Extend and Upgrade Microsoft® Visual Basic® 6.0 Applications with Visual Basic 2008 and the Interop Forms Toolkit 2.0

Dominic Betts, Senior Technologist - Content Master Ltd
10/10/2008

Introduction

Many Visual Basic 6.0 applications have not yet made the jump to the world of .NET. These applications are not just missing out on the additional functionality that the Microsoft .NET Framework offers; they will become increasingly difficult to migrate and maintain as Visual Basic 6.0 skills start to disappear. Although the Microsoft Visual Studio® .NET Visual Basic Upgrade Wizard, or alternatives such as tools from ArtinSoft (www.ArtinSoft.com) and Code Architects (www.vbmigration.com), offer an automated migration path for Visual Basic 6.0 projects, this approach does not work effectively in all cases. If automated code migration is not possible, the cost in terms of both resources and time of a full rewrite of the application in Visual Basic .NET may be prohibitive.

The Interop Forms Toolkit 2.0 does not offer automated migration, rather it makes a phased upgrade approach possible. You can extend the application's functionality by rewriting individual forms to take advantage of new .NET functionality or by adding completely new .NET forms. You can also use the Toolkit to facilitate a form-by-form upgrade to Visual Basic .NET, avoiding the need to do a complete rewrite of the entire application. The Toolkit provides all of the infrastructure and interop code that is necessary to create a hybrid Visual Basic 6.0 and Visual Basic .NET application, leaving the developer free to focus on the application's business logic and presentation aspects.

Version 2.0 of the Interop Forms Toolkit offers some major enhancements over the previous version:

- Interop UserControls enable you to create custom controls in Visual Studio .NET that can be hosted on a Visual Basic 6.0 form.
- The Toolkit now supports Visual Basic 6.0 multiple-document interface (MDI) forms by using Interop UserControls.
- The deployment of these hybrid applications is much simpler.

This paper will outline how to use the Interop Forms Toolkit and how to create deployment packages that are suitable for a range of scenarios. Visual Basic 6.0 and Visual Studio 2008 were used to create all of the examples in this paper.

Background

Component Object Model (COM) interop is one of the interoperability mechanisms that the common language runtime (CLR) offers. COM interop enables managed code to interact with COM components written in an unmanaged language. It also enables the CLR to expose managed objects as COM components. Using this mechanism, Visual Basic 6.0 code can interact with an object hosted by the CLR and written in any .NET language. A layer of code known as a COM callable wrapper (CCW) handles the run-time translation of calls from an unmanaged COM environment, such as a Visual Basic 6.0 application, to the managed environment and back again.

A developer can generate a CCW by using either Visual Studio or the Type Library Exporter (Tlbexp.exe). The process of generating the CCW and referencing the exposed, managed component from Visual Basic 6.0 is simple. However, there is more complexity involved in creating Visual Basic .NET code that will behave in the expected way when unmanaged Visual Basic 6.0 code calls your managed code. In practice, the usage of attributes defined in the **System.Runtime.InteropServices** namespace and a careful choice of data types are necessary to manage the behaviour of the COM interop services at run time.

The Interop Forms Toolkit 2.0 automates the creation of two types of COM component. The Toolkit helps to expose both Windows® Forms and **UserControl** derived classes created by using Visual Studio 2005 or Visual Studio 2008 as COM components. Visual Basic 6.0 developers can add these components to their applications and use them just like any other COM component.

Interop Forms Toolkit 2.0

What Is the Interop Forms Toolkit 2.0?

The Interop Forms Toolkit 2.0 is a set of tools designed to help a developer achieve the following tasks:

1. Create Windows Forms by using Visual Basic in Visual Studio 2008 (or 2005), and then host these forms in a Visual Basic 6.0 application. These forms can use any of the extensive set of controls available in the .NET Framework and have any code-behind logic implemented in Visual Basic .NET.
2. Create **UserControl** derived classes by using Visual Basic in Visual Studio 2008 (or 2005), and then host these controls, just like any Microsoft ActiveX® control, on a Visual Basic 6.0 form. **UserControl** classes are composite controls that contain other .NET controls and can include code-behind logic written in Visual Basic .NET.
3. Share global application state data between the managed and unmanaged elements of a hybrid application, and propagate events from managed code to unmanaged code or from unmanaged code to managed code.
4. Deploy hybrid applications to an end user's desktop, making sure that all of the prerequisites for both managed and unmanaged code are installed and that the installer completes any necessary component registration.

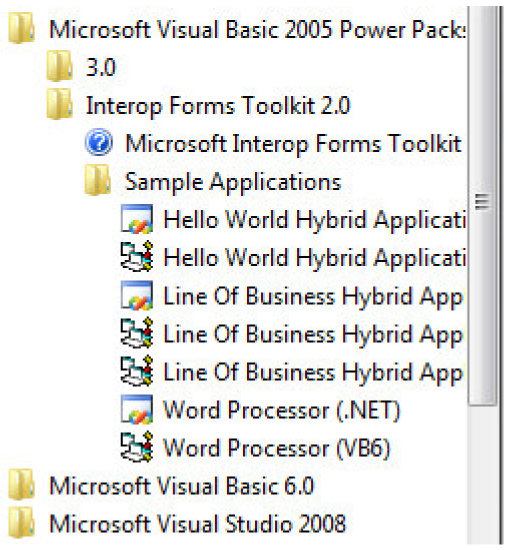
A Visual Basic 6.0 developer can use any of the components generated by the Toolkit in a completely natural Visual Basic 6.0 way. The developer sets references to the components in the Visual Basic 6.0 integrated development environment (IDE) in the standard way. The Microsoft IntelliSense® technology then exposes all of the interop component's properties, methods and events to the Visual Basic 6.0 code editor.

Installation and Setup in Visual Studio 2008

The prerequisites for installing the Interop Forms Toolkit are Visual Basic 6.0 and a full version of either Visual Studio 2005 or Visual Studio 2008. (Note: Visual Studio add-ins are not supported in Visual Basic Express Edition.)

The Interop Forms Toolkit 2.0 download page is located at http://msdn.microsoft.com/en-us/vbasic/bb419144.aspx.

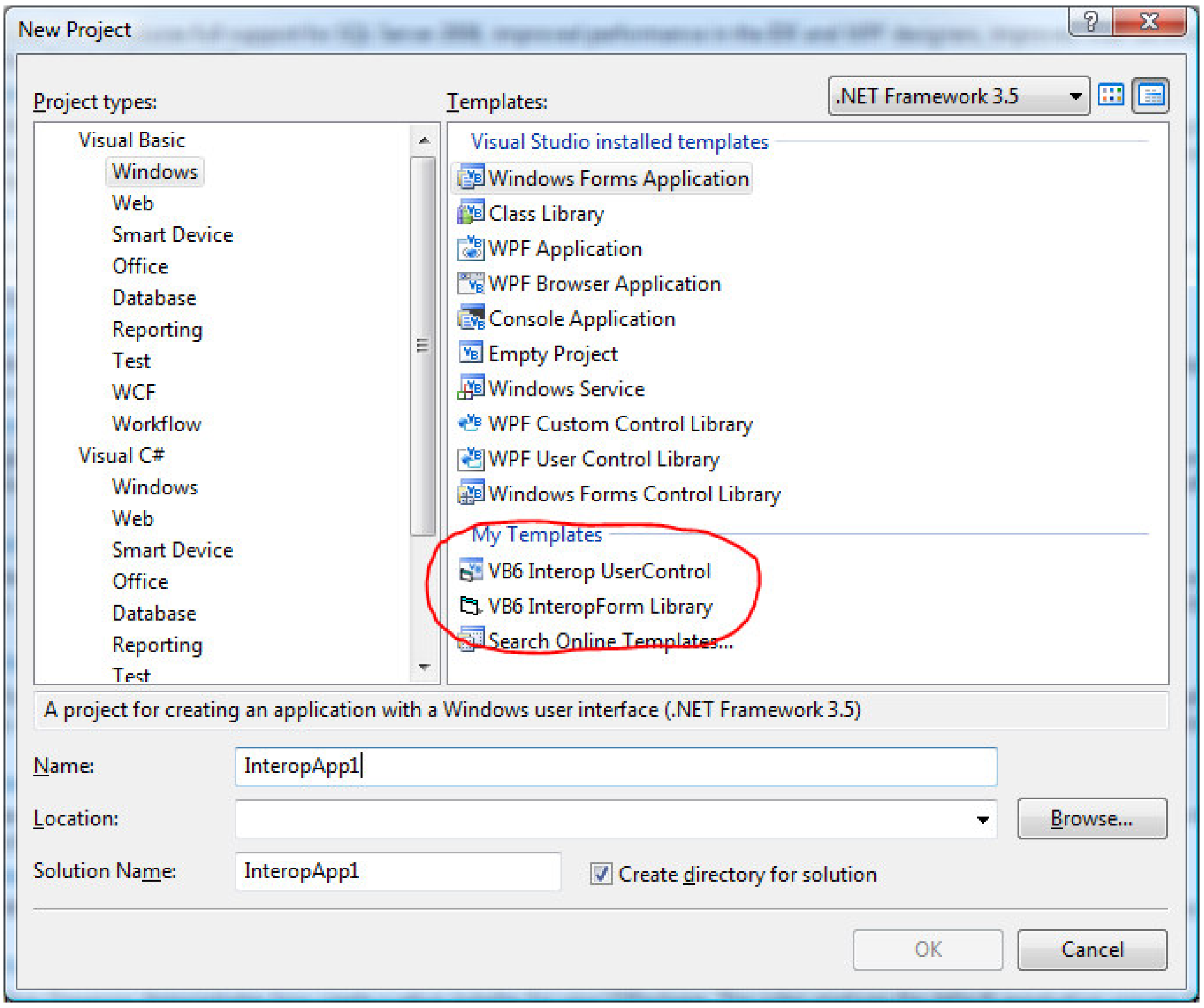
The installer creates new **Start** menu entries that point to the help file and a set of sample applications (Figure 1).



Microsoft Visual Basic 2005 Power Pack
3.0
Interop Forms Toolkit 2.0
Microsoft Interop Forms Toolkit
Sample Applications
Hello World Hybrid Applicati
Hello World Hybrid Applicati
Line Of Business Hybrid App
Line Of Business Hybrid App
Line Of Business Hybrid App
Word Processor (.NET)
Word Processor (VB6)
Microsoft Visual Basic 6.0
Microsoft Visual Studio 2008

Figure 1: Installed Start menu entries

The installer creates two new project templates in Visual Studio (Figure 2).



New Project

Project types:
Visual Basic
Windows
Web
Smart Device
Office
Database
Reporting
Test
WCF
Workflow
Visual C#
Windows
Web
Smart Device
Office
Database
Reporting
Test

Templates: .NET Framework 3.5

Visual Studio installed templates
Windows Forms Application
Class Library
WPF Application
WPF Browser Application
Console Application
Empty Project
Windows Service
WPF Custom Control Library
WPF User Control Library
Windows Forms Control Library

My Templates
VB6 Interop UserControl
VB6 InteropForm Library
Search Online Templates...

A project for creating an application with a Windows user interface (.NET Framework 3.5)

Name: InteropApp1
Location:
Solution Name: InteropApp1
☑ Create directory for solution

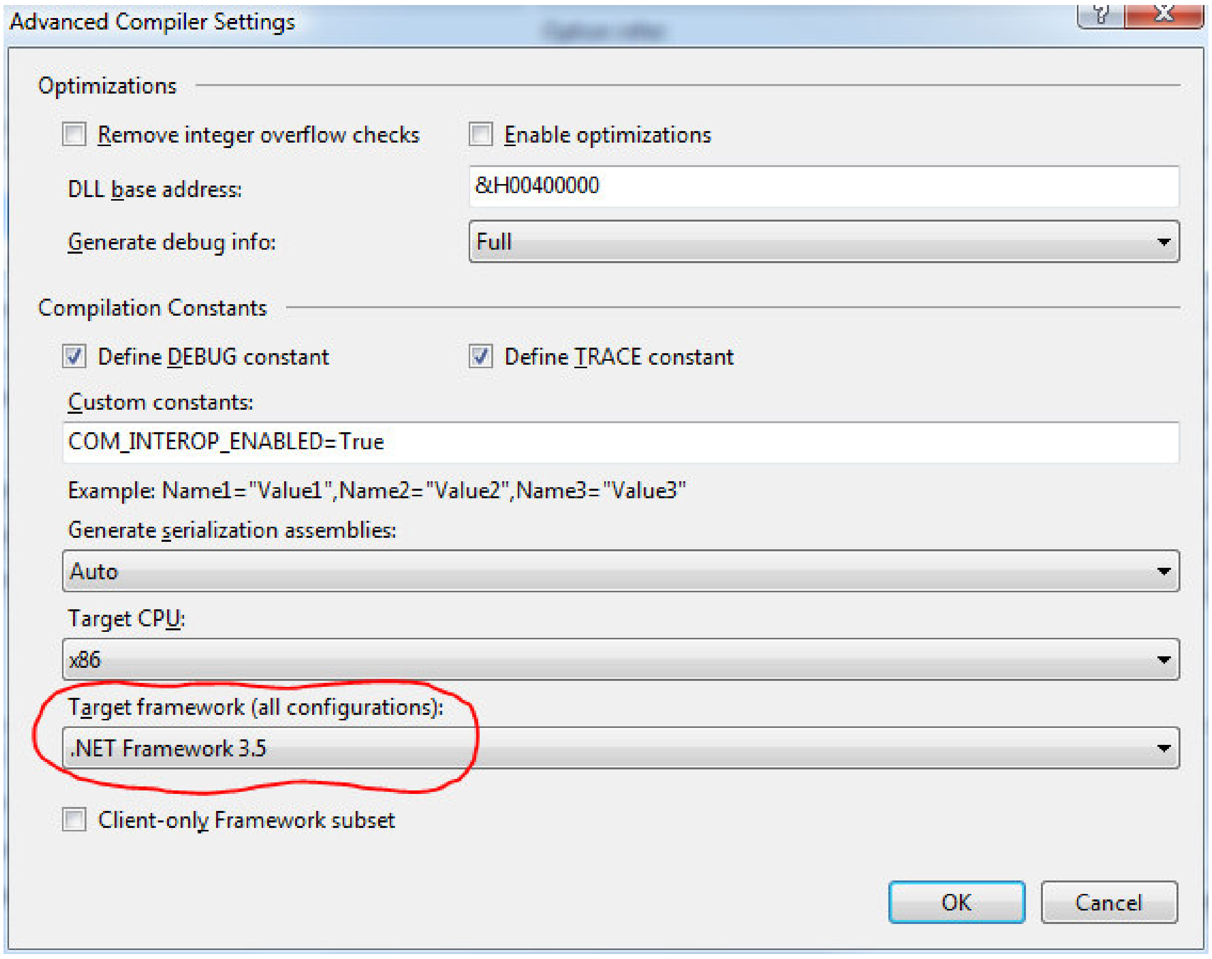
Browse... OK Cancel

Figure 2: New project templates

The installer also adds item templates for creating new InteropForms and Interop UserControls, and adds a collection of code snippets to help the developer write any custom interop code.

If you are using Visual Studio 2008, there are a couple of changes that you should make to your project settings. You can either make these changes manually in each Interop project, or modify the supplied project templates so that you do not have to remember to make the changes each time you create a new Interop project:

1. Change the target framework from 2.0 to 3.5. You do this in the **Advanced Compiler Settings** dialog box (Figure 3).



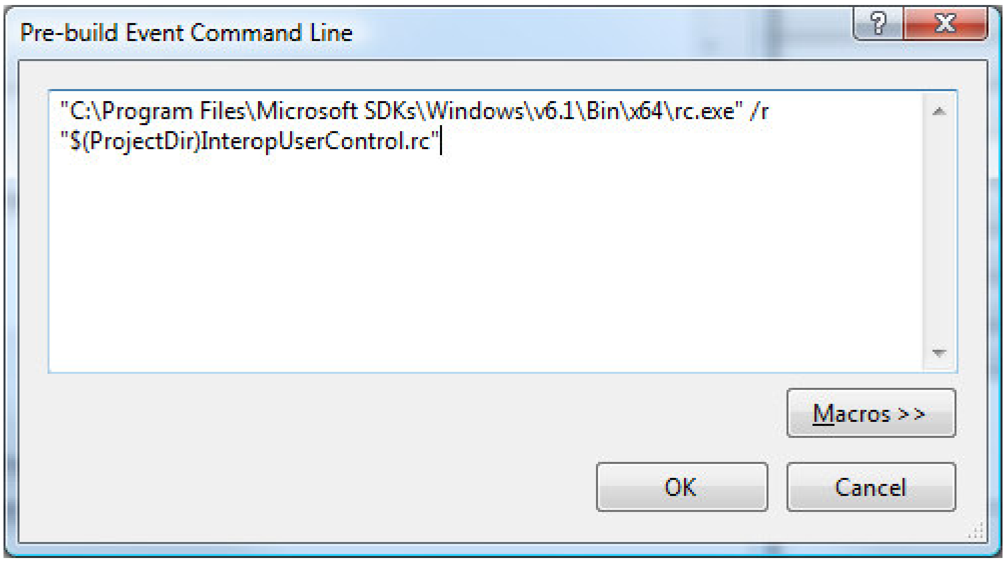
Remove integer overflow checks: ☐
Enable optimizations: ☐
DLL base address: &H00400000
Generate debug info: Full

Compilation Constants
Define DEBUG constant: ✓
Define TRACE constant: ✓
Custom constants: COM_INTEROP_ENABLED=True
Example: Name1="Value1",Name2="Value2",Name3="Value3"
Generate serialization assemblies: Auto
Target CPU: x86
Target framework (all configurations): .NET Framework 3.5
Client-only Framework subset: ☐

Figure 3: The Advanced Compiler Settings dialog box

2. VB6 Interop UserControl projects must have a pre-build step that correctly locates the rc.exe file. This program is one of the Microsoft Win32® development tools installed as part of the Microsoft Windows software development kit (SDK). If it is in its default location, the pre-

build step should look like Figure 4.



```
"C:\Program Files\Microsoft SDKs\Windows\v6.1\Bin\x64\rc.exe" /r
"$(ProjectDir)InteropUserControl.rc"
```

Figure 4: The Pre-build event

You can learn how to create your own custom templates that incorporate these settings if you search for "Export Template Wizard" in the Visual Studio 2008 help file.

InteropForms

To use InteropForms in a Visual Basic 6.0 project, you must complete several steps:

1. Create a VB6 InteropForm Library project in Visual Studio 2008.
2. Create one or more forms that your Visual Basic 6.0 project will access.
3. Build the .NET Interop project, creating a new COM component.
4. Reference the new component from your Visual Basic 6.0 project.
5. Use the InteropForms in your Visual Basic 6.0 project.

For an online how-to video that describes this process, see http://msdn.microsoft.com /en-gb/vbasic/bb643843.aspx.

Create a VB6 InteropForm Library Project

Create a new Visual Basic .NET project by using the VB6 InteropForm Library template. This project contains a starter form and a module that contains some utility code. You can add any additional forms that you want to make available to Visual Basic 6.0 by using the VB6 InteropForm item template.

Create One or More Forms That Your Visual Basic 6.0 Project Will Access

The forms in your Interop Library project are standard .NET forms that can contain any available .NET controls. To expose functionality to your Visual Basic 6.0 environment, you must decorate the relevant properties, methods, constructors and events with the appropriate attributes (**InteropForm**, **InteropFormInitializer**, **InteropFormProperty**, **InteropFormMethod** and **InteropFormEvent**). The following code example (<InteropForm() > _

```
Public Class InteropForm1
    <InteropFormInitializer()> _
    Public Sub New(ByVal title As String)
        InitializeComponent()
        Me.Text = title
    End Sub

    Public Sub New()
        InitializeComponent()
    End Sub

    <InteropFormProperty()> _
    Public WriteOnly Property ShowToday() As Boolean
        Set(ByVal value As Boolean)
            calendar.ShowToday = value
        End Set
    End Property

    <InteropFormMethod()> _
    Public Sub SetCalendar(ByVal month As Integer, _
                           ByVal year As Integer)
        calendar.SetDate(New Date(year, month, 1))
    End Sub

    <InteropFormEvent()> _
    Public Event DateChanged As DateChangedEventHandler
    Public Delegate Sub DateChangedEventHandler _
    (ByVal StartDate As String, ByVal EndDate As String)

    Private Sub calendar_DateChanged _
    (ByVal sender As System.Object, _
     ByVal e As System.Windows.Forms.DateRangeEventArgs) _
     Handles calendar.DateChanged
        RaiseEvent DateChanged(e.Start.ToShortDateString(), _
                               e.End.ToShortDateString())
    End Sub
End Class
```

Listing 1) illustrates the use of these attributes.

```
<InteropForm()> _
Public Class InteropForm1
    <InteropFormInitializer()> _
    Public Sub New(ByVal title As String)
        InitializeComponent()
        Me.Text = title
    End Sub

    Public Sub New()
        InitializeComponent()
    End Sub

    <InteropFormProperty()> _
    Public WriteOnly Property ShowToday() As Boolean
        Set(ByVal value As Boolean)
            calendar.ShowToday = value
        End Set
    End Property

    <InteropFormMethod()> _
    Public Sub SetCalendar(ByVal month As Integer, _
                           ByVal year As Integer)
        calendar.SetDate(New Date(year, month, 1))
    End Sub

    <InteropFormEvent()> _
    Public Event DateChanged As DateChangedEventHandler
    Public Delegate Sub DateChangedEventHandler _
    (ByVal StartDate As String, ByVal EndDate As String)

    Private Sub calendar_DateChanged _
    (ByVal sender As System.Object, _
     ByVal e As System.Windows.Forms.DateRangeEventArgs) _
     Handles calendar.DateChanged
        RaiseEvent DateChanged(e.Start.ToShortDateString(), _
                               e.End.ToShortDateString())
    End Sub
End Class
```

Listing 1: Visual Basic .NET InteropForm

For simplicity, the Toolkit supports only a subset of Types as parameters or return types for Interop members. They are:

- **Integer (Int32)**
- **String**
- **Boolean**
- **Object**

The Toolkit also installs several code snippets to help you create these properties, methods and events (Figure 5).

interopsub + Tab	Insert a **Sub** method skeleton
interopfunc + Tab	Insert a **Function** method skeleton
interopprop + Tab	Insert a **Property** skeleton
interopinit + Tab	Insert a constructor
interopevent + Tab	Insert event definition

Figure 5: Code snippets

Build the .NET Interop Project

Before you build the project, you must generate the InteropForm wrapper classes. The Toolkit installs a menu entry in Visual Studio to perform this task (Figure 6).

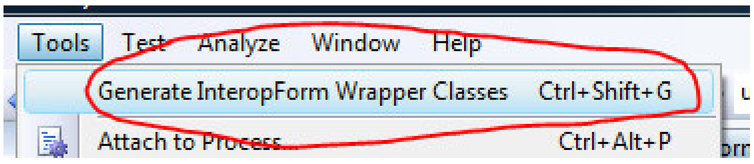
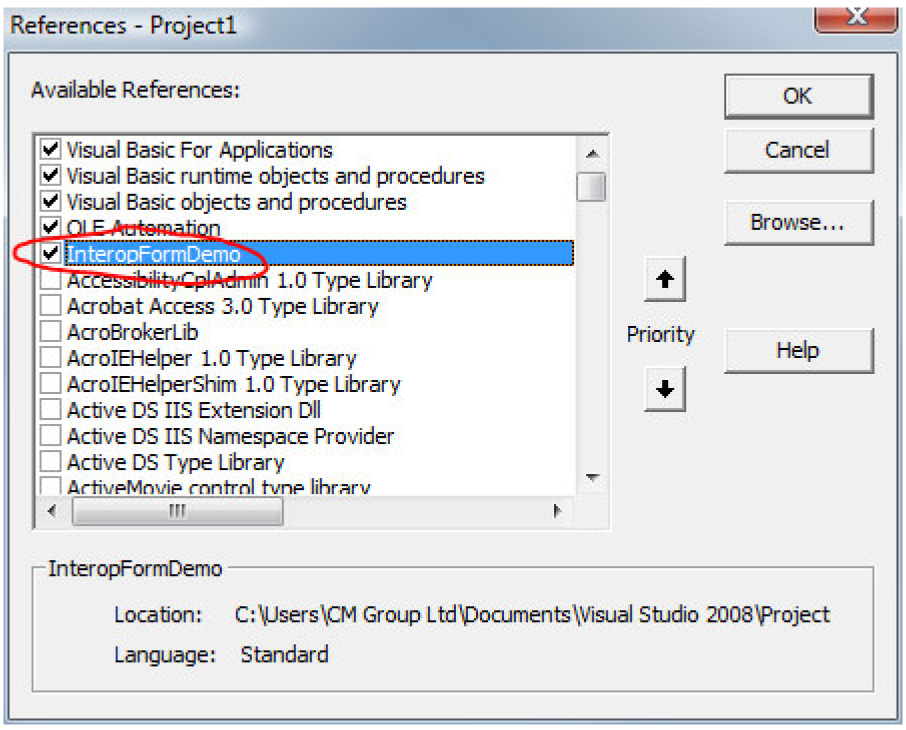

Tools
Test
Analyze
Window
Help
Generate InteropForm Wrapper Classes Ctrl+Shift+G
Attach to Process... Ctrl+Alt+P

Figure 6: Generate InteropForm wrapper classes

After you have generated the wrapper classes, building the project automatically registers any forms that are decorated with the **InteropForm** attribute as COM components.

Reference the New Component from Your Visual Basic 6.0 Project

Before you can use your InteropForms in Visual Basic 6.0, you must add a reference to the InteropForms Library component that your Visual Basic .NET project generates (Figure 7).



References - Project1

Available References:

- [✓] Visual Basic For Applications
- [✓] Visual Basic runtime objects and procedures
- [✓] Visual Basic objects and procedures
- [✓] OLE Automation
- [✓] InteropFormDemo
- [ ] AccessibilityCplAdmin 1.0 Type Library
- [ ] Acrobat Access 3.0 Type Library
- [ ] AcroBrokerLib
- [ ] AcroIEHelper 1.0 Type Library
- [ ] AcroIEHelperShim 1.0 Type Library
- [ ] Active DS IIS Extension Dll
- [ ] Active DS IIS Namespace Provider
- [ ] Active DS Type Library
- [ ] ActiveMovie control type library

InteropFormDemo

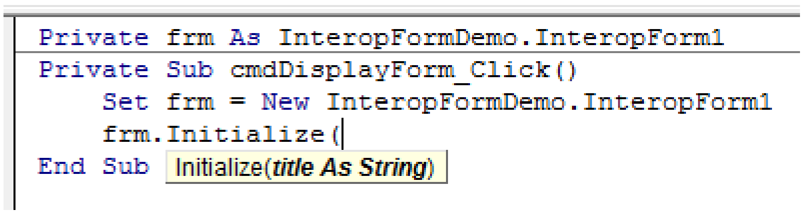
Location: C:\Users\CM Group Ltd\Documents\Visual Studio 2008\Project

Language: Standard

Figure 7: Visual Basic 6.0 project references

Use the InteropForms in Your Visual Basic 6.0 Project

After you have set the reference in Visual Basic 6.0 IntelliSense helps you to create code that interacts with your InteropForm (Figure 8).



```
Private frm As InteropFormDemo.InteropForm1
Private Sub cmdDisplayForm_Click()
    Set frm = New InteropFormDemo.InteropForm1
    frm.Initialize(
End Sub  Initialize(title As String)
```

Figure 8: IntelliSense

The following code example (Listing 2) illustrates the use of all of the exposed properties, methods and events from our Visual Basic .NET application.

```
Private WithEvents frm As _
InteropFormDemo.InteropForm1

Private Sub cmdDisplayForm_Click()
    Set frm = New _
    InteropFormDemo.InteropForm1
    frm.Initialize ("Loaded from VB6")

    frm.SetCalendar 7, 2009
    frm.ShowToday = False

    frm.Show vbModeless
End Sub

Private Sub frm_DateChanged( _
ByVal StartDate As String, _
ByVal EndDate As String)
    Label1 = StartDate
    Label2 = EndDate
End Sub
```

Listing 2: Visual Basic 6.0 InteropForm client

Additional Notes

There are a couple of other items to be aware of if you are using InteropForms:

- Bringing a Visual Basic 6.0 form to the front brings all Visual Basic 6.0 forms to the front.
- A Visual Basic 6.0 form that modally shows another Visual Basic 6.0 form works; however, the modal form only "flickers" to indicate modality when another Visual Basic 6.0 form, not a .NET form, is clicked.
- You cannot use an InteropForm as an MDI child form in Visual Basic 6.0. If you want to extend an MDI form, use an Interop UserControl on a Visual Basic 6.0 MDI child form and size it to fit the entire client area of the form.

Interop UserControls

You should use Interop UserControls instead of InteropForms, largely because they simplify the deployment process and offer some additional functionality such as being able to create MDI child forms. However, using Interop UserControls is just as easy as using InteropForms:

1. Create a VB6 Interop UserControl project in Visual Studio 2008.
2. Create one or more **UserControl** derived classes that your Visual Basic 6.0 forms will host.
3. Build the .NET Interop project. This creates a new COM component.
4. Add the new component to your Visual Basic 6.0 Toolbox.
5. Use the Interop UserControls component on your Visual Basic 6.0 forms.

For an online how-to video that describes this process, see http://msdn.microsoft.com /en-gb/vbasic/bb643844.aspx.

Create a VB6 Interop UserControl Project

Use the VB6 Interop UserControl template to create the new Visual Basic .NET project. This project contains a starter **UserControl** derived class, a “helpers” module and a manifest file.

Create One or More UserControls That Your Visual Basic 6.0 Forms Will Host

Create the user interface for your Interop UserControl in the same way that you would for a normal .NET **UserControl** derived class. The VB6 Interop UserControl item template that is provided contains all of the necessary infrastructure code to make the control work in Visual Basic 6.0. The template also defines several code regions, with example code, where you can define your own interop properties, methods and events. All public properties, methods and events will be visible to Visual Basic 6.0. Notice that there is no need for any attributes in this code, and that you can declare your events without the need to declare a delegate explicitly (Listing 3).

```
#Region "VB6 Events"

    Public Event DateChanged(ByVal StartDate As String, ByVal EndDate As String)

#End Region

#Region "VB6 Properties"

    Public WriteOnly Property ShowToday() As Boolean
        Set(ByVal value As Boolean)
            calendar.ShowToday = value
        End Set
    End Property

#End Region

#Region "VB6 Methods"

    Public Sub SetCalendar(ByVal month As Integer, _
                           ByVal year As Integer)
        calendar.SetDate(New Date(year, month, 1))
    End Sub
#End Region

Private Sub calendar_DateChanged _
     (ByVal sender As System.Object, _
      ByVal e As System.Windows.Forms.DateRangeEventArgs) _
      Handles calendar.DateChanged
        RaiseEvent DateChanged(e.Start.ToShortDateString(), _
        e.End.ToShortDateString())
End Sub
```
Listing 3: Visual Basic .NET InteropControl

Build the .NET Interop Project

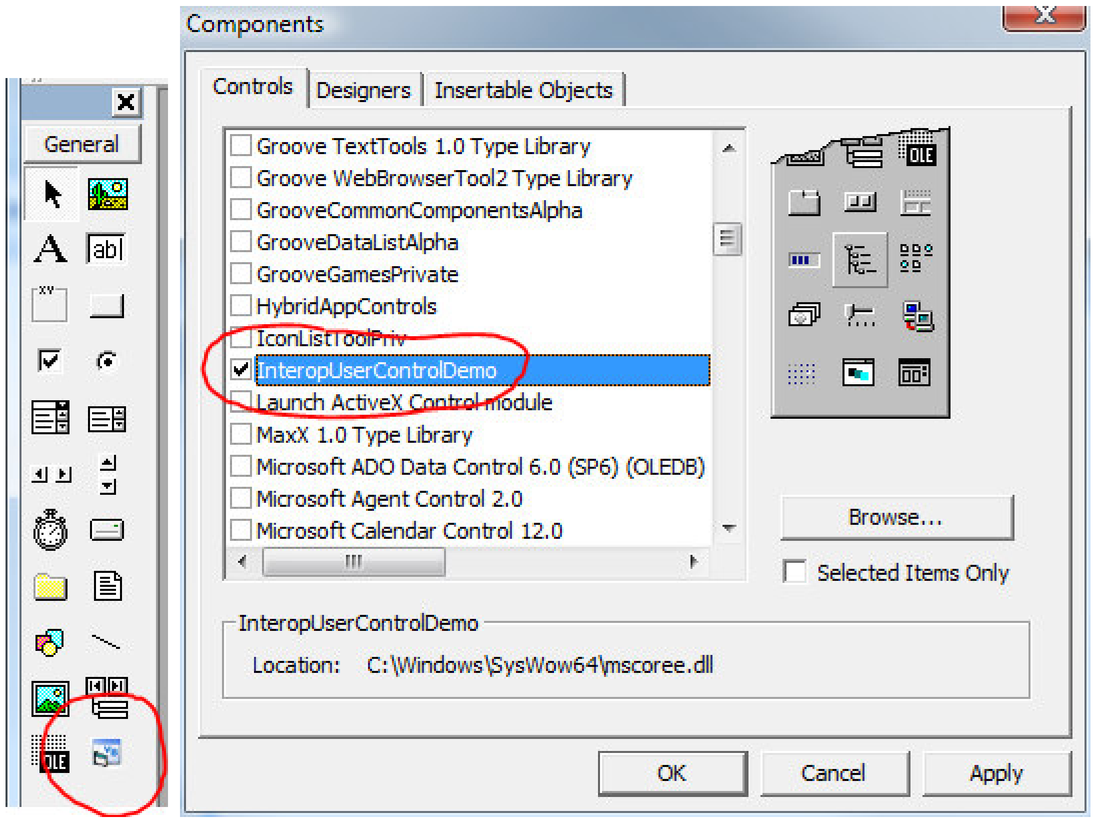
You can change the icon that is used to represent the control in the Visual Basic 6.0 Toolbox by editing the InteropUserControl.bmp file in the project. If you change the name of this .bmp file, you will also have to edit the InteropUserControl.rc file.

There are no additional steps to building your Interop UserControl project. If the build succeeds, it will automatically register your new control as a COM component.

If you are not using application events or sharing global events, you can remove the **Microsoft.InteropFormTools** reference from your project at this point. For the project to build successfully, you must delete the **Imports** statement and all of the code in the **My** namespace from the ActiveXControlHelpers.vb code file.

Reference the New Component from Your Visual Basic 6.0 Project

Before you can use your Interop UserControls in Visual Basic 6.0, you need to add the new control that your Visual Basic .NET project generated to the Visual Basic 6.0 Toolbox (Figure 9).


Components

Controls | Designers | Insertable Objects

- [ ] Groove TextTools 1.0 Type Library
- [ ] Groove WebBrowserTool2 Type Library
- [ ] GrooveCommonComponentsAlpha
- [ ] GrooveDataListAlpha
- [ ] GrooveGamesPrivate
- [ ] HybridAppControls
- [ ] IconListToolPriv
- [x] InteropUserControlDemo
- [ ] Launch ActiveX Control module
- [ ] MaxX 1.0 Type Library
- [ ] Microsoft ADO Data Control 6.0 (SP6) (OLEDB)
- [ ] Microsoft Agent Control 2.0
- [ ] Microsoft Calendar Control 12.0

InteropUserControlDemo

Location: C:\Windows\SysWow64\mscoree.dll

Figure 9: Adding a new control to the Toolbox

Use the Interop UserControls on Your Visual Basic 6.0 Forms

Add the new control to a Visual Basic 6.0 form. You can then write code that interacts with the Visual Basic .NET properties, methods and events (Listing 4).

```
Private Sub Form_Load()
    With interopUserControl
        .ShowToday = True
        .SetCalendar 2, 2007
    End With
End Sub

Private Sub interopUserControl_DateChanged _
(ByVal StartDate As String, _
ByVal EndDate As String)
    Label1 = StartDate
    Label2 = EndDate
End Sub
```

Listing 4: Visual Basic 6.0 InteropControl client

If you need to modify the Visual Basic .NET Interop UserControl you will have to close the Visual Basic 6.0 environment first. Otherwise, you will get build errors in Visual Studio .NET.

Additional Notes

- You can use Interop UserControls on Visual Basic 6.0 MDI child forms.
- To maintain binary compatibility, you should:
    - Add any new properties, methods and events to the bottom of the class.
    - Avoid changing the names of the assembly and the class names.

- Do not show a Visual Basic .NET form from your Interop UserControl.
- Do not use the **GotFocus** and **LostFocus** events in Visual Basic .NET. Use the **Enter** and **Leave** events instead.
- You cannot create control arrays of Interop UserControls.
- You cannot merge a .NET **MenuStrip** control's menu with a **WindowList** on an MDI form.
- You can only host Interop UserControls on Visual Basic 6.0 forms; you cannot place them on ActiveX controls or in any other type of host (for example, the 2007 Microsoft Office system).

Sharing Data Between Managed and Unmanaged Code

Some hybrid applications will need to exchange data between managed and unmanaged code. The Toolkit offers support for two mechanisms: using application-level events or using shared global data. Both of these mechanisms support a two-way exchange of data, from managed to unmanaged code, and from unmanaged to managed code.

To use this functionality, the Visual Basic .NET project must have a reference to the **Microsoft.InteropFormsTools** assembly (Figure 10).

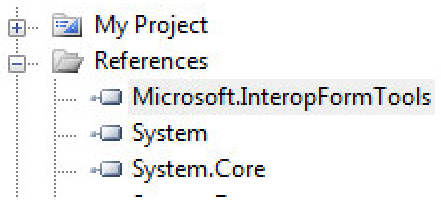
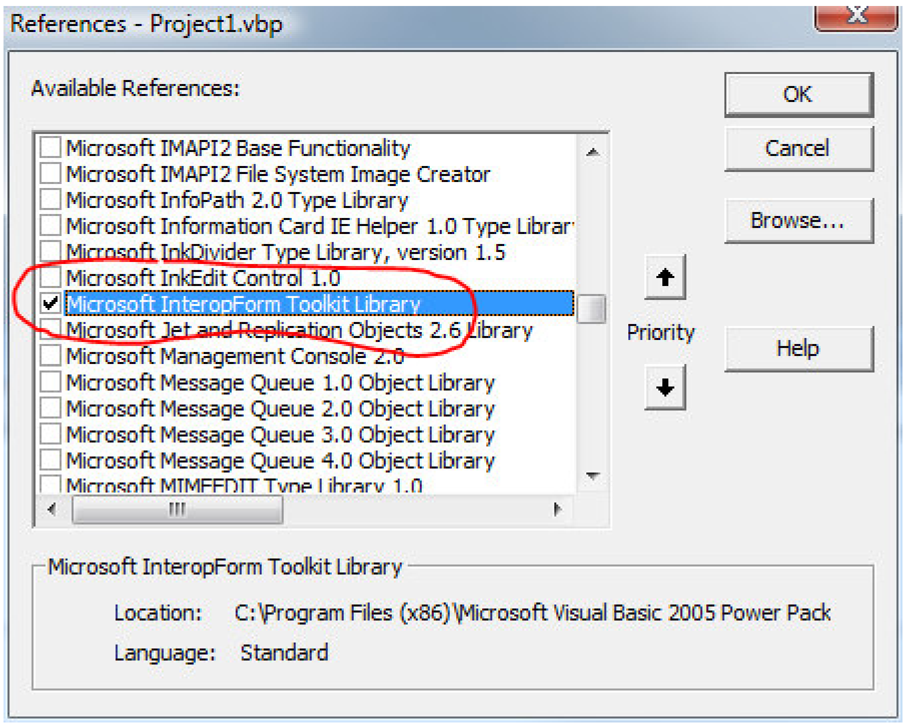


Figure 10: Referencing the Microsoft.InteropFormTools assembly

To use this functionality, the Visual Basic 6.0 project must have a reference to the Microsoft InteropForm Toolkit Library (Figure).


References - Project1.vbp

Available References:

- [ ] Microsoft IMAPI2 Base Functionality
- [ ] Microsoft IMAPI2 File System Image Creator
- [ ] Microsoft InfoPath 2.0 Type Library
- [ ] Microsoft Information Card IE Helper 1.0 Type Librar
- [ ] Microsoft InkDivider Type Library, version 1.5
- [ ] Microsoft InkEdit Control 1.0
- [✓] Microsoft InteropForm Toolkit Library
- [ ] Microsoft Jet and Replication Objects 2.6 Library
- [ ] Microsoft Management Console 2.0
- [ ] Microsoft Message Queue 1.0 Object Library
- [ ] Microsoft Message Queue 2.0 Object Library
- [ ] Microsoft Message Queue 3.0 Object Library
- [ ] Microsoft Message Queue 4.0 Object Library
- [ ] Microsoft MIMEEDIT Type Library 1.0

Microsoft InteropForm Toolkit Library

Location: C:\Program Files (x86)\Microsoft Visual Basic 2005 Power Pack

Language: Standard

Figure 11: Referencing the InteropForm Toolkit Library

The following Visual Basic 6.0 code example (Listing 5) shows the recommended pattern for using the InteropForm Toolkit Library. It shows how to initialize the **InteropToolbox** and notify the managed environment when the application starts up and shuts down. Use of this pattern makes sure that the managed environment cleanly disposes of any forms. If your Visual Basic 6.0 application does not have a **Sub Main** method, place this code in the startup form's **Load** and **Unload** events instead.

```
Public g_InteropToolbox As InteropToolbox

Sub Main()
    Set g_InteropToolbox = New InteropToolbox
    g_InteropToolbox.Initialize
    g_InteropToolbox.EventMessenger.RaiseApplicationStartedupEvent

    Dim frm1 As New Form1
    frm1.Show vbModal

    g_InteropToolbox.EventMessenger.RaiseApplicationShutdownEvent
End Sub
```

Listing 5: InteropToolbox initialization pattern

The following Visual Basic 6.0 code example (Listing 6) shows how to raise a custom application event and how to set some shared global data.

```
g_InteropToolbox.EventMessenger.RaiseApplicationEvent _
        "Connecting", "NorthwindDB"
g_InteropToolbox.Globals.Add "Username", "JohnD"
```

Listing 6: Visual Basic 6.0 using the InteropToolbox

The following Visual Basic .NET code in the InteropInfo.vb file from a VB6 InteropForm Library project (

```
Private Sub HandleAppEvents(ByVal type As String, ByVal params As Object) _
Handles _mesenger.ApplicationEventRaised
            If (type = "Connecting") Then
                ConnectDB(params.ToString())
            End If
End Sub
```

Listing 7), shows how to handle the event and how to retrieve the shared global data.

```
Private _toolbox As New InteropToolbox
Private WithEvents _mesenger As InteropEventMessenger = _toolbox.EventMessenger

Private Sub HandleAppEvents(ByVal type As String, ByVal params As Object) _
Handles _mesenger.ApplicationEventRaised
            If (type = "Connecting") Then
                ConnectDB(params.ToString())
            End If
End Sub
```

Listing 7: Visual Basic .NET code handling an unmanaged event

The following code example

```
(My.InteropToolbox.EventMessenger.RaiseApplicationEvent("SEVERE_ERROR", _
            "Connection Lost.")
```

Listing 8) demonstrates how to raise an application event in managed code and handle it in unmanaged code.

```
My.InteropToolbox.EventMessenger.RaiseApplicationEvent("SEVERE_ERROR", _
            "Connection Lost.")
```

Listing 8: Raising an interop event in managed code

The following code example (Listing 9) shows the Visual Basic 6.0 code that is required to handle this event.

```
Public g_InteropToolbox As InteropToolbox
Private WithEvents g_InteropMessenger As InteropEventMessenger
Private Sub Form_Load()
    Set g_InteropToolbox = New InteropToolbox
    g_InteropToolbox.Initialize
    Set g_InteropMessenger = g_InteropToolbox.EventMessenger
    g_InteropToolbox.EventMessenger.RaiseApplicationStartedupEvent
End Sub
Private Sub g_InteropMessenger_ApplicationEventRaised _
 (ByVal eventName As String, ByVal eventArgs As Variant)
If eventName = "SEVERE_ERROR" Then
        MsgBox eventArgs, vbCritical + vbOKOnly
    End If
End Sub
```

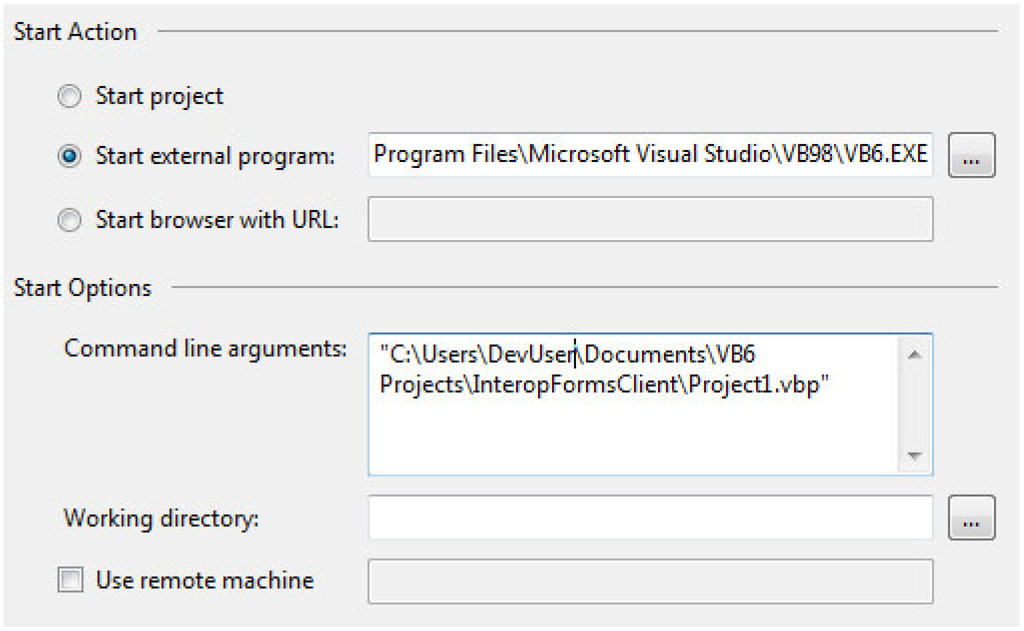
Listing 9: Handling an event raised in managed code in Visual Basic 6.0

Debugging

There are several ways that you could set up your development environment to facilitate debugging both your managed and unmanaged code. The approach described below is the best way to automate the debugging process and minimizes the number of manual steps that are required to initiate a debugging session. You will launch the debugging session from Visual Studio .NET and you will be able to set breakpoints in both managed and unmanaged code:

1. Start with the Visual Basic 6.0 IDE closed.

2. Set your Visual Basic .NET debug options as shown in Figure , with the correct path to VB6.EXE on your computer and the correct path to your Visual Basic 6.0 host project


Start Action

( ) Start project
(●) Start external program: Program Files\Microsoft Visual Studio\VB98\VB6.EXE
( ) Start browser with URL:

Start Options

Command line arguments: "C:\Users\DevUser\Documents\VB6 Projects\InteropFormsClient\Project1.vbp"

Working directory:

☐ Use remote machine

Figure 12: Debugger configuration

3. Set whatever breakpoints you need in Visual Basic .NET and start debugging.
4. The Visual Basic 6.0 IDE will launch with your project loaded. You can set whatever breakpoints you need in your Visual Basic 6.0 code. Then, start the Visual Basic 6.0 debugger. You are now debugging both your managed and unmanaged code.

It is important to remember that the Visual Basic 6.0 environment will close as soon as you stop the Visual Basic .NET debugger. Therefore, make sure that you save any changes to your Visual Basic 6.0 code as you go along.

Deployment

Deploying your hybrid application introduces some challenges when you need to create a single installer for your hybrid application. There are several installation technologies to choose from: the Visual Basic 6.0 Package and Deployment Wizard, a Visual Studio .NET Setup project, ClickOnce deployment and Registration-free (Reg-free) COM. A deployment solution may make use of one or several of these technologies.

For an online how-to video that describes this process, see http://msdn.microsoft.com /en-gb/vbasic/bb643845.aspx.

Scenario 1

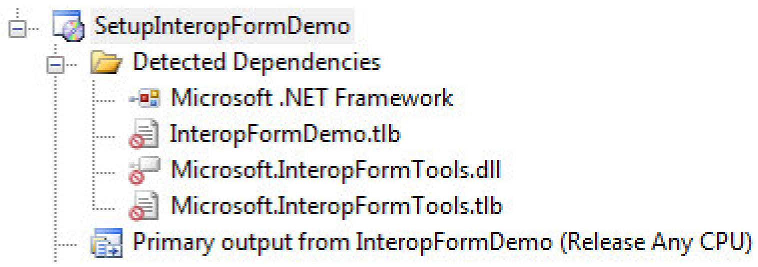
This scenario covers deploying a hybrid application by using a Visual Studio .NET Setup project. The application uses InteropForms.

For the purposes of this scenario, these are the assumptions made:

1. The target operating system is Windows® XP or Windows Vista®. This means that the Visual Basic 6.0 runtime is already present.

2. The .NET portion of the application uses the .NET Framework 3.5 Service Pack 1 (SP1). The framework **is not** pre-installed on the end user's computer.

Create a new Setup Project in Visual Studio .NET, and then add the library that contains your InteropForms. Visual Studio will detect several dependencies, but we should exclude these files and add them as prerequisites instead (Figure).

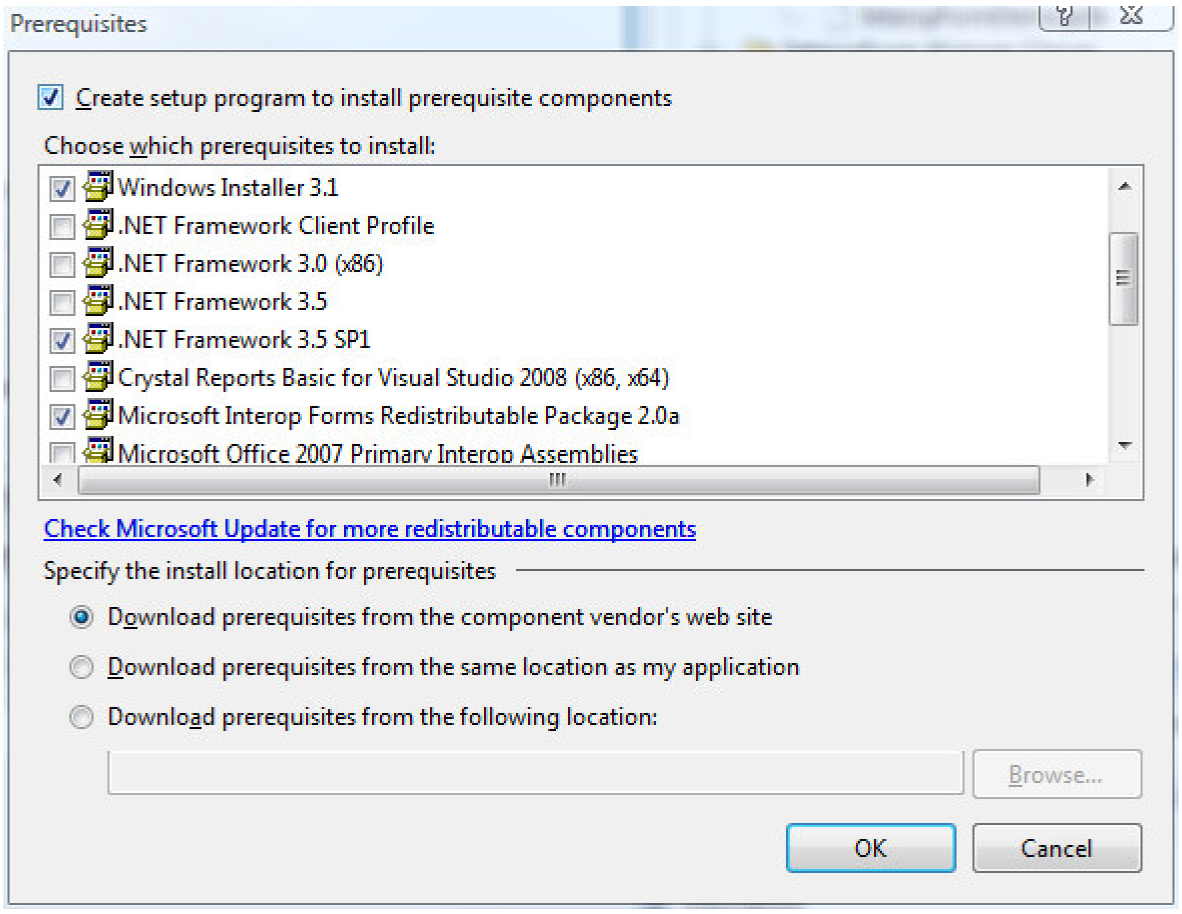


```
SetupInteropFormDemo
  Detected Dependencies
    Microsoft .NET Framework
    InteropFormDemo.tlb
    Microsoft.InteropFormTools.dll
    Microsoft.InteropFormTools.tlb
  Primary output from InteropFormDemo (Release Any CPU)
```

Figure 13: Setup Project contents

You must make sure that the library that contains your InteropForms registers as a COM component on the target computer, so set the **Register** property of this file to **vsdrpCOM**.

Access the **Prerequisites** dialog box from the Setup Project properties and select any prerequisites. Figure shows the Windows Installer 3.1, the .NET Framework 3.5 SP1 and the Microsoft Interop Forms Redistributable Package. This dialog box also enables you to set where the installer should look for the prerequisites: on the Microsoft Web site, with your install files or in a custom location. Figure shows that the installer will download the prerequisites from the Microsoft Web site.


Prerequisites

[✓] Create setup program to install prerequisite components

Choose which prerequisites to install:

- [✓] Windows Installer 3.1
- [ ] .NET Framework Client Profile
- [ ] .NET Framework 3.0 (x86)
- [ ] .NET Framework 3.5
- [✓] .NET Framework 3.5 SP1
- [ ] Crystal Reports Basic for Visual Studio 2008 (x86, x64)
- [✓] Microsoft Interop Forms Redistributable Package 2.0a
- [ ] Microsoft Office 2007 Primary Interop Assemblies

Check Microsoft Update for more redistributable components

Specify the install location for prerequisites

- ● Download prerequisites from the component vendor's web site
- ○ Download prerequisites from the same location as my application
- ○ Download prerequisites from the following location:

Figure 14: The selection in the Prerequisites dialog box

The last step is to add your Visual Basic 6.0 .exe file and any other Visual Basic 6.0 components to your Setup Project. Be sure to mark any Visual Basic 6.0 COM components for COM registration.

Before you build your Setup Project, create any required **Start** menu items or desktop shortcuts by using the File System Editor.

For more information about creating Windows Installer deployment packages, search the Visual Studio 2008 help file for “Windows Installer Deployment”.

Scenario 2

This scenario covers deploying a hybrid application that uses Interop UserControls. It uses Reg-free COM to deploy your .NET Interop Control library as part of a standard Visual Basic 6.0 Setup and Deployment Wizard package.

For the purposes of this scenario, these are the assumptions made:

1. The target operating system is Windows XP or Windows Vista. This means that you can use Reg-free COM.
2. The operating system already has the .NET Framework 3.5 SP1 installed. This simplifies the scenario, but it is possible to include the installation of the .NET Framework as part of the deployment package.

Reg-free COM is a technology in Windows XP and Windows Vista that enables you to deploy COM components without having to register them. This is one of the advantages of using Interop UserControls instead of InteropForms because InteropForms do not work with Reg-free COM.

Reg-free COM depends on two manifest files. The VB6 Interop UserControl project template creates the first one for you (

```
<!-- You don't need to worry about anything in this file unless you're
     using registration-free COM.
     There should be an appropriate <clrclass> section for every InteropUserControl
     defined in the project -->
<assembly xmlns="urn:schemas-microsoft-com:asm.v1"
  manifestVersion="1.0">
<assemblyIdentity
            type="win32"
            name="InteropUserControlDemo"
            version="1.0.0.0" />
<clrClass
            clsid="{9acd49fc-5e4c-4791-9961-289cc8bafee2}"
            progid="InteropUserControlDemo.InteropUserControl"
            threadingModel="Both"
            name="InteropUserControlDemo.InteropUserControl" >
</clrClass>
</assembly>
```

Listing 10). The default name of this manifest file is InteropUserControl.manifest.

```
<!-- You don't need to worry about anything in this file unless you're
     using registration-free COM.
     There should be an appropriate <clrclass> section for every InteropUserControl
     defined in the project -->
<assembly xmlns="urn:schemas-microsoft-com:asm.v1"
  manifestVersion="1.0">
<assemblyIdentity
            type="win32"
            name="InteropUserControlDemo"
            version="1.0.0.0" />
<clrClass
            clsid="{9acd49fc-5e4c-4791-9961-289cc8bafee2}"
            progid="InteropUserControlDemo.InteropUserControl"
            threadingModel="Both"
            name="InteropUserControlDemo.InteropUserControl" >
</clrClass>
</assembly>
```

Listing 10: Interop UserControl embedded manifest

If you have changed the name of your control, you may need to change the values of the **clrClass** element attributes: **clsid**, **progid** and **name**. The build process embeds this manifest into the generated assembly.

You must create the second manifest file manually (

```
<?xml version="1.0" encoding="UTF-8"
standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  <assemblyIdentity type="win32" name="Project1" version="1.0.0.0"
    processorArchitecture="x86" />
  <dependency>
    <dependentAssembly>
      <assemblyIdentity type="win32" name="InteropUserControlDemo"
        version="1.0.0.0" />
    </dependentAssembly>
  </dependency>
</assembly>
```

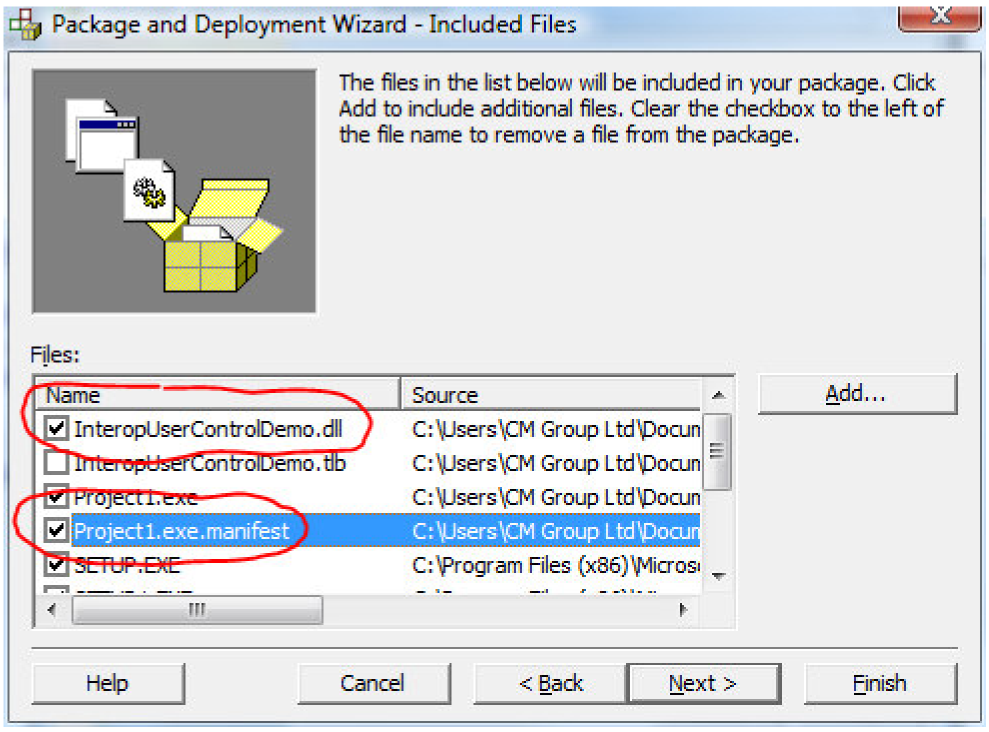
Listing 11). Deploy this file to the same folder as your Visual Basic 6.0 .exe file and make sure that it follows this naming convention: if the name of your Visual Basic 6.0 .exe file is **MyApp.exe**, the name of the manifest file must be **MyApp.exe.manifest**.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  <assemblyIdentity type="win32" name="Project1" version="1.0.0.0"
    processorArchitecture="x86" />
  <dependency>
    <dependentAssembly>
      <assemblyIdentity type="win32" name="InteropUserControlDemo"
        version="1.0.0.0" />
    </dependentAssembly>
  </dependency>
</assembly>
```

Listing 11: Application manifest

In the example above, **MyApp** is the name of the Visual Basic 6.0 .exe file and **InteropDemoControl** is the name of the .NET library (.dll) file. The version attributes contain the correct version numbers of the .exe file and the library.

You should include the library and manifest in the Visual Basic 6.0 Package and Deployment Wizard script (Figure).


Package and Deployment Wizard - Included Files

The files in the list below will be included in your package. Click Add to include additional files. Clear the checkbox to the left of the file name to remove a file from the package.

Files:

Name	Source
☑ InteropUserControlDemo.dll	C:\Users\CM Group Ltd\Docun
☐ InteropUserControlDemo.tlb	C:\Users\CM Group Ltd\Docun
☑ Project1.exe	C:\Users\CM Group Ltd\Docun
☑ Project1.exe.manifest	C:\Users\CM Group Ltd\Docun
☑ SETUP.EXE	C:\Program Files (x86)\Micros

Figure 15: Visual Basic 6.0 Package and Deployment Wizard

Scenario 3

This scenario covers deploying a hybrid application that uses Interop UserControls. It uses ClickOnce deployment to install the application and, if necessary, the .NET Framework.

For the purposes of this scenario, these are the assumptions made:

1. The target operating system is Windows XP or Windows Vista. This means that you can use Reg-free COM. ClickOnce deployment does not support COM registration. This also means that the Visual Basic 6.0 runtime is already present.
2. The operating system does not have the .NET Framework 3.5 SP1 installed. ClickOnce will manage installing any prerequisites.

The key to this approach is to create a small .NET application that will launch the Visual Basic 6.0 application on the target computer. This project contains the Visual Basic 6.0 .exe file, its manifest file for Reg-free COM deployment (see Scenario 2) and your **InteropUserControls** assembly (Figure).

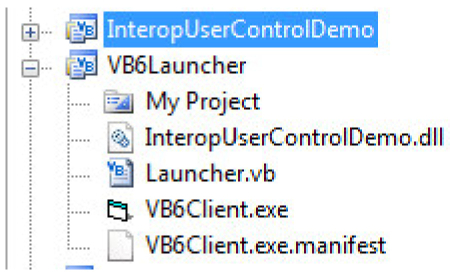


Figure 16: ClickOnce project

You can add these files to the project by using the project's **Add Existing Item** menu option. These files must also have their build type set to **Content**. The code in Launcher.vb launches the Visual Basic 6.0 .exe file (Listing 12).

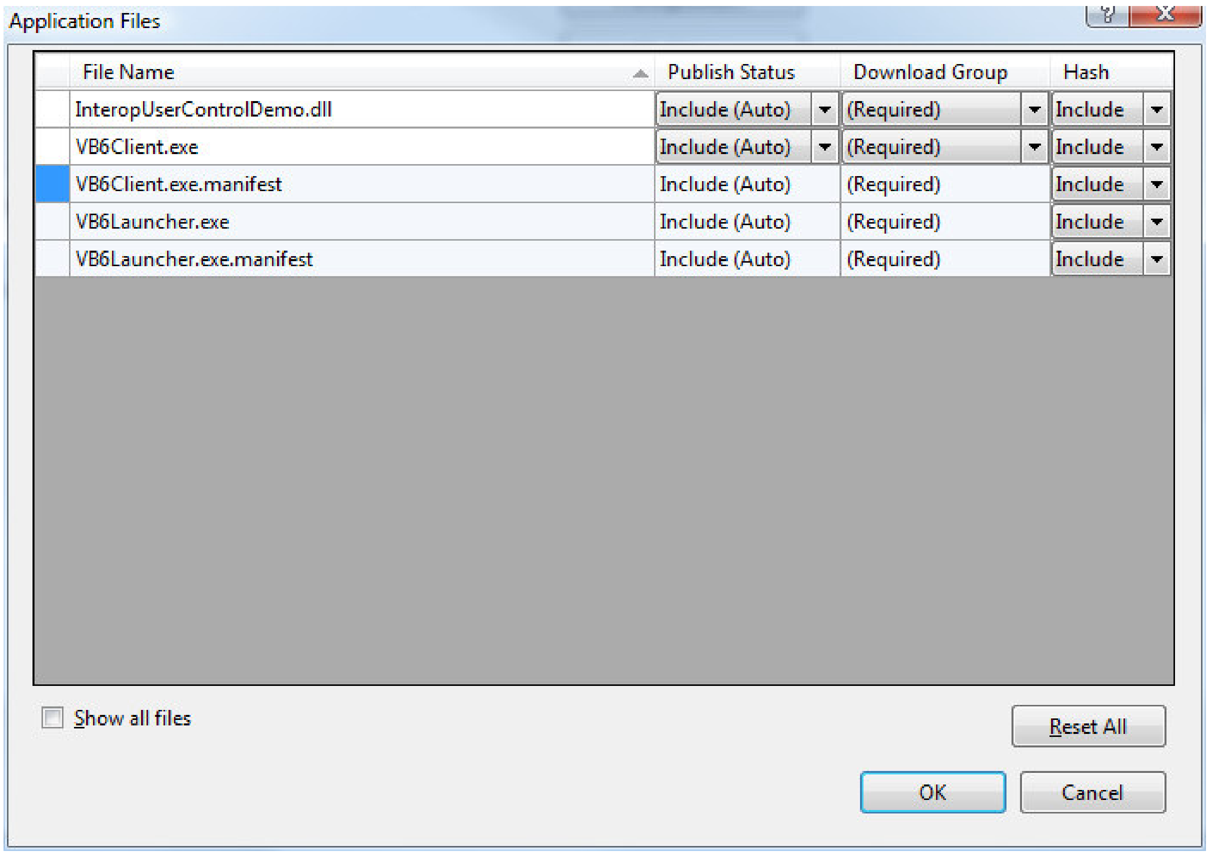
```
Module Launcher

    Sub Main()
        Process.Start("Project1.exe")
    End Sub

End Module
```
Listing 12: Visual Basic .NET Launcher

If you change the application type from Console Application to Windows Forms Application, the console window will not display when you launch the application.

You can create the ClickOnce deployment files by using the **Publish** tab on the project's property sheet. Here, you can check that you will be deploying the correct files (Figure), check that ClickOnce will install any prerequisites and determine where to host the deployment files.


Application Files

	File Name	Publish Status	Download Group	Hash
	InteropUserControlDemo.dll	Include (Auto)	(Required)	Include
	VB6Client.exe	Include (Auto)	(Required)	Include
	VB6Client.exe.manifest	Include (Auto)	(Required)	Include
	VB6Launcher.exe	Include (Auto)	(Required)	Include
	VB6Launcher.exe.manifest	Include (Auto)	(Required)	Include

Figure 17: ClickOnce publishing

For more information about publishing applications by using ClickOnce deployment, search the Visual Studio 2008 help file for "ClickOnce".

Summary

Using the Interop Forms Toolkit 2.0, you can extend and upgrade your existing Visual Basic 6.0 applications to Visual Basic .NET. You can use features of the .NET Framework to add business value to the application without either having to upgrade the entire code base or get involved in detailed interoperability issues. You can also use the Toolkit to support a phased upgrade, one form at a time, to the .NET platform. Furthermore, the Toolkit supports a range of deployment scenarios, making it easy to deploy a hybrid application to an end user's desktop.

Where possible, you should use the Interop UserControls instead of the InteropForms. This approach offers a simpler coding model and more functionality, for example, supporting Visual Basic 6.0 MDI applications. Additionally, Interop UserControls offer more flexibility, enabling you to work at a sub-form level with more opportunities for code reuse. The Interop UserControls approach also opens up the options of deploying your application by using Reg-free COM and ClickOnce technologies.

Resources

"Deploying Applications with the InteropForms 2.0 toolkit"
The Visual Basic Team Blog

http://blogs.msdn.com/vbteam/archive/2007/06/04
/deploying-applications-with-the-interopforms-2-0-toolkit.aspx

"InteropForms ToolKit - Visual Studio 2008 edition (Todd Apley)"
The Visual Basic Team Blog
http://blogs.msdn.com/vbteam/archive/2008/03/05
/interopforms-toolkit-visual-studio-2008-edition.aspx

"Visual Basic Interop and Upgrade"
MSDN Forums
http://social.msdn.microsoft.com/Forums/en-US/vbinterop/threads/

Visual Basic 6.0 Resource Center
http://msdn.microsoft.com/en-gb/vbrun/default.aspx

"How Do I" Videos – Visual Basic
http://msdn.microsoft.com/en-gb/vbasic/bb466226.aspx

"Goto 100 - Development with Visual Basic"
(UK blog about development with Visual Basic)
http://blogs.msdn.com/goto100/

"Sales Support: A Hybrid Visual Basic 6.0 & Visual Basic 2005 Application"
http://msdn.microsoft.com/en-us/library/cc300139(VS.80).aspx

"Extending Visual Basic 6 ActiveX EXEs With Visual Basic 2005 and the Interop Forms Toolkit"
http://msdn.microsoft.com/en-gb/library/bb397409(VS.80).aspx

"Upgrading Visual Basic 6.0 Applications to Visual Basic .NET and Visual Basic 2005"
(This document focuses on upgrading Visual Basic 6.0 applications and was written before the Interop Forms Toolkit was released.)
http://msdn.microsoft.com/en-gb/library/aa480541.aspx

VB Upgrade Companion from ArtinSoft
http://www.artinsoft.com/

VB Migration Partner from Code Architects
http://www.vbmigration.com/Default.aspx