modifiez ce code pour y ajouter un appel à RaiseEvent.

```
Imports Microsoft.InteropFormTools

<InteropForm()> _
Public Class HelloWorldForm

   <InteropFormEvent()> _
   Public Event SampleEvent As SampleEventHandler
   Public Delegate Sub SampleEventHandler(ByVal sampleEventText As String)

   Private Sub LinkLabel1_LinkClicked(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.LinkLabelLinkClickedEventArgs) Handles LinkLabel1.LinkClicked

      RaiseEvent SampleEvent(String.Format("Exemple de Texte d'Evenement", Me.lblHelloText.Text))

   End Sub
End Class
```

Le code commence par importer l'espace de nom InteropFormTools. L'attribut InteropForm() va permettre de générer une classe d'empaquetage visible comme COM. L'attribut InteropFormEvent() sert à définir un évènement COM visible, et enfin le gestionnaire d'événement est define pour être déclenché quand on clique sur le LinkLabel.

11. Dans le menu, choisissez « Outils » → Generate InteropForm Wrapper Class. Le composant additionnel du ToolKit va créer un nouveau dossier contenant un fichier un seul fichier, HelloWorldForm.wrapper.vb. Le schéma ci-dessous montre visuellement ce que fabrique l'exécution de ce composant.



12. Voici le code que contient ce nouveau fichier.

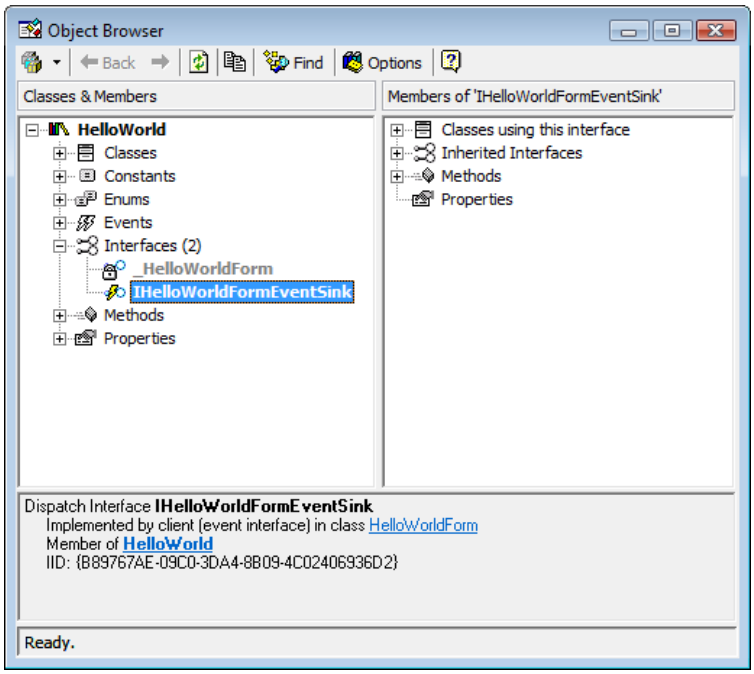
```
'------------------------------------------------------------------------------
' <auto-generated>
'     Ce code a été généré par un outil.
'     Version du runtime:2.0.50727.1378
'
'    Les modifications apportées à ce fichier peuvent provoquer un comportement incorrect et
'    seront perdues si
'    le code est régénéré.
' </auto-generated>
'------------------------------------------------------------------------------

Option Strict Off
Option Explicit On

Imports Microsoft.InteropFormTools

Namespace Interop
```

```
<System.Runtime.InteropServices.ClassInterface(Runtime.InteropServices.ClassInterfaceType.AutoDual),
System.Runtime.InteropServices.ComVisible(true)>  _
    Partial Public Class HelloWorldForm
        Inherits InteropFormProxyBase

        Public Sub New()
            MyBase.New
            FormInstance = New HelloWorld.HelloWorldForm()
            RegisterFormInstance()
        End Sub
    End Class
End Namespace
```

13. Compilez la solution, ceci termine le form VB.Net.

Nous avons fini de coder en VB.NET, et nous pouvons maintenant passer à Visual FoxPro pour voir comment appeler ce form VB.Net depuis VFP.

1. Démarrez Visual FoxPro, et créez un nouveau projet que vous appellerez HelloWorld.
2. Depuis le menu de VFP, lancez l'explorateur d'objets par Outils → Explorateur d'Objets.
3. Cliquez sur le bouton Open Type Library de l'explorateur d'objets.
4. Choisissez l'onglet COM Libraries dans la boite de dialogue Ouvrir.
5. Cliquez sur le bouton Browse.
6. Naviguez jusqu'à l'emplacement du fichier HelloWorld.tlb. Sur ma machine, il se trouve dans le dossier \BIN\DEBUG qui est dans le dossier du projet .Net. L'objet COM HelloWorld est alors affiché dans l'arborescence.



7. Développez le nœud Interfaces dans l'arborescence.
8. Ajoutez un nouveau fichier de programme au projet.
9. Faîtes glisser le IHelloWorldFormEventSink et déposez-le sur le nouveau fichier de programme. Vous allez ainsi créer un gestionnaire d'événement dans VFP. En cliquant sur le LinkLabel du form VB.Net, on déclenchera un événement qui sera intercepté ce code VFP.

```
x=NEWOBJECT("myclass")

DEFINE CLASS myclass AS session OLEPUBLIC

   IMPLEMENTS IHelloWorldFormEventSink IN "C:\INTEROP\HELLOWORLD\BIN\DEBUG\HELLOWORLD.TLB"
```

```
    PROCEDURE IHelloWorldFormEventSink_SampleEvent(sampleEventText AS STRING) AS VOID
    * add user code here
    ENDPROC

ENDDEFINE
```

10. Renommez la classe `myclass` en `cusEventHandler`.
11. Remplacez le commentaire `* add user code here` par

```
MESSAGEBOX(sampleEventText)
```

    Le déclenchement de l'événement sur le form VB.Net va passer une chaine de caractère au gestionnaire d'événement VFP, qui va alors l'afficher dans un MESSAGEBOX.

12. Enregistrez ce fichier sous le nom InteropEventHandler.prg, et fermez-le.
13. Fermez l'Explorateur d'Objets.
14. Ajoutez une nouvelle classe cusInteropToolkit basée sur la classe Custom, que vous mettrez dans une bibliothèque InteropToolkit.
15. Ajoutez lui une propriété, que vous appellerez oNetInteropToolbox.
16. Ajoutez le code suivant à la method Init :

```
WITH This
   .oNetInteropToolbox = CREATEOBJECT("Microsoft.InteropFormTools.InteropToolbox")
   .oNetInteropToolbox.Initialize()
   .oNetInteropToolbox.EventMessenger.RaiseApplicationStartedupEvent()
ENDWITH
```

    L' InteropToolbox est un composant installé par le ToolKit. Le lancement de ce code initialise le process d'interopérabilité, et déclenche l'événement d'interop signalant que l'application a démarré.

17. Ajoutez le code suivant à la méthode Destroy :

```
This.oNetInteropToolbox.EventMessenger.RaiseApplicationShutdownEvent()
```

18. Enregistrez la classe, fermez-la.
19. Ajoutez un nouveau form au projet.
20. Ajoutez lui 2 propriétés que vous appellerez oEventHandler et oInteropForm.
21. Définissez sa propriété WindowType à Modal.
22. Ajoutez le code suivant à la method Init :

```
* Instantiate the Event Handler object
This.oEventHandler = NEWOBJECT("cusEventHandler", "InteropEventHandler.prg")
This.oInteropForm = CREATEOBJECT("HelloWorld.Interop.HelloWorldForm")
EVENTHANDLER(This.oInteropForm, This.oEventHandler)
```

    Le gestionnaire d'événement est instancié, puis le form VB.Net est créé. Toutefois, le form VB.Net n'est pas affiché.

23. Ajoutez le code suivant à la méthode Destroy, de façon à déconnecter le gestionnaire d'événement à la fermeture du form.

```
EVENTHANDLER(This.oInteropForm, This.oEventHandler, .T.)
```

24. Ajoutez un CommandButton sur le form, et redimensionnez-le de telle sorte qu'il recouvre presque tout le form.
25. Définissez le CommandButton.Caption à « Lancement du Form .Net ».
26. Ajoutez ce qui suit à l'événement click de ce bouton :

```
ThisForm.oInteropForm.Show()
```

    En cliquant sur le bouton du form VFP, on affichera le form VB.Net.

27. Enregistrez ce form sous le nom de HelloWorld, et fermez-le.
28. Ajoutez un nouveau fichier de Programme qui contiendra le code suivant :

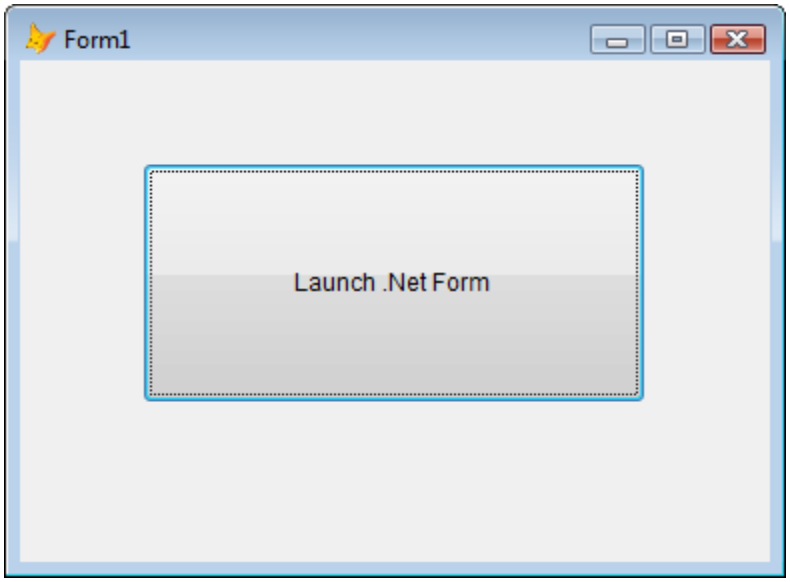
```
* Hello World .Net Forms Interop
```

```
* Initialize the Forms Interop Toolbox
PRIVATE poInteropToolbox
m.poInteropToolbox = NEWOBJECT("cusInteropToolkit", "InteropToolkit")

* Launch the VFP form
DO FORM HelloWorld

* Destroy the Forms Interop Toolbox
m.poInteropToolbox = NULL
```

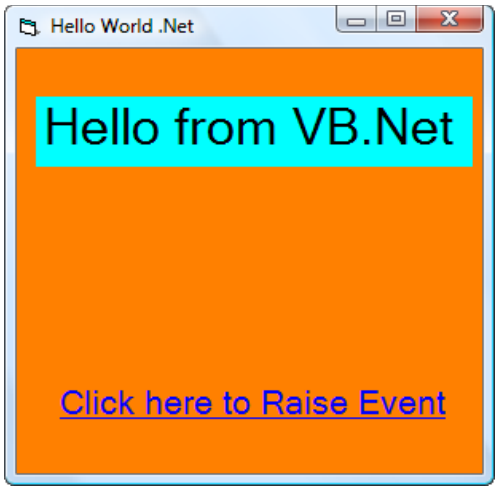
29. Enregistrez ce fichier en tant que Main.prg, et fixez l'origine de votre projet sur ce fichier dans le Gestionnaire de Projet.
30. Générez le projet.

Le code est fini, il ne reste plus qu'à le lancer pour voir comment ça fonctionne.

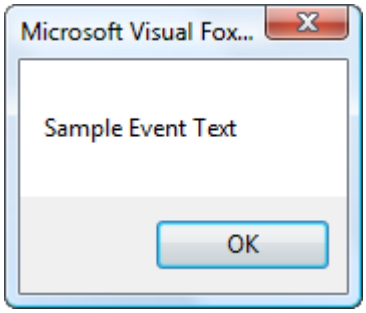
1. Assure-vous que vous pointez bien sur le main.prg dans le Gestionnaire de Projet VFP.
2. Cliquez sur Exécuter ; le form VFP apparaît.



3. Cliquez sur le CommandButton du form. Le form VB.Net apparaît.



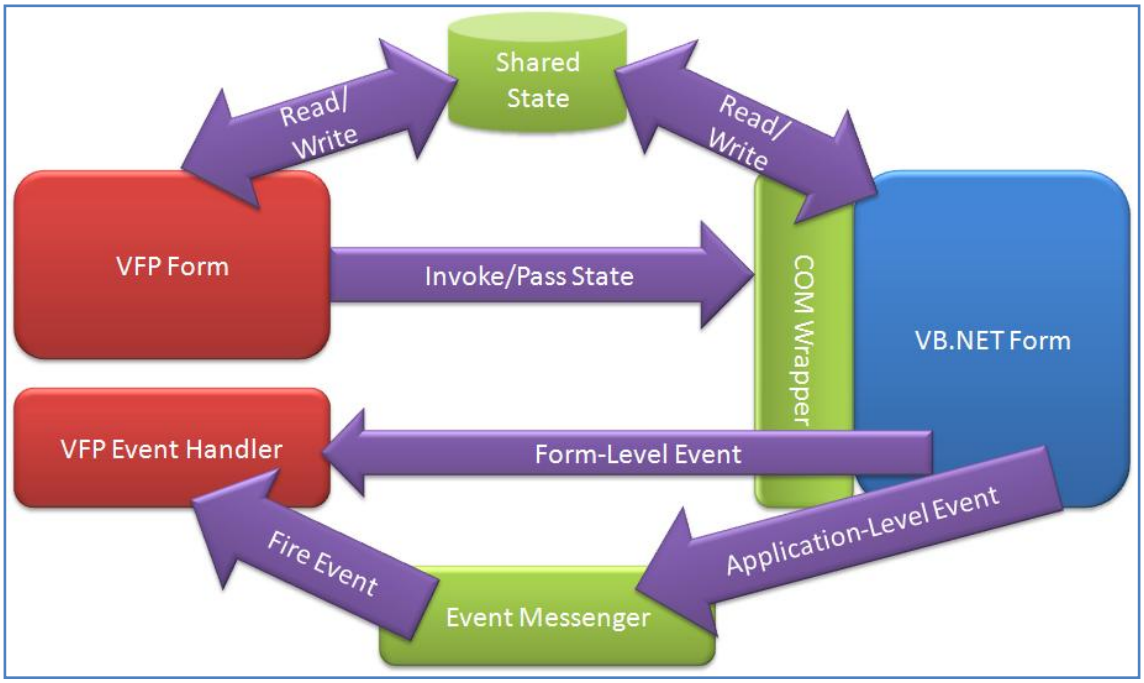
4. Cliquez sur le LinkLabel (là où vous avez le texte « Cliquez ici pour déclencher l'événement ») sur le form VB.Net ; vous déclenchez bien un événement et VFP affiche un MessageBox.



5. Fermez le MessageBox et le Form VFP. Remarquez que le form VB.Net est automatiquement fermé par l'appel du RaiseApplicationShutDownEvent dans la classe VFP cusInteropToolkit.

Nous avons pas mal avancé, prenons le temps de résumer ce que nous avons fait. En premier lieu, nous avons créé un form VB.Net qui possède les attributs d'interopérabilité. En lançant le composant additionnel de Visual Studio, nous l'avons empaqueté dans un objet COM et mis dans une dll .Net.

Nous avons alors instancié dans Visual FoxPro une classe .Net pour gérer la communication interop. VFP instancie et affiche le form .Net, et capture un événement déclenché par .Net. L'application VFP gère entièrement le process.



Un exemple plus complexe

Ce qui précède était une application très simple, donc essayons maintenant quelque chose de plus compliqué, qui transmettrait des données entre VFP et .Net. Les deux plateformes VFP et .Net peuvent utiliser des jeux d'enregistrements ADO, qui sont de vrais objets COM. Le passage des données serait plus simple si nous construisions une application avec ADO. Oui, mais… la plupart des développeurs VFP ne se servent pas d'ADO, mais des curseurs et vues VFP, que .Net ne peut pas utiliser.

Une deuxième option possible serait de créer un paramètre objet basé sur la classe Empty, et de le passer à .Net. Il faudrait alors traiter en plus le problème de .Net qui ne peut pas voir les types de données, tout devenant un objet en .Net ; il nous faudrait nous servir de la Reflection pour aller jusqu'aux données.

Dans une troisième option, nous pouvons construire un composant COM en .Net, l'instancier dans VFP, puis le passer aux composants .Net; mais là encore, ça n'est pas spontané pour les développeurs VFP. Cela dit, ça marche !

Note	Si vous désirez plus d'information sur la façon de passer des objets entre VFP et .NET, lisez l'article de Rick Strahl « Comment passer des objets entre FoxPro et des Composants COM .Net » sur http://www.west-wind.com/presentations/dotnetfromVfp/DotNetFromVfp_ComplexObjects.asp

XML semble être un bon choix : aussi bien VFP que .Net peuvent créer et consommer du XML. On peut utiliser `CURSORTOXML()` et passer les données à .Net, qui à son tour les utilisera dans un DataSet. Mais c'est au retour que les problèmes surgissent, quand on aura sérialisé les données dans du XML et qu'on les renvoie à VFP : on ne peut pas se servir de `XMLTOCURSOR()` pour utiliser les valeurs retournées, et les classes XML ne sont pas non plus ici d'un grand secours.

Nous voilà face à un dilemme enfermé dans une énigme, elle-même au milieu d'un sac de nœuds ! Il semblerait que n'existe aucune solution simple pour récupérer des données venant d'un form .Net. Je persiste à penser que le RecordSet ADO serait la meilleure solution. Après quelques recherches sur Internet, j'ai trouvé deux articles dans la Base de connaissances Microsoft (Q192692 et Q192762), qui donnent du code permettant la conversion d'un cursor en recordset et vice-versa. J'y ai apporté quelques modifications, et je vous le fournis ici avec ce téléchargement.

Voici le code pour convertir un Cursor en RecordSet

```
* CursorToRS
* Adapted from Microsoft KB Q192762

#INCLUDE "adovfp.h"

LPARAMETERS tcCursor, tcScope, tlAddRecNoColumn, tlAddVariantColumn

LOCAL lcScope, lnDataTypeEnum, loRS
LOCAL lvValue, lnFieldCount, lcField, lcFieldType, lnFieldSize, lnFieldAttributes
LOCAL lnItemCount, lnCount, lnMatchCount, lnLastSelect, lnLastRecNo
LOCAL laFields[1], laItems[1], laValues[1]

lcScope = IIF(EMPTY(tcScope), "ALL", ALLTRIM(tcScope))

lnLastSelect = SELECT()
SELECT (tcCursor)
lnLastRecNo = IIF(EOF(), 0, RECNO())

loRS = NEWOBJECT("ADODB.Recordset")
loRS.cursorLocation = ADUSECLIENT
loRS.cursorType = ADOPENSTATIC
loRS.lockType = ADLOCKOPTIMISTIC

lnFieldCount = AFIELDS(laFields)
lnItemCount = 0

IF tlAddRecNoColumn
  lcField = "_RecNo"
  lnItemCount = 1
  laItems[1] = lcField
  loRS.Fields.Append(lcField, ADINTEGER, 8)
ENDIF

FOR lnCount = 1 TO lnFieldCount
  lcField = laFields[lnCount, 1]
  lcFieldType = laFields[lnCount, 2]
  IF lcFieldType == "G"
    LOOP
  ENDIF

  lnFieldSize = laFields[lnCount, 3]
  lnFieldAttributes = ADFLDFIXED + ADFLDUPDATABLE

  IF laFields[lnCount, 5]
    lnFieldAttributes = lnFieldAttributes + ADFLDISNULLABLE
  ENDIF

  DO CASE
    CASE lcFieldType == "C"
      lnDataTypeEnum = ADCHAR

    CASE lcFieldType == "M"
      lnDataTypeEnum = ADCHAR
      lnFieldSize = 256
```

```
    CASE lcFieldType == "L"
      lnDataTypeEnum = ADBOOLEAN

    CASE lcFieldType == "D"
      lnDataTypeEnum = ADDBDATE

    CASE lcFieldType == "T"
      lnDataTypeEnum = ADDBTIMESTAMP
      lnFieldSize = 6

    CASE lcFieldType == "N"
      lnDataTypeEnum = ADSINGLE

    CASE lcFieldType == "B"
      lnDataTypeEnum = ADDOUBLE

    CASE lcFieldType == "I"
      lnDataTypeEnum = ADINTEGER

    CASE lcFieldType == "F"
      lnDataTypeEnum = ADDOUBLE

    CASE lcFieldType == "Y"
      lnDataTypeEnum = ADCURRENCY

    OTHERWISE
      LOOP
  ENDCASE

  lnItemCount = lnItemCount + 1
  DIMENSION laItems[lnItemCount]
  laItems[lnItemCount] = lcField
  loRS.Fields.Append(lcField, lnDataTypeEnum, lnFieldSize, lnFieldAttributes)
ENDFOR

IF tlAddVariantColumn
  lcField = "_Variant"
  lnItemCount = lnItemCount+1
  DIMENSION laItems[lnItemCount]
  laItems[lnItemCount] = lcField
  loRS.Fields.Append(lcField, ADVARIANT, 1)
ENDIF

IF lnItemCount <= 1
  RETURN .NULL.
ENDIF

DIMENSION laValues[lnItemCount]
loRS.Open()
lnMatchCount = 0

SCAN &lcScope
  lnMatchCount = lnMatchCount + 1
  laValues[1] = RECNO()
  FOR lnCount = IIF(tlAddRecNoColumn, 2, 1) TO (lnItemCount - IIF(tlAddVariantColumn, 1, 0))
    lvValue=EVALUATE(laItems[lnCount])

    DO CASE
      CASE VARTYPE(lvValue) == "T" AND EMPTY(lvValue)
        lvValue = {^1980/01/01 12:00:00 AM}

       CASE VARTYPE(lvValue) == "D" AND EMPTY(lvValue)
          lvValue = {^1980/01/01}
    ENDCASE

    laValues[lnCount] = lvValue
  ENDFOR

  IF tlAddVariantColumn
    laValues[lnItemCount] = ""
  ENDIF

  loRS.AddNew(@laItems,@laValues)
```

```
ENDSCAN

IF lnMatchCount > 0
  loRS.MoveFirst
ENDIF

GO lnLastRecNo
SELECT (lnLastSelect)
RETURN loRS
```

Et voici le code pour convertir un RecordSet en Cursor

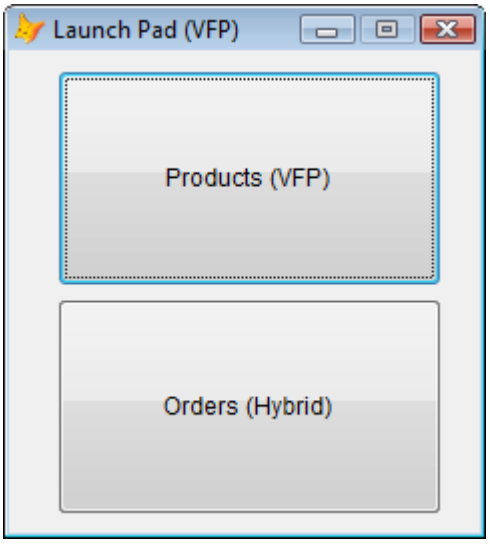
```
* RSTOCURSOR
* Adapted from Microsoft KB Q102692
#INCLUDE "adovfp.h"

LPARAMETERS toRS, tcCursor
LOCAL lcAlias, lnDataTypeEnum, lnLastAbsolutePosition
LOCAL lvValue, lnFieldCount, lcField, lcFieldType, lnFieldSize, lnFieldAttributes
LOCAL lnCount, lnCount2, lnRecNo, lnLastSelect, lnLastRecNo, llRecNoColumn, llMatch
LOCAL lvFieldValue, lnFieldPrecision, lnRecCount, llCreateTable, loField
LOCAL laFields[1, 16]

* Initialize Table Field
llRecNoColumn = (LOWER(toRS.Fields(0).Name) == LOWER("_RecNo"))
lnLastSelect = SELECT()
lcAlias = tcCursor

lnFieldCount = toRS.Fields.Count - IIF(llRecNoColumn, 1, 0)
llCreateTable = .F.

IF !USED(lcAlias)
  llMatch = .T.
  lnCount2 = 0

  * Determine each fileds data type
  FOR lnCount = 1 TO lnFieldCount
    loField = toRS.Fields(lnCount - IIF(llRecNoColumn, 0, 1))
    lcField = LEFT(CHRTRAN(loField.Name, "- ", "__"), 10)
    lnFieldSize = loField.DefinedSize
    lnDataTypeEnum = loField.Type
    lnFieldPrecision = 0
    lnFieldAttributes = loField.Attributes

    DO CASE
      CASE lnDataTypeEnum = ADCHAR OR lnDataTypeEnum = ADVARCHAR
        IF lnFieldSize >= 255
          lcFieldType = "M"
          lnFieldSize = 4
        ELSE
          lcFieldType = "C"
        ENDIF

      CASE lnDataTypeEnum = ADBOOLEAN
        lcFieldType = "L"
        lnFieldSize = 1

      CASE lnDataTypeEnum = ADDBDATE
        lcFieldType = "D"

      CASE lnDataTypeEnum = ADDBTIMESTAMP
        lcFieldType = "T"
        lnFieldSize = 8

      CASE lnDataTypeEnum = ADSINGLE
        lcFieldType = "N"
        lnFieldPrecision = loField.Precision

      CASE lnDataTypeEnum = ADDOUBLE
        lcFieldType = "B"
        lnFieldPrecision = loField.Precision

      CASE lnDataTypeEnum = ADINTEGER
```

```
        lcFieldType = "I"

      CASE lnDataTypeEnum = ADDOUBLE
        lcFieldType = "F"
        lnFieldPrecision = loField.Precision

      CASE lnDataTypeEnum = ADCURRENCY
        lcFieldType = "Y"
        lnFieldPrecision = loField.Precision

        CASE lnDataTypeEnum = ADVARIANT
          lcFieldType = "C"

        OTHERWISE
          LOOP
      ENDCASE

    llCreateTable = .T.
    lnCount2 = lnCount2 + 1

    * Populate table structure array
    DIMENSION laFields[lnCount2, 16]
    laFields[lnCount2, 1] = lcField
    laFields[lnCount2, 2] = lcFieldType
    laFields[lnCount2, 3] = lnFieldSize
    laFields[lnCount2, 4] = lnFieldPrecision
    laFields[lnCount2, 5] = BITTEST(lnFieldAttributes, 5)
    laFields[lnCount2, 6] = .F.
    laFields[lnCount2, 7] = ""
    laFields[lnCount2, 8] = ""
    laFields[lnCount2, 9] = ""
    laFields[lnCount2, 10] = ""
    laFields[lnCount2, 11] = ""
    laFields[lnCount2, 12] = ""
    laFields[lnCount2, 13] = ""
    laFields[lnCount2, 14] = ""
    laFields[lnCount2, 15] = ""
    laFields[lnCount2, 16] = ""
  ENDFOR

  IF !llCreateTable
    SELECT (lnLastSelect)
    RETURN .F.
  ENDIF

  * Create cursor based on table structure array
  SELECT 0
  CREATE CURSOR (lcAlias) FROM ARRAY laFields

  IF NOT USED(lcAlias)
    SELECT (lnLastSelect)
    RETURN .F.
  ENDIF
ENDIF

lnLastRecNo = IIF(EOF(), 0, RECNO())
lnRecCount = RECCOUNT()

* Scan through ADO recordset
IF toRS.BOF OR toRS.EOF
  IF toRS.recordCount > 0
    toRS.MoveFirst
    lnLastAbsolutePosition = toRS.absolutePosition
  ELSE
    lnLastAbsolutePosition = 0
  ENDIF
ELSE
  lnLastAbsolutePosition = toRS.absolutePosition
ENDIF

FOR lnCount = 1 TO toRS.RecordCount
  toRS.AbsolutePosition=lnCount
  lnRecNo = IIF(llRecNoColumn, toRS.Fields(F_RECNO_FIELD).Value, lnCount)
  IF VARTYPE(lnRecNo) != "N"
```

```
    lnRecNo = 0
  ENDIF

  DO CASE
    CASE lnRecNo<0 ;
    OR (!llCreateTable AND llRecNoColumn AND lnRecNo>RECCOUNT())
      LOOP

    CASE !BETWEEN(lnRecNo, 1, lnRecCount)
      * Insert records into cursor from ADO recordset
      APPEND BLANK
      lnRecNo = RECNO()
      IF llRecNoColumn ;
      AND (toRS.LockType = ADLOCKOPTIMISTIC OR toRS.lockType = ADLOCKBATCHOPTIMISTIC)
        toRS.Fields(F_RECNO_FIELD).Value = lnRecNo
      ENDIF
  ENDCASE

  GO lnRecNo
  FOR lnCount2 = 1 TO lnFieldCount
    loField = toRS.Fields(lnCount2 - IIF(llRecNoColumn, 0, 1))
    lcField = loField.Name
    lvValue = loField.Value
    lcFieldType = TYPE(lcField)
    IF !INLIST(lcFieldType, "C", "M", "L", "D", "T", "N", "I", "F", "B", "Y")
      LOOP
    ENDIF

    lvFieldValue = EVALUATE(lcField)

    * Handle differences in ADO vs. VFP dates
    IF !VARTYPE(lvFieldValue) == VARTYPE(lvValue)
      DO CASE
        CASE lcFieldType == "T"
          IF EMPTY(lvValue)
            lvValue = {^1980/01/01 12:00:00 AM}
          ELSE
            lvValue = DTOT(lvValue)
          ENDIF

        CASE lcFieldType == "D"
          IF EMPTY(lvValue)
            lvValue = {^1980/01/01}
          ELSE
            lvValue = TTOD(lvValue)
          ENDIF

        OTHERWISE
          LOOP
      ENDCASE
    ENDIF

    IF lvFieldValue==lvValue
      LOOP
    ENDIF

    REPLACE (lcField) WITH lvValue
  ENDFOR
ENDFOR

IF lnLastAbsolutePosition>0
  toRS.AbsolutePosition = lnLastAbsolutePosition
ENDIF

LOCATE
SELECT (lnLastSelect)
RETURN
```

J'ai inclus ces fonctions dans la bibliothèque de classes visuelle, l'InteropToolKit.vcx. Cette bibliothèque contient également le code d'initialisation et de destruction de l'Interop.

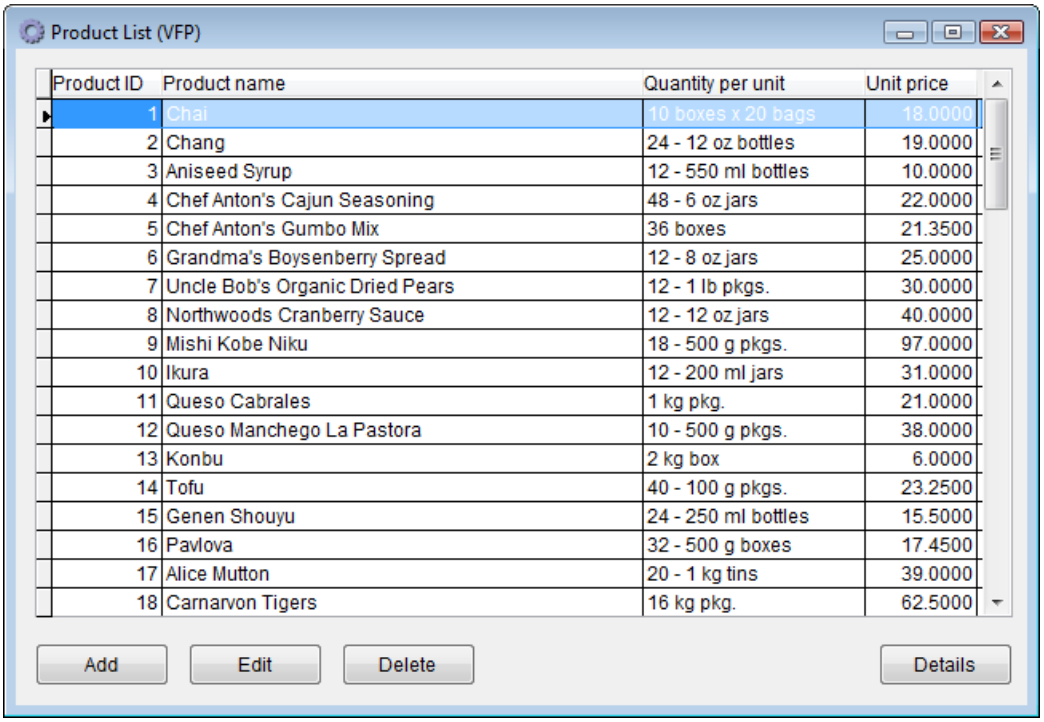
Notre application utilisera la base de données exemple Northwind, qui est fournie avec VFP. Elle présentera les deux fonctions Produits et Commandes, qui seront lancées depuis un seul form VFP.



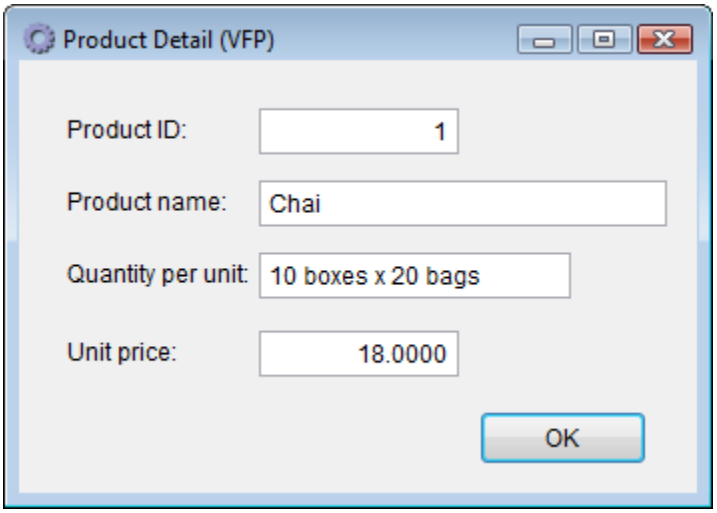
Nous ne nous attarderons pas dans une découverte pas à pas de cette application, je vais uniquement vous en présenter les points importants et vous laisser explorer le détail du code. La première chose intéressante est le Main.prg, dans lequel est définie une variable globale qui sera utilisée aussi bien par VFP que par les composants VB.Net. Le nom d'utilisateur n'est pas vérifié, nous admettrons qu'il est toujours correct.

```
* MyCompany .Net WinForms Interop Sample
LOCAL lcTalk, lcConfirm, lcLoginResult, loLogin
LOCAL lcLoginResult, lcUsername, lcPassword
PRIVATE goInteropToolkit

lcTalk = SET("TALK")
lcConfirm = SET("CONFIRM")
lcExclusive = SET("EXCLUSIVE")
lcLoginResult = ""

CLOSE DATABASES ALL
SET TALK OFF
SET CONFIRM ON
SET EXCLUSIVE OFF

loLogin = NEWOBJECT("frmLogin", "MyCompany")
loLogin.Show()
lcLoginResult = loLogin.cHowExit
lcUsername = loLogin.cUsername
lcPassword = loLogin.cPassword
loLogin = NULL

IF "OK" = lcLoginResult
  * Assume user passed check for valid login
  goInteropToolkit = NEWOBJECT("cusInteropToolkit", "InteropToolkit")

  * Pass the username to the shared globals area of the toolkit so it can be used by VB.NET
  goInteropToolkit.oNetInteropToolbox.Globals.Add("Username", lcUsername)

  OPEN DATABASE (HOME(2) + "\Northwind\Northwind") SHARED
  SET DATABASE TO Northwind
  USE Products SHARED IN 0 NOUPDATE
  USE Orders IN 0 NOUPDATE
  USE OrderDetails IN 0 NOUPDATE
  DO FORM LaunchPad
ELSE
  MESSAGEBOX("Login failed. Application will exit", "Launch pad")
ENDIF


SET TALK &lcTalk.
SET CONFIRM &lcConfirm.
SET EXCLUSIVE &lcExclusive.
CLOSE DATABASES ALL
CLEAR ALL
RELEASE ALL
```

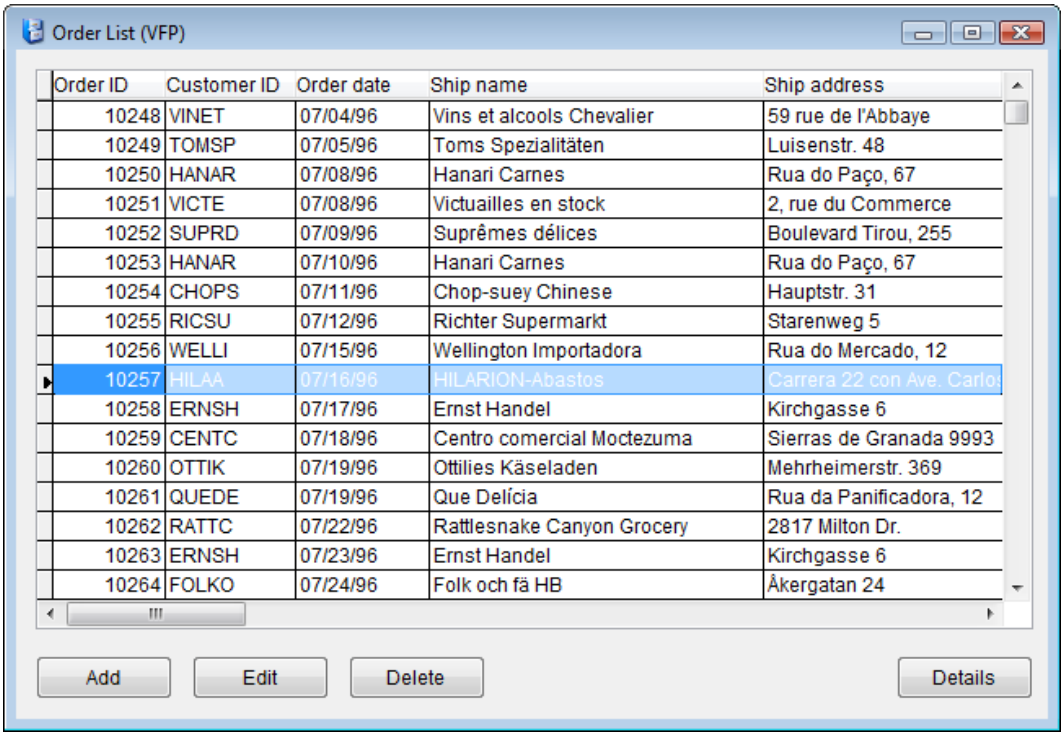
Il n'y a rien de particulier dans le code du form Launch Pad. En cliquant sur le bouton « Products », on ouvre un form VFP qui affiche la table Products dans une grille.



Il n'y a aucune fonctionnalité codée dans les boutons Add, Edit, et Delete, qui ne font qu'afficher un Messagebox. Le bouton Details lance un deuxième form VFP, avec les détails de l'enregistrement sélectionné dans la grille.



Le bouton Orders du Launch Pad affichera les détails de commande dans un form VFP.



La méthode Init du form amène au niveau du form le gestionnaire d'événement qui sera supprimé dans la méthode Destroy.

```
DODEFAULT()

This.oEventHandler = NEWOBJECT("cusProductDetailsEventHandler", "classes\ProductDetailsEventHandler.prg")
This.oOrderDetails = CREATEOBJECT("MyCompany.Winforms.Orders.Interop.OrderDetails")
EVENTHANDLER(This.oOrderDetails, This.oEventHandler)
```

Le gestionnaire d'événements est contenu dans son prg spécifique.

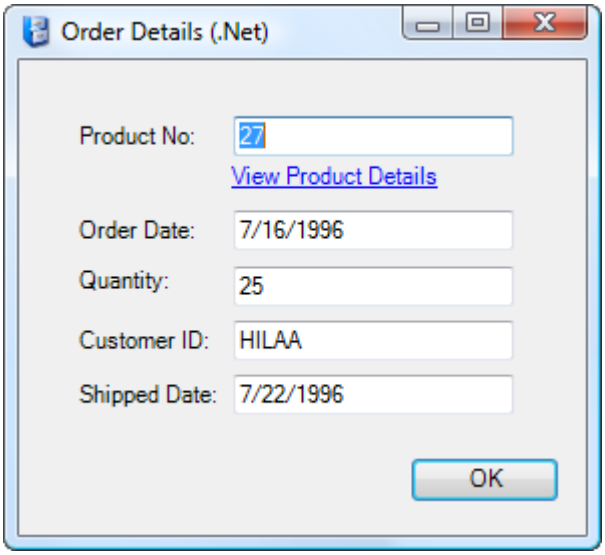
```
DEFINE CLASS cusProductDetailsEventHandler AS custom OLEPUBLIC

  IMPLEMENTS IOrderDetailsEventSink IN ;
"C:\INTEROP\MYCOMPANY\MYCOMPANY.WINFORMS.ORDERS\MYCOMPANY.WINFORMS.ORDERS\BIN\DEBUG\MYCOMPANY.WINFORMS.ORDERS.TLB"

  PROCEDURE IOrderDetailsEventSink_ViewProductDetailsEvent(tiProductID AS Number) AS VOID
    SELECT * ;
    FROM Products ;
    WHERE ProductID = tiProductID ;
    INTO CURSOR cProduct NOFILTER

    DO FORM ProductDetail
    USE IN cProduct
  ENDPROC
ENDDEFINE
```

Ici aussi, les boutons Add, Edit, et Delete ne servent à rien. Mais le bouton Details, lui, appelle un WinForm VB.Net qui contient les détails de l'enregistrement sélectionné. Voici le code pour ce bouton Details.

```
LOCAL lnOrderId, loRS

lnOrderId = Orders.OrderId

SELECT ProductId, OrderDate, Quantity, CustomerID, ShippedDate ;
       FROM Orders JOIN OrderDetails ON Orders.OrderId = OrderDetails.OrderId ;
       WHERE Orders.OrderId = lnOrderId ;
       INTO CURSOR cOrderDet

* Create the ADO RecordSet and add it to interop globals
loRS = goInteropToolkit.CursorToRs("cOrderDet")
IF goInteropToolkit.oNetInteropToolbox.Globals.ContainsKey("rsOrderDetail")
  goInteropToolkit.oNetInteropToolbox.Globals.Item("rsOrderDetail") = loRS
```

```
ELSE
  goInteropToolkit.oNetInteropToolbox.Globals.Add("rsOrderDetail", loRS)
ENDIF

* Instantiate the VB.NET form
DO CASE
  CASE ISNULL(ThisForm.oOrderDetails) ;
  OR VARTYPE(ThisForm.oOrderDetails) != "O"
    ThisForm.oOrderDetails = CREATEOBJECT("MyCompany.Winforms.Orders.Interop.OrderDetails")

  CASE ThisForm.oOrderDetails.IsFormDisposed
    ThisForm.oOrderDetails = CREATEOBJECT("MyCompany.Winforms.Orders.Interop.OrderDetails")
ENDCASE

ThisForm.oOrderDetails.SetupData()
ThisForm.oOrderDetails.Show()

USE IN cOrderDet
```

Dans ce code, on récupère l'information concernant la commande en cours, puis on la convertit en un RecordSet ADO, qu'on passe ensuite à la collection Interop Globals. Il aurait été plus simple de passer le RecordSet à une méthode dans le form VB.Net, mais quand j'ai essayé, j'ai eu un message d'erreur provenant de l'Interop Form Toolkit me signalant que les objets ADODB n'étaient pas supportés, et que la méthode d'interop ne serait donc pas générée. L'utilisation d'un Global résout ce problème.



En cliquant sur le lien View Product Details, vous lancez et affichez le form VFP ProductDetails. Voici le code VB.Net.

```
Imports Microsoft.InteropFormTools
Imports System.Data
Imports System.Data.OleDb
Imports ADODB


<InteropForm()> _
Public Class OrderDetails

  Dim OrderData As ADODB.Recordset

  <InteropFormMethod()> _
  Public Sub SetupData()
    ' Sets control values from Order info.
    OrderData = My.InteropToolbox.Globals("rsOrderDetail")

    txtProductNo.Text = OrderData.Fields("ProductID").Value
    txtOrderDate.Text = OrderData.Fields("OrderDate").Value
    txtQuantity.Text = OrderData.Fields("Quantity").Value
    txtCustomerID.Text = OrderData.Fields("CustomerID").Value
    txtShippedDate.Text = OrderData.Fields("ShippedDate").Value
  End Sub
```

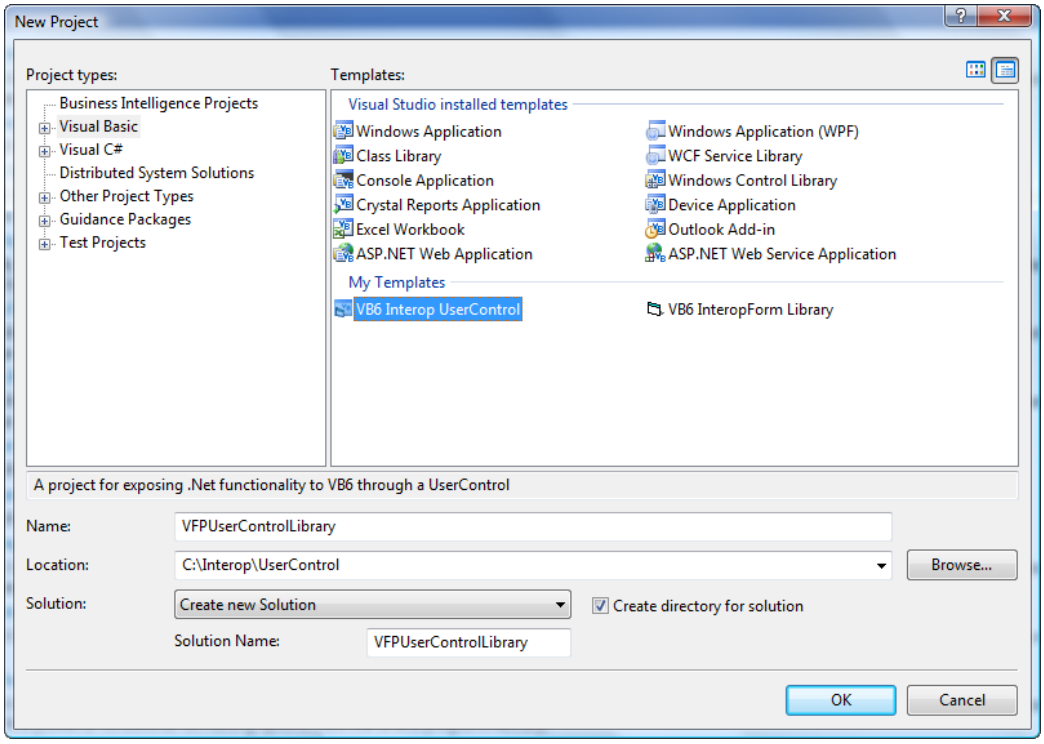
```
  <InteropFormEvent()> _
  Public Event ViewProductDetailsEvent As ProductDetailsEventHandler
  Public Delegate Sub ProductDetailsEventHandler(ByVal ProductID As Integer)

  Private Sub cmdOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdOK.Click
Me.Close()
  End Sub

  Private Sub lnkProductDetails_LinkClicked(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.LinkLabelLinkClickedEventArgs) Handles lnkProductDetails.LinkClicked
    RaiseEvent ViewProductDetailsEvent(txtProductNo.Text)
  End Sub
End Class
```

Le code important est celui de l'InteropFormEvent, et le RaiseEvent de la méthode lnkProductDetails_LinkClicked qui gère l'appel à VFP pour afficher le form ProductDetails.

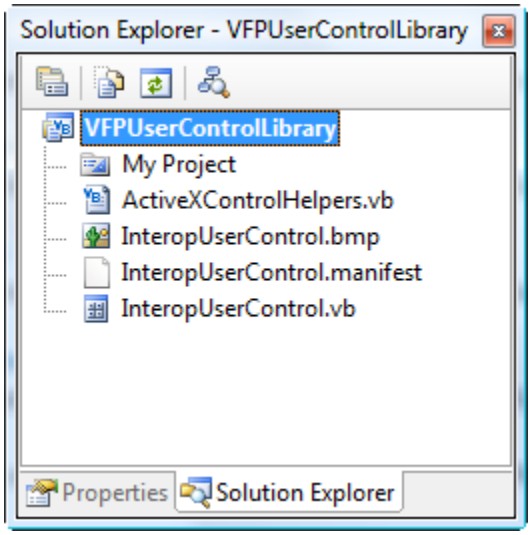
Contrôles Utilisateurs Personnalisés

Avec l'Interop Form Toolkit, nous pouvons facilement créer nos propres contrôles utilisateurs personnalisés pour les utiliser sur des forms. Du point de vue de VFP, ces contrôles utilisateurs sont des contrôles ActiveX. Nous allons ainsi pouvoir apporter à notre application VFP de nouvelles fonctionnalités, grâce aux possibilités de cette technique. Dans l'exemple suivant, je vais vous montrer comment créer un contrôle et le poser sur votre form VFP.

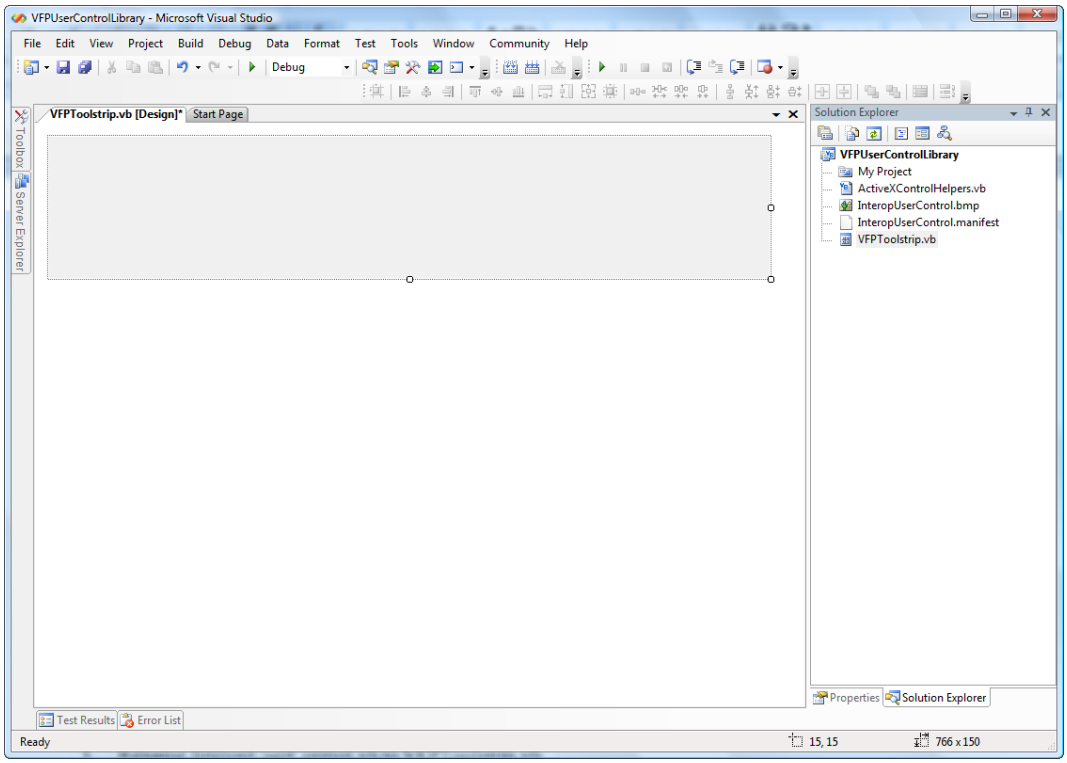
1. Dans Visual Studio, créez un nouveau projet de type « VB User Control », appelez-le VFPUserControlLibrary.



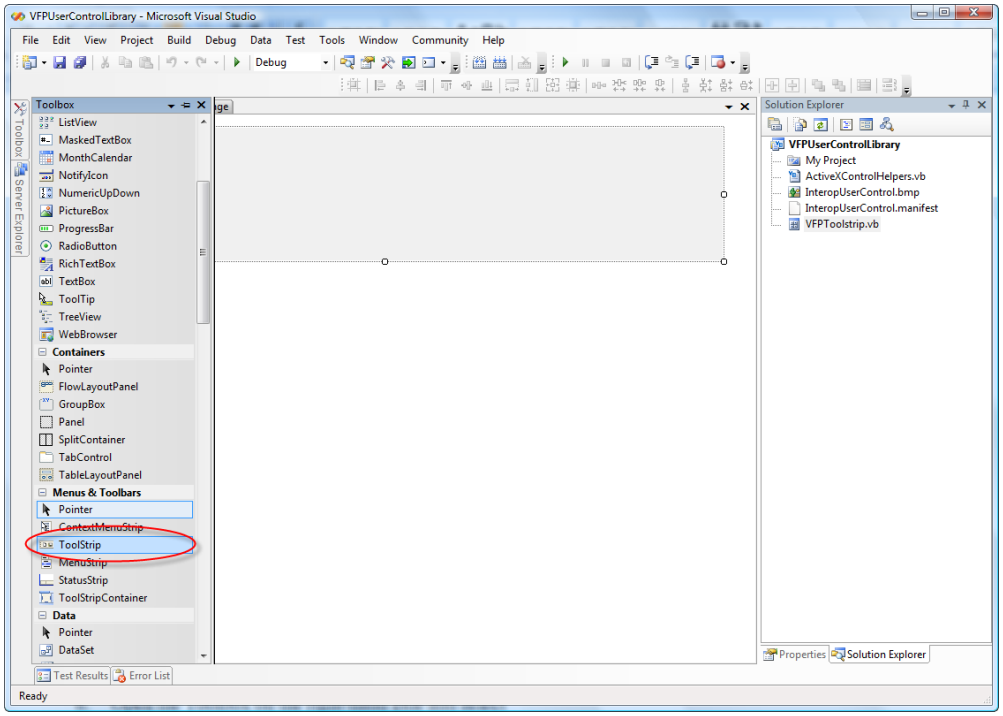
2. Plusieurs éléments sont automatiquement ajoutés à la Solution.



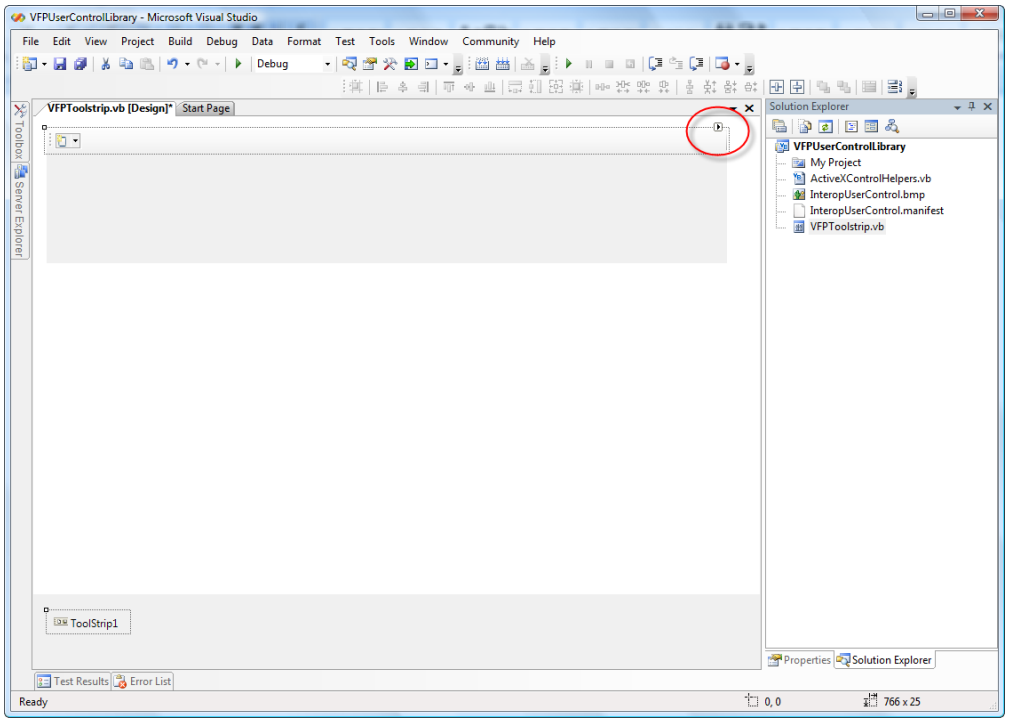
3. Renommez InteropUserControl.vb en VFPToolstrip.vb
4. Double-cliquez sur VFPToolstrip.vb pour l'ouvrir en mode de conception.
5. Élargissez la zone grise en tirant vers la droite à l'aide de la poignée.



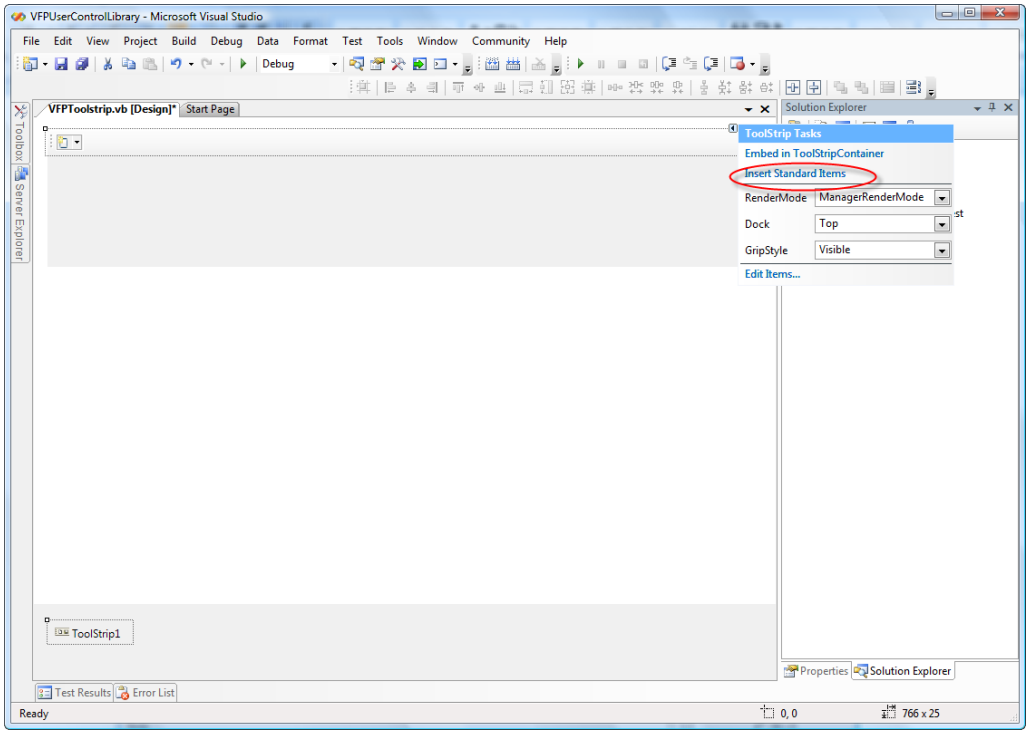
6. Ouvrez la boite à outils et repérez le contrôle ToolStrip.



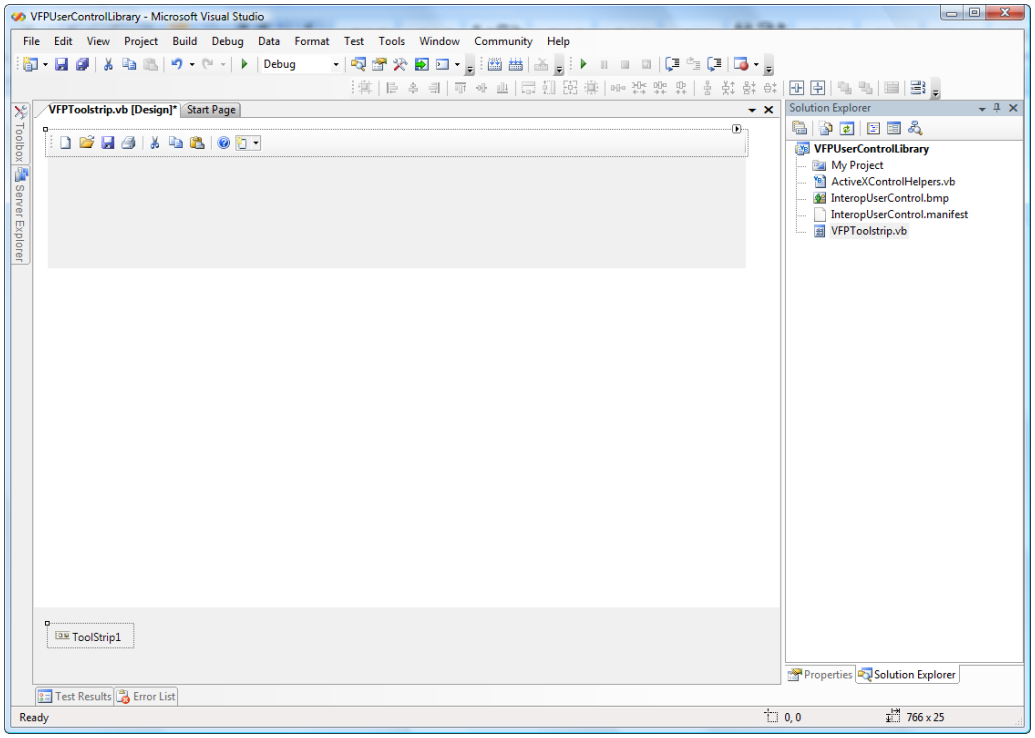
7. Faites glisser le contrôle ToolStrip sur le rectangle gris dans le concepteur.



8. Cliquez sur la petite flèche en haut à droite du ToolStrip pour afficher le menu des tâches, et choisissez Insérer des éléments standard.



9. De nouveaux boutons sont ainsi ajoutés au ToolStrip.



10. Cliquez-droit sur le ToolStrip, choisissez « Afficher le Code » pour ouvrir la fenêtre de code. Créez un événement pour chaque bouton en saisissant le code suivant :

```
'Please enter any new code here, below the Interop code

Public Event NewDocument()
Public Event OpenDocument()
Public Event SaveDocument()
Public Event PrintDocument()
Public Event Cut()
Public Event Copy()
Public Event Paste()
Public Event Help()
```

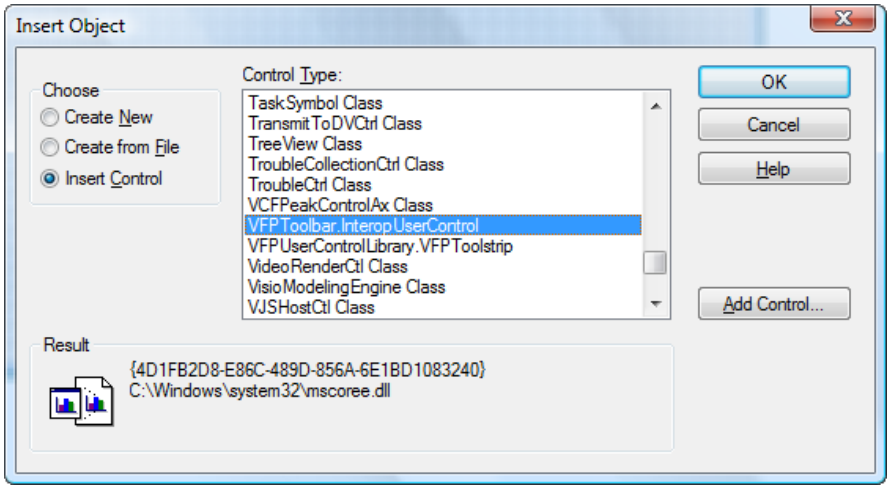
11. Revenez maintenant au concepteur, et créez les gestionnaires de Click en double-cliquant sur chacun des boutons.
12. Saisissez le code de déclenchement d'événement (RaiseEvent) pour chaque bouton. Utilisez les noms des événements que vous venez de créer.

```
    Private Sub NewToolStripButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles NewToolStripButton.Click
        RaiseEvent NewDocument()
    End Sub

    Private Sub OpenToolStripButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles OpenToolStripButton.Click
        RaiseEvent OpenDocument()
    End Sub

    Private Sub SaveToolStripButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles SaveToolStripButton.Click
        RaiseEvent SaveDocument()
    End Sub

    Private Sub PrintToolStripButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles PrintToolStripButton.Click
        RaiseEvent PrintDocument()
    End Sub

    Private Sub CutToolStripButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles CutToolStripButton.Click
        RaiseEvent Cut()
    End Sub

    Private Sub CopyToolStripButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles CopyToolStripButton.Click
        RaiseEvent Copy()
    End Sub

    Private Sub PasteToolStripButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles PasteToolStripButton.Click
        RaiseEvent Paste()
    End Sub

    Private Sub HelpToolStripButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles HelpToolStripButton.Click
        RaiseEvent Help()
    End Sub
```

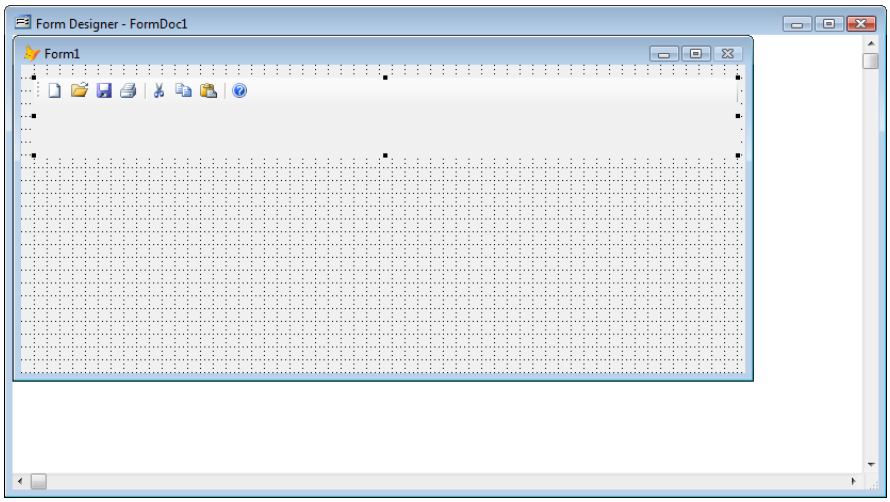
13. Le complément Generate Interop... n'est pas nécessaire pour les contrôles utilisateurs, il ne sert que pour les forms. Il ne reste qu'à compiler le projet.

Nous en avons terminé avec le code VB.Net pour le contrôle utilisateur. Il nous faut maintenant créer le form VFP qui recevra ce contrôle.

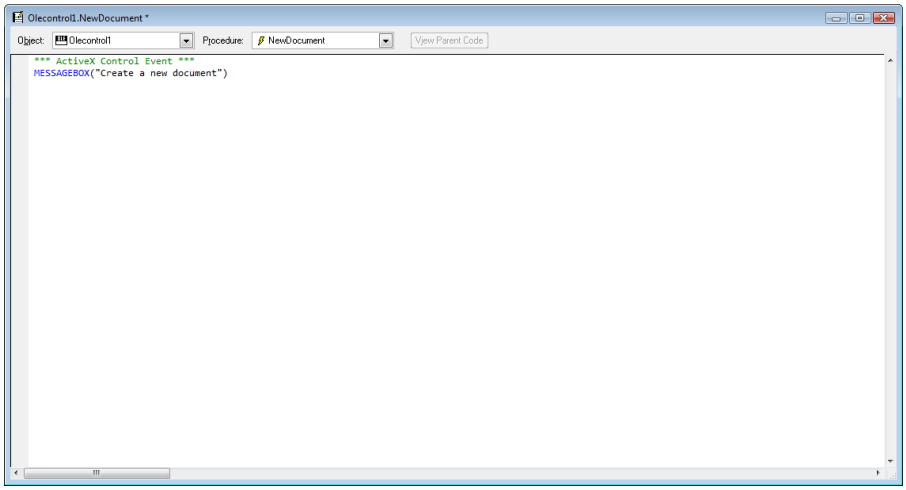
1. Dans Visual FoxPro, créez un nouveau form.
2. Ajoutez un contrôle OLE à ce form. La boite de dialogue « Insert Object » apparaît.



3. Dans la liste des contrôles, descendez jusqu'à VFPToolbar.InteropUserControl, validez par OK. Le contrôle que nous venons de créer est ajouté au form.



4. Double-cliquez sur ce contrôle pour ouvrir une fenêtre de code.
5. Repérez la méthode NewDocument, ajoutez-y le code nécessaire pour afficher un MessageBox dont le texte correspondra au bouton.



6. Faites de même pour chacun des boutons.
7. Enregistrez votre form, lancez-le. Remarquez les infos-bulles en passant au dessus de chaque bouton. Cliquez sur les boutons pour voir les MessageBox.

Vous savez maintenant à quel point il est facile de créer des contrôles personnalisés en VB.Net et de les utiliser sur un form VFP.

Accueillir XAML

Windows Presentation Foundation (WPF) est une technologie d'Interface Utilisateur (UI) que Microsoft a mis dans le Framework .Net 3.0. XAML en est un des composants principaux, permettant de décrire la mise en page de l'interface utilisateur dans un langage de type XML. On peut aussi accueillir XAML sur un WinForm et le rendre compatible COM.

1. Créez une nouvelle bibliothèque InteropForm, appelez-la XAMLForm.
2. Ajoutez un deuxième projet à cette solution. Vous pouvez créer un nouveau projet basé sur la Custom Control Library (WPF) du Framework .Net 3.0, ou bien rattacher un projet existant. J'ai ici choisi d'utiliser le projet exemple 15Puzzle qui est fourni avec le SDK Windows.
3. Générez la solution.
4. Sélectionnez le projet XAMLForm et ajoutez-y une référence au projet que vous venez d'ajouter.
5. Ajoutez des références supplémentaires aux bibliothèques suivantes : PresentationCore, PresentationFramework, WindowsBase, et WindowsFormsIntegration.
6. Ouvrez le form InteropForm1.vb dans le concepteur.
7. Cliquez-droit pour ouvrir la fenêtre de code, ajoutez le code ci-dessous :

```
Imports Microsoft.InteropFormTools
Imports System.Windows.Forms.Integration
Imports System.ComponentModel
Imports System.Windows.Forms

<InteropForm()> _
Public Class InteropForm1
    Private Sub InteropForm1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles MyBase.Load

        Dim host As New ElementHost()
        host.Dock = DockStyle.Fill

        Dim uc As New PuzzleProject.VectorContent

        host.Child = uc

        Me.Controls.Add(host)
    End Sub
End Class
```

8. Ceci ajoute un des contrôles XAML, VectorContent, depuis le projet WPF, et en fait un contrôle sur le WinForm.
9. Dans le menu, choisissez Outils → Generate InteropForm Wrapper Classes
10. Générez la solution.
11. Dans Visual FoxPro, créez un nouveau programme que vous appellerez XAML.prg. Son code ne contiendra que l'ossature indispensable au fonctionnement de ce form. Saisissez le code suivant :

```
loToolbox = CREATEOBJECT("Microsoft.InteropFormTools.InteropToolbox")
loToolbox.Initialize()
loToolbox.EventMessenger.RaiseApplicationStartedupEvent()

loXAMLForm = CREATEOBJECT("XAMLForm.Interop.InteropForm1")
loXAMLForm.Show(1)

loToolbox.EventMessenger.RaiseApplicationShutdownEvent()
```

12. Enregistrez, compilez, exécutez le programme. Le form Interop sera affiché avec le contrôle XAML hébergé, animation comprise.

Nous en avons terminé avec le code. Maintenant, vous connaissez plusieurs façons d'utiliser l' Interop Forms Toolkit.

Distribution du Runtime

Il faut que le runtime du Net 2.0 soit installé pour utiliser le Toolkit. Ce sera le cas de la plupart des utilisateurs, puisqu'il est préinstallé avec Windows Vista, et téléchargé en tant que mise à jour système de Windows XP. Il vous faut également le runtime de l'InteropForm, qui est disponible sur le même site de téléchargement que le Toolkit.

Microsoft préconise l'utilisation de la technique de projet d'installation dans VS2005. Un configurateur personnalisé est un Setup.exe spécifique généré par Visual Studio, qui connaît les prérequis de votre application. Vous pouvez également lancer l'install de votre choix, mais ça complique les choses.

Les composants COM écrits en .Net bénéficient d'un point positif, c'est qu'il n'est pas nécessaire de les enregistrer pour qu'ils fonctionnent. Vous obtiendrez même une erreur si vous essayez de les enregistrer avec Regsrv32. Malheureusement, on ne peut pas utiliser cette installation sans enregistrement, à cause d'une partie de ce que fait le Toolkit. Il faut enregistrer les assemblies avec Regasm.exe.

Étendre le Toolkit

Le Toolkit est conçu pour être étendu. Vous pouvez l'étendre au travers de ces quatre orientations spécifiques :

- Interfaces – Tous les services du noyau du Toolkit présentent des Interfaces sur lesquelles vous pouvez développer votre propre implémentation
- Héritage – Tous les services du noyau du Toolkit sont héritables, et toutes les fonctionnalités peuvent être substituées
- Classes Partielles – Le code généré par le complément peut être étendu par l'utilisation de classes partielles
- Code Source – Tout le code source est installé avec le Toolkit

En Résumé…

L'Interop Forms Toolkit vous permet d'ajouter facilement des WinForms à vos applications Visual FoxPro. Ces forms peuvent interagir avec VFP par l'intermédiaire d'événements ou de données globales. Ces forms peuvent aussi héberger des contrôles XAML qui vous permettent de profiter des spécificités de Windows Presentation Foundation. Vous pouvez également construire des contrôles utilisateurs personnalisés et les déposer sur des forms Visual FoxPro, VFP les voyant alors comme des contrôles ActiveX. L'utilisation du Toolkit prolonge la vie de vos applications VFP, et vous aide à commencer la migration de vos applications sous .Net.

Craig Berntson est Microsoft Most Valuable Professional (MVP) Visual FoxPro, et Microsoft Certified Solution Developer ; il est aussi President du Salt Lake City Fox User Group. Il est l'auteur de « CrysDev: A Developer's Guide to Integrating Crystal Reports », disponible aux éditions Hentzenwerke. Il a aussi écrit pour FoxTalk et pour le bulletin du Visual FoxPro User Group (VFUG). Il a été conférencier à l' Advisor DevCon, à l' Essential Fox, aux DevEssentials, au Great Lakes Great Database Workshop, à Southwest Fox, au DevTeach, au FoxCon, au German FoxPro Devcon, au Prague FoxPro DevCon, aux Microsoft DevDays, et dans des groupes d'utilisateurs partout aux USA. Actuellement, Craig est Senior Software Engineer dans une société du Fortune 100 à Salt Lake City (Utah), où il développe des applications centrées sur les données pour l'industrie dans le secteur médical. Vous pouvez lui écrire à craig@craigberntson.com, www.craigberntson.com, ou sur son blog www.craigberntson.com/blog.