Part II

Upgrading Applications

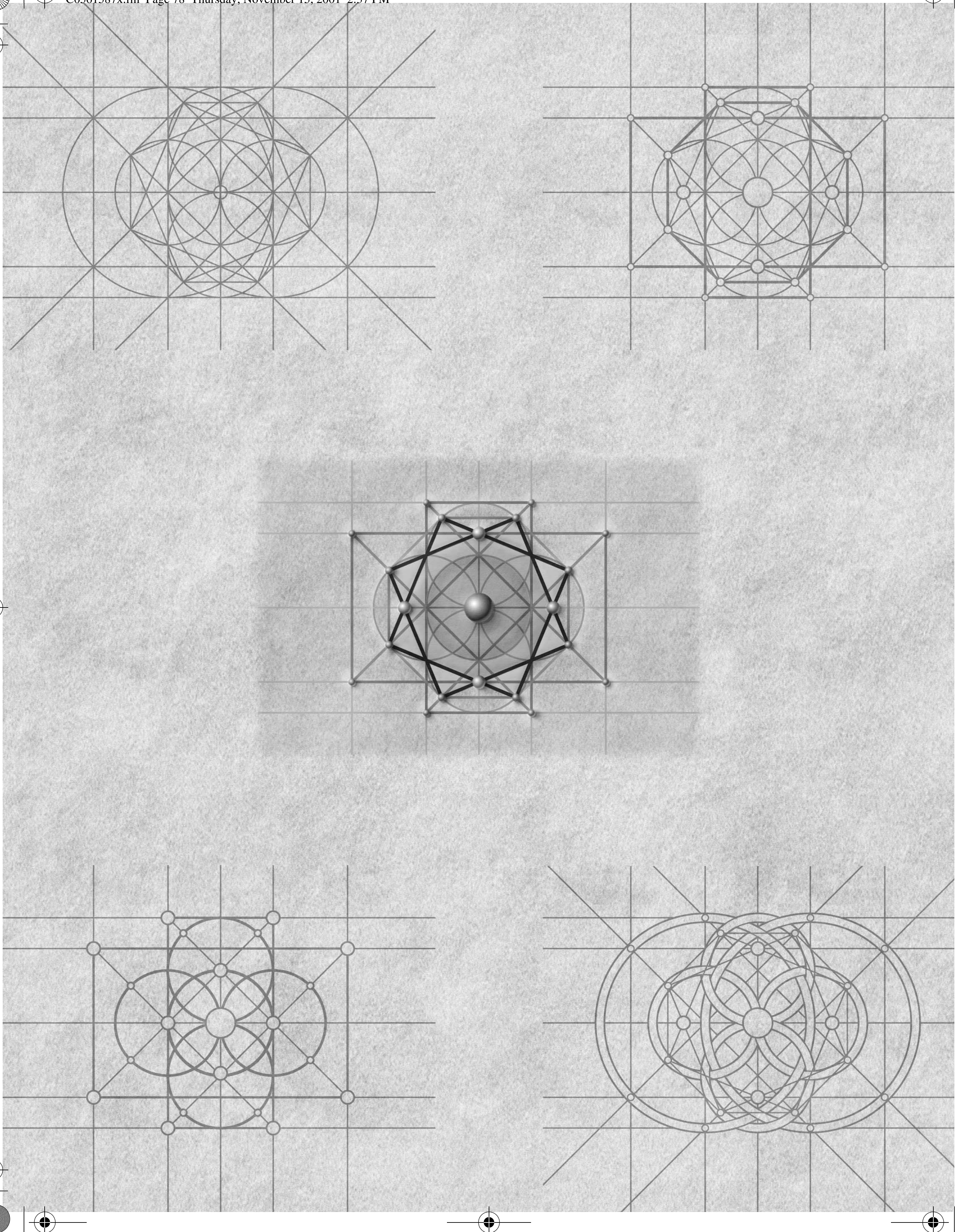
5 Your First Upgrade 79

6 Common Tasks in Visual Basic .NET 101

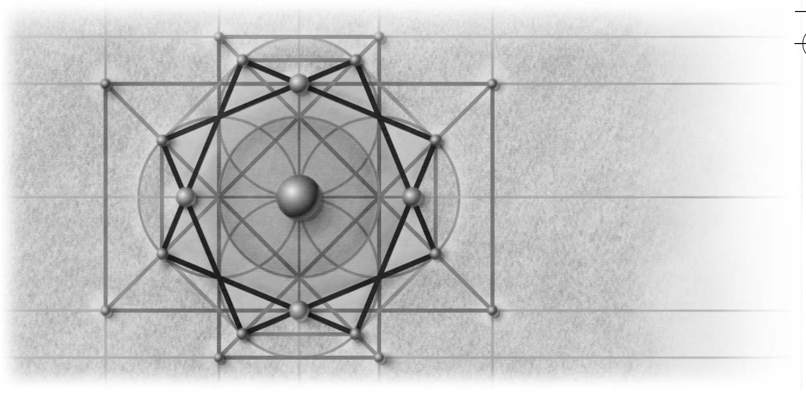
7 Upgrading Wizard Ins and Outs 117

8 Errors, Warnings, and Issues 149

9 Using Visual Basic 6 with Visual Basic .NET: COM Interop 175



5



Your First Upgrade

In Part I, we explored the differences between Microsoft Visual Basic 6 and Visual Basic .NET, looked at the different upgrading options, and covered how to prepare your Visual Basic 6 application for the upgrade to Visual Basic .NET. Now it's time to begin talking about the how-to of upgrading. This chapter walks you through the process of using the Upgrade Wizard to upgrade a simple project and examining what your new Visual Basic .NET project looks like. It also introduces you to techniques for fixing problems with the upgraded project and discusses some advanced upgrading techniques such as upgrading project groups and using the VB6 Snippet Upgrade add-in.

Upgrade Walkthrough

Let's start by creating a simple project in Visual Basic 6. We will assume that you have Visual Basic 6 and Visual Basic .NET installed on the same machine. Having both versions installed is actually a recommended configuration. Visual Basic .NET installs to a different directory than Visual Basic 6. If Visual Basic 6 is already installed, it is left intact. Visual Basic .NET does not overwrite any Visual Basic 6 files—the two versions can be installed and run side by side. The order of installation doesn't matter either; you can set up Visual Basic 6 after installing Visual Basic .NET or vice versa. The benefits of installing both products on the same machine are twofold:

- It allows you to ensure that your project actually runs. (If the project doesn't work in Visual Basic 6, it's unlikely that it will work after upgrading.)

- Visual Basic 6 Setup installs common ActiveX controls and libraries. These are not distributed with Visual Basic .NET. The easiest way to get them on the machine is to install Visual Basic 6.

We are going to create a simple Visual Basic 6 project, prjFirstUpgrade, and upgrade it to Visual Basic .NET. (You can find a completed version of this project on the companion CD that accompanies this book.) The project we are about to develop generates a new random number between 0 and 10 every second and shows the number in a TextBox. It continues to do this until you click a CommandButton to stop it. Let's get started. Launch Visual Basic 6, and follow these steps to create the project:

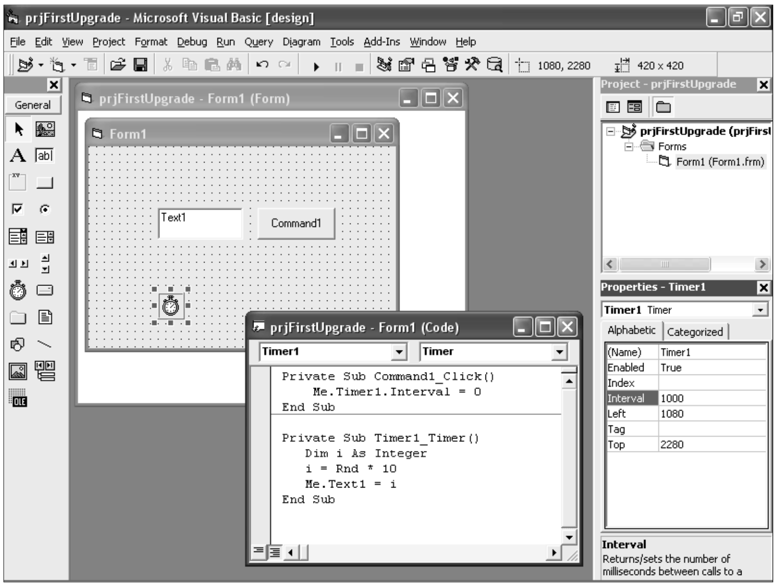
1. Create a new Standard EXE project.
2. After Visual Basic 6 creates the project, you'll see that Form1 is displayed in the integrated development environment (IDE). Add a TextBox, a CommandButton, and a Timer control to Form1.
3. Select the Timer control. In the Properties grid, set the interval to 1000.
4. Double-click the Command1 CommandButton, and add this code to the *Command1_Click* procedure:

   ```
   Private Sub Command1_Click()
      Me.Timer1.Interval = 0
   End Sub
   ```

5. Now return to the Form window, and double-click the Timer control to create the *Timer1_Timer* event. Add the following code to the event:

   ```
   Private Sub Timer1_Timer()
      Dim i As Integer
      i = Rnd * 10
      Me.Text1 = i
   End Sub
   ```

6. Save the project, using the default name for the form: Form1. We'll name the project something meaningful: prjFirstUpgrade. Make sure you save the project into a directory on your local computer, rather than to a network share. This step ensures that Visual Basic .NET has enough security permissions to upgrade and run the project. The form and code should look similar to Figure 5-1.


prjFirstUpgrade - Microsoft Visual Basic [design]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
1080, 2280
420 x 420
General
prjFirstUpgrade - Form1 (Form)
Form1
Text1
Command1
Project - prjFirstUpgrade
prjFirstUpgrade (prjFirst
Forms
Form1 (Form1.frm)
prjFirstUpgrade - Form1 (Code)
Timer1
Timer
Properties - Timer1
Timer1 Timer
Alphabetic
Categorized

```
Private Sub Command1_Click()
    Me.Timer1.Interval = 0
End Sub

Private Sub Timer1_Timer()
   Dim i As Integer
   i = Rnd * 10
   Me.Text1 = i
End Sub
```

(Name)	Timer1
Enabled	True
Index	
Interval	1000
Left	1080
Tag	
Top	2280

Interval
Returns/sets the number of milliseconds between calls to a

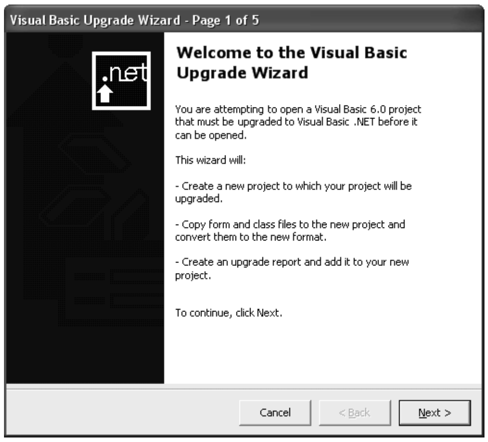
Figure 5-1 Random number generator in Visual Basic 6.

When you run the project, you'll see that the TextBox updates every second with a new random number. It stops when you click Command1.

Now that we've created a Visual Basic 6 project, let's upgrade it to Visual Basic .NET. You can leave Visual Basic 6 running, with the project still loaded—doing so doesn't interfere with the upgrade. Start Visual Basic .NET, and follow these steps to upgrade your project:

1. In Visual Basic .NET, from the File menu, choose Open and then Project to display the Open Project dialog box. Navigate to the location where you saved your Visual Basic 6 project file, select prjFirstUpgrade.vbp, and click the Open button to upgrade prjFirstUpgrade using the Upgrade Wizard.
2. The first page of the wizard contains welcome information, as shown in Figure 5-2. After you've read the welcome information, click Next to move to the second page.

C0561587x.fm Page 82 Thursday, November 15, 2001 2:37 PM


Visual Basic Upgrade Wizard - Page 1 of 5

Welcome to the Visual Basic Upgrade Wizard

You are attempting to open a Visual Basic 6.0 project that must be upgraded to Visual Basic .NET before it can be opened.

This wizard will:

- Create a new project to which your project will be upgraded.

- Copy form and class files to the new project and convert them to the new format.

- Create an upgrade report and add it to your new project.

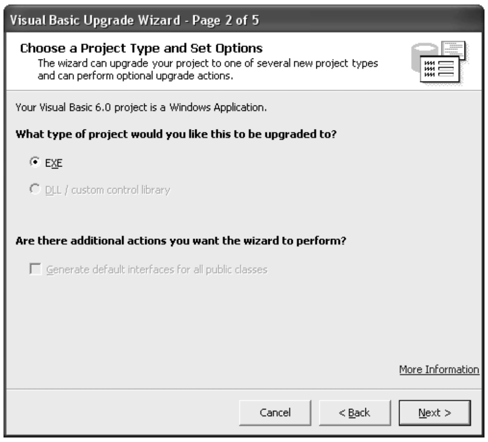
To continue, click Next.

Figure 5-2 Welcome screen of the Upgrade Wizard.

3. The second page, shown in Figure 5-3, is where you set options for the upgrade. These options may be disabled depending on the project type, and in most cases, you will leave them at their default setting, but let's examine them more closely to see when you would use them. There are two options. The first determines what type of Visual Basic .NET project to upgrade to. As you would expect, EXEs upgrade to EXEs, and DLLs upgrade to DLLs; for standard EXE and DLL projects, the upgrade type is determined automatically. With EXE server projects, however, you have some flexibility. These projects act like a combination of an EXE and a DLL. When upgrading these projects, you have the choice of converting them to a Visual Basic .NET EXE or to a Visual Basic .NET DLL. Since our project is a standard EXE, the first choice (EXE) is automatically selected. The second choice, DLL/Custom Control Library, is disabled, and for this reason the second option, Generate Default Interfaces For All Public Classes, is disabled also.

This option is available only for DLL and EXE server projects. You should select this option only if other projects use the *Implements* keyword to implement classes from within your DLL or EXE server. Selecting this option causes the wizard to create an interface for every public class in your project.

Click Next to move to page 3.


Visual Basic Upgrade Wizard - Page 2 of 5

Choose a Project Type and Set Options
The wizard can upgrade your project to one of several new project types and can perform optional upgrade actions.

Your Visual Basic 6.0 project is a Windows Application.

What type of project would you like this to be upgraded to?

[✓ EXE]

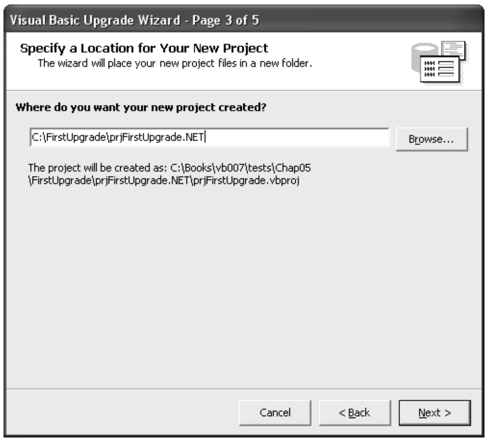
DLL / custom control library

Are there additional actions you want the wizard to perform?

Generate default interfaces for all public classes

Figure 5-3 Choosing a project type.

4. On the third page, shown in Figure 5-4, you specify the destination directory for the upgraded project. The project is not moved to this location. Instead, the Upgrade Wizard leaves your Visual Basic 6 project untouched and creates a brand new Visual Basic .NET project in the directory you specify here. By default, the wizard suggests placing the upgraded project in a new directory called <*project-name*>.NET. You can use the text box and the Browse button to change the location of this directory, but in most cases you should simply accept the default name. You can always move the directory later simply by copying it to a different location.


Visual Basic Upgrade Wizard - Page 3 of 5

Specify a Location for Your New Project
The wizard will place your new project files in a new folder.

Where do you want your new project created?

C:\FirstUpgrade\prjFirstUpgrade.NET

The project will be created as: C:\Books\vb007\tests\Chap05\FirstUpgrade\prjFirstUpgrade.NET\prjFirstUpgrade.vbproj

Figure 5-4 Specifying a destination directory.

If the destination directory already contains files, these will be deleted to make room for the new project. With our soon-to-be-upgraded Visual Basic 6 project, let's accept the suggested name, prjFirstUpgrade.NET. Click Next to move to page 4.

5. Because the prjFirstUpgrade.NET folder doesn't exist yet, a warning message appears asking you if you want to create it. Click Yes.
6. On page 4, the wizard tells you that it is ready to perform the upgrade. Click Next to generate the upgraded project.

Status text and a progress bar show you what the Upgrade Wizard is processing and give you a visual indicator of the time left. The upgrade is not instantaneous; it can take anywhere from a minute (for a small project such as ours) to several hours (for a project with hundreds of forms and classes). After the wizard has finished, your project opens in Visual Basic .NET. The result will be similar to Figure 5-5, depending on what windows you already have open in Visual Basic .NET.

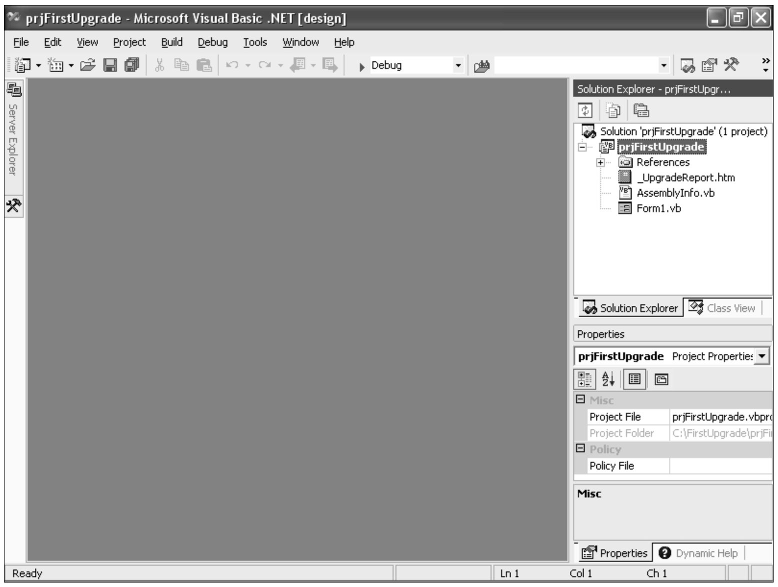
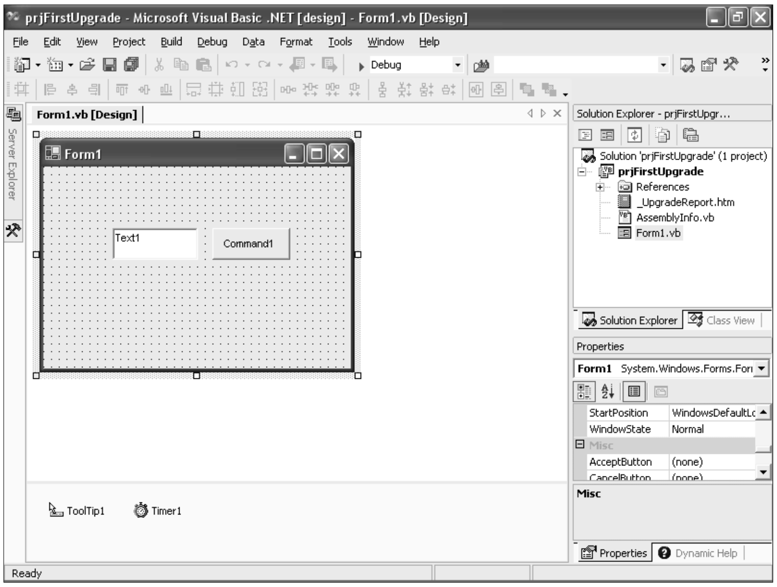


Figure 5-5 The upgraded project opens in Visual Basic .NET.

What Just Happened?

Let's look at the upgraded project to see what the wizard did with our Visual Basic 6 project. You should see the Solution Explorer window in the top right corner of the IDE. (If the window is not visible, press Ctrl+Alt+L to see it.) Double-click Form1.vb to open the upgraded form. As you can see in Figure 5-6, the project looks exactly as it did in Visual Basic 6, with the exception that the Timer is now in the component tray beneath the form. Why has it moved? Nonvisual controls such as Timers, ToolTips, and Menus are now displayed in the component tray instead of on the form itself.


prjFirstUpgrade - Microsoft Visual Basic .NET [design] - Form1.vb [Design]
File Edit View Project Build Debug Data Format Tools Window Help
Debug
Form1.vb [Design]
Server Explorer
Form1
Text1
Command1
ToolTip1
Timer1
Ready
Solution Explorer - prjFirstUpgr...
Solution 'prjFirstUpgrade' (1 project)
prjFirstUpgrade
References
_UpgradeReport.htm
AssemblyInfo.vb
Form1.vb
Solution Explorer
Class View
Properties
Form1 System.Windows.Forms.Fori
Misc
Properties
Dynamic Help

StartPosition	WindowsDefaultLc
WindowState	Normal
Misc	
AcceptButton	(none)
CancelButton	(none)

Figure 5-6 Upgraded form.

Now let's try running the project. Press F5, just as you would in Visual Basic 6. The first time you run an upgraded project in Visual Basic .NET, you are prompted to save the solution file. A **solution file** is similar to a group file in Visual Basic 6. A solution contains one or more projects, just as a group can contain one or more projects. In Visual Basic .NET, since every project must be part of a solution, one is created for you. Press Enter to save it with the default name. The project then runs, and Form1 begins showing a new random number every second, as shown in Figure 5-7.

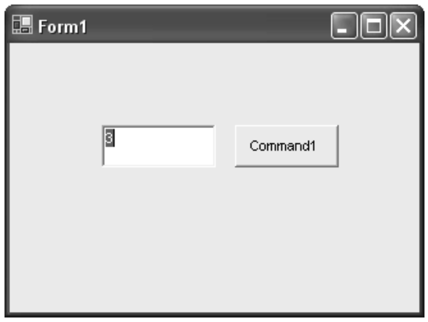
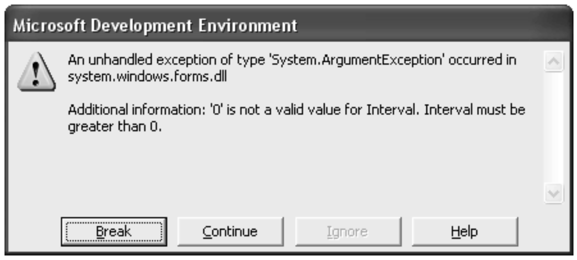


Figure 5-7 Generating random numbers in the upgraded project.

Click Command1 to stop the Timer. The Timer indeed stops, but not in the way we intended. Instead, the program breaks with an exception: "'0' is not a valid value for Interval. Interval must be greater than 0," as shown in Figure 5-8.


Microsoft Development Environment

An unhandled exception of type 'System.ArgumentException' occurred in system.windows.forms.dll

Additional information: '0' is not a valid value for Interval. Interval must be greater than 0.

Figure 5-8 Exception generated by the upgraded project.

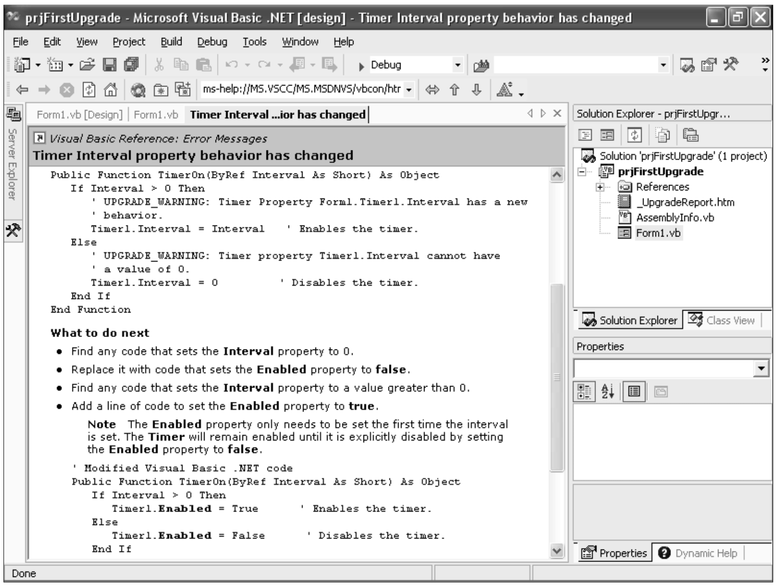
Click Continue to stop the program and return to the IDE. The exception occurs at this line in the *Command1_Click* event:

```
Me.Timer1.Interval = 0
```

What happened? Let's look at the upgraded code for the *Command1_Click* event:

```
Private Sub Command1_Click(ByVal eventSender As System.Object, _
ByVal eventArgs As System.EventArgs) Handles Command1.Click
   'UPGRADE_WARNING: Timer property Timer1.Interval cannot have a value
   'of 0. Click for more:
   'ms-help://MS.VSCC/commoner/redir/redirect.htm?key-
   'word="vbup2020.htm"
   Me.Timer1.Interval = 0
End Sub
```

The Upgrade Wizard has inserted a warning: "Timer Property Timer1.Interval cannot have a value of 0." Warnings alert you to run-time differences between Visual Basic 6 and Visual Basic .NET. Click the underlined part of the comment, and a Help topic opens in the IDE with the title "Timer Interval Property Behavior Has Changed." Figure 5-9 shows what the Help topic looks like.


prjFirstUpgrade - Microsoft Visual Basic .NET [design] - Timer Interval property behavior has changed
File Edit View Project Build Debug Tools Window Help
Debug
ms-help://MS.VSCC/MS.MSDNVS/vbcon/htr
Form1.vb [Design] | Form1.vb | Timer Interval ...ior has changed
Server Explorer
Solution Explorer - prjFirstUpgr...
Solution 'prjFirstUpgrade' (1 project)
prjFirstUpgrade
References
_UpgradeReport.htm
AssemblyInfo.vb
Form1.vb
Solution Explorer
Class View
Properties
Properties
Dynamic Help
Done

Visual Basic Reference: Error Messages

Timer Interval property behavior has changed

```
Public Function TimerOn(ByRef Interval As Short) As Object
   If Interval > 0 Then
      ' UPGRADE_WARNING: Timer Property Form1.Timer1.Interval has a new
      ' behavior.
      Timer1.Interval = Interval    ' Enables the timer.
   Else
      ' UPGRADE_WARNING: Timer property Timer1.Interval cannot have
      ' a value of 0.
      Timer1.Interval = 0          ' Disables the timer.
   End If
End Function
```

What to do next

- Find any code that sets the **Interval** property to 0.
- Replace it with code that sets the **Enabled** property to **false**.
- Find any code that sets the **Interval** property to a value greater than 0.
- Add a line of code to set the **Enabled** property to **true**.

Note The **Enabled** property only needs to be set the first time the interval is set. The **Timer** will remain enabled until it is explicitly disabled by setting the **Enabled** property to **false**.

```
' Modified Visual Basic .NET code
Public Function TimerOn(ByRef Interval As Short) As Object
   If Interval > 0 Then
      Timer1.Enabled = True       ' Enables the timer.
   Else
      Timer1.Enabled = False       ' Disables the timer.
   End If
```

Figure 5-9 Help topic for the Timer Interval property.

The Help topic says that *Timer.Interval* no longer disables the timer. Instead you have to use the statement *Timer1.Enabled = False*. Let's change our code. Replace the line

```
Me.Timer1.Interval = 0
```

with

```
Me.Timer1.Enabled = False
```

Now press F5 to run the project. Notice that this time Command1 stops the timer just as it did in Visual Basic 6.

We've finished upgrading our project, but let's not stop there. We'll do something with our upgraded project that you couldn't do in Visual Basic 6—make the form fade out. Add the following code to the *Command1_Click* event:

```
Dim i As Integer
For i = 99 To 0 Step -1
   Me.Opacity = i / 100
Next
End
```

After you add this code, the event should look like this:

```
Private Sub Command1_Click(ByVal eventSender As System.Object, _
ByVal eventArgs As System.EventArgs) Handles Command1.Click
   'UPGRADE_WARNING: Timer property Timer1.Interval cannot have a value
   'of 0. Click for more: ms-help://MS.MSDNVS/vbcon/html/vbup2020.htm
   Me.Timer1.Enabled = False
   Dim i As Integer
   For i = 99 To 0 Step -1
      Me.Opacity = i / 100
   Next
   End
End Sub
```

Press F5 to run the project. Now when you click Command1, the form becomes transparent and fades from view. As you can see, you may have to make some modifications to get your project working, but after you've done so, you can take advantage of the many features in Visual Basic .NET that weren't available in Visual Basic 6. Upgrading enables you to add more value to your projects.

Language Changes

Now that we've got our project working, let's take a few moments to review where the Upgrade Wizard changed the code, stepping through the original code line by line. Here is the original Visual Basic 6 code:

```
Option Explicit

Private Sub Command1_Click()
   Me.Timer1.Interval = 0
End Sub
```

```
Private Sub Timer1_Timer()
   Dim i As Integer
   i = Rnd * 10
   Me.Text1 = i
End Sub
```

The first line, *Option Explicit,* is upgraded to

```
Option Strict Off
Option Explicit On
```

Notice that *On* was added to *Option Explicit*. This statement has the same effect as the plain *Option Explicit* in Visual Basic 6—it means that all variables must be explicitly declared using the *Dim*, *Public*, or *Private* statement. As you might guess, you can also write *Option Explicit Off*, which lets you use variables without explicitly declaring them. *Option Strict* is a new option. When it is set to *On*, you will get compile errors if your code does either of the following:

- Uses late binding
- Performs a narrowing conversion—that is, converts a variable of one type to another, where an overflow or type mismatch might occur (like assigning a *Date* to a *String* or an *Integer* to a *Long*)

When your project is upgraded, *Option Strict Off* is put at the top of each file to turn off strict type checking, since setting it to *On* would cause compile errors in most Visual Basic 6 code.

The Different *Option* Statements

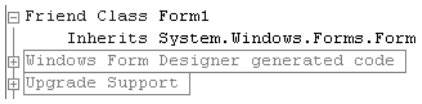
Visual Basic .NET has three types of *Option* statements. Along with *Option Strict* and *Option Explicit*, there is *Option Compare*. As it did in Visual Basic 6, *Option Compare* sets the string comparison mode. By default, *Strict* is set to *Off*, *Explicit* is set to *On*, and *Compare* is set to *Binary*. You can change the defaults for each option in the Build section of the project's properties. If you change the default for, say, *Option Compare* to *Text*, all forms, modules, and classes will use *Option Compare Text*, unless they have their own *Option Compare* statements that override the default setting.

The wizard also inserts a *Class* statement. In Visual Basic .NET, all source files have a .vb file extension. Whether a file is a module, a class, or a form depends on the code inside the file. Why does the wizard insert a *Class* statement instead of a *Form* statement? Because all forms are actually classes that inherit from the base class *System.Windows.Forms.Form*. Here is what the *Class* statement looks like. (Event Code) indicates where your events are placed.

```
Friend Class Form1
   Inherits System.Windows.Forms.Form
   (Event Code)
End Class
```

If you want to, you can put several forms or classes into a single file, provided each has its own *Class* statement.

Just beneath the *Class* statement, you'll see two collapsed regions: Windows Form Designer Generated Code and Upgrade Support, as shown in Figure 5-10.



```
Friend Class Form1
    Inherits System.Windows.Forms.Form
Windows Form Designer generated code
Upgrade Support
```

Figure 5-10 Collapsed code.

This is "system-generated code," or code used by Windows Forms to store the design layout of the form and by the Upgrade Wizard to store instancing information. We'll discuss these sections in Chapter 6 and in Chapter 12For now, let's move on and look at your upgraded event code. The *Command1_Click* event code is upgraded from

```
Private Sub Command1_Click()
   Me.Timer1.Interval = 0
End Sub
```

to

```
Private Sub Command1_Click(ByVal eventSender As System.Object, _
ByVal eventArgs As System.EventArgs) Handles Command1.Click
   'UPGRADE_WARNING: Timer property Timer1.Interval cannot have a value
   'of 0. Click for more: ms-help://MS.VSCC/commoner/redir/
   'redirect.htm?keyword="vbup2020.htm"
   Me.Timer1.Interval = 0
End Sub
```

The signature of the *Click* procedure has changed from Visual Basic 6. In Visual Basic .NET, most events are passed two parameters: *eventSender*, an object that is set to the control (in this case CommandButton1); and *eventArgs*, an object that contains extra arguments (such as mouse coordinates). The contents of *eventArgs* change depending on the type of event. For *Click* events, it contains nothing of interest. At the end of the *Sub* statement, you'll see the *Handles* keyword. This keyword associates the event procedure with the control event. Each event procedure can be associated with multiple controls and multiple events. For example, the following *Handles* clause would connect a procedure to the *Command1.Click*, *Command2.Click*, and *Command1.TextChanged* events:

```
Handles Command1.Click, Command2.Click, Command1.TextChanged
```

The line after the *Sub* statement contains the following upgrade warning:

```
'UPGRADE_WARNING: Timer property Timer1.Interval cannot have a value
'of 0. Click for more: ms-help://MS.VSCC/commoner/redir/
'redirect.htm?keyword="vbup2020.htm"
```

The Upgrade Wizard inserts warnings whenever it encounters a statement that may not work the same in Visual Basic .NET. It inserts the warning on one line before the statement with the problem. You can filter the Task List to show the upgrade warnings. For information on using the Task List, see Chapter 6.

The final statement in the *Command1_Click* event is

```
Me.Timer1.Interval = 0
```

This statement is exactly the same as it was in Visual Basic 6.

Timer1_Tick has also undergone some changes during the upgrade. The Visual Basic 6 code

```
Private Sub Timer1_Timer()
   Dim i As Integer
   i = Rnd * 10
   Me.Text1 = i
End Sub
```

is upgraded to

```
Private Sub Timer1_Tick(ByVal eventSender As System.Object, _
ByVal eventArgs As System.EventArgs) Handles Timer1.Tick
   Dim i As Short
   i = Rnd() * 10
   Me.Text1.Text = CStr(i)
End Sub
```

Just as we saw with *Command1_Click*, the *Timer1_Tick* procedure now has a new signature. The line after the *Sub* statement has also changed. In Visual Basic .NET, the *Integer* data type has grown in size from 16 bits to 32 bits, and a new data type, *Short*, has been added to the language for 16-bit numbers. For this reason, the line

```
Dim i As Integer
```

is upgraded to

```
Dim i As Short
```

This ensures that the variable *i* is the same size as it was in Visual Basic 6. Let's move on to the next line:

```
i = Rnd * 10
```

is upgraded to

```
i = Rnd() * 10
```

In Visual Basic .NET, all calls to functions must have parentheses, so the Upgrade Wizard has added them to the *Rnd* function. Now let's look at the final line of code in this event procedure.

```
Me.Text1 = i
```

is upgraded to

```
Me.Text1.Text = CStr(i)
```

This line has undergone two changes. Because Visual Basic .NET does not support default properties, the Upgrade Wizard resolves the default property of *Me.Text1* and expands the statement to *Me.Text1.Text*. Because the *Text* property is being assigned to a number, the wizard also explicitly converts *i* to a string, using the *CStr* function.

Whew! The upgrade resulted in quite a number of changes, but looking over the upgraded code you'll see that it still looks familiar. The overarching impression is that upgraded code is more explicit than the Visual Basic 6 version. That, in a nutshell, is the real difference between Visual Basic 6 and Visual Basic .NET. Visual Basic 6 concealed a lot of details, like forms being classes, the design layout, and default properties. In Visual Basic .NET, these are all exposed for you to see.

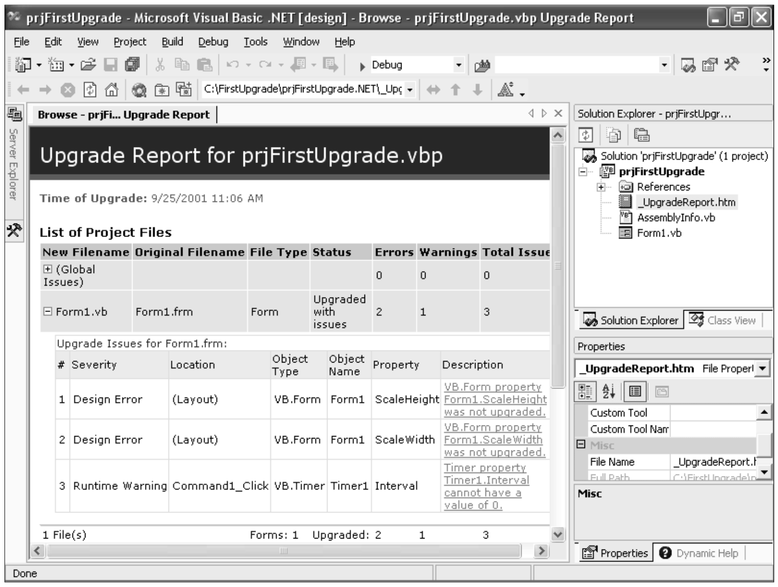
Other Files in Your Project

Now close the Code Editor and Windows Form Designer windows. Let's turn our attention to the Solution Explorer for a second. The Solution Explorer shows all the files in your upgraded project. Notice that the Upgrade Wizard has added two extra files to your project: _UpgradeReport.htm and Assembly-

Info.vb. Go ahead and double-click AssemblyInfo.vb to open it in the Code Editor. This file contains attributes. Attributes don't perform any action; they are used to store additional information about a method, form, class, or project. The attributes in this file allow you to change information about your project such as the assembly title and assembly description. Generally, unless you have a specific need, you can leave these values at their default settings. Close the AssemblyInfo.vb Code Editor and we'll look at the upgrade report.

> **More Info** A discussion of the assembly attributes in AssemblyInfo.vb is beyond the scope of this book. These attributes can be used to specify the identity of your Visual Basic .NET application. To learn more about attributes, search the Visual Basic .NET Help for "Attributes" and "Setting Assembly Attributes."

The upgrade report is added to your project with the filename _UpgradeReport.htm. It is an HTML file. Double-click the file in the Solution Explorer to open it. Click the + sign next to Form1 to display all of the issues with Form1, and you'll see that the difference with *Timer.Interval* has been recorded. The report should look similar to Figure 5-11.



Upgrade Report for prjFirstUpgrade.vbp

Time of Upgrade: 9/25/2001 11:06 AM

List of Project Files

New Filename	Original Filename	File Type	Status	Errors	Warnings	Total Issue
(Global Issues)				0	0	0
Form1.vb	Form1.frm	Form	Upgraded with issues	2	1	3

Upgrade Issues for Form1.frm:

#	Severity	Location	Object Type	Object Name	Property	Description
1	Design Error	(Layout)	VB.Form	Form1	ScaleHeight	VB.Form property Form1.ScaleHeight was not upgraded.
2	Design Error	(Layout)	VB.Form	Form1	ScaleWidth	VB.Form property Form1.ScaleWidth was not upgraded.
3	Runtime Warning	Command1_Click	VB.Timer	Timer1	Interval	Timer property Timer1.Interval cannot have a value of 0.

1 File(s) Forms: 1 Upgraded: 2 1 3

prjFirstUpgrade - Microsoft Visual Basic .NET [design] - Browse - prjFirstUpgrade.vbp Upgrade Report
File Edit View Project Build Debug Tools Window Help
Debug
C:\FirstUpgrade\prjFirstUpgrade.NET_Upg
Browse - prjFi... Upgrade Report
Server Explorer
Solution Explorer - prjFirstUpgr...
Solution 'prjFirstUpgrade' (1 project)
prjFirstUpgrade
References
_UpgradeReport.htm
AssemblyInfo.vb
Form1.vb
Solution Explorer
Class View
Properties
_UpgradeReport.htm File Propert
Custom Tool
Custom Tool Nam
Misc
File Name _UpgradeReport.h
Full Path C:\FirstUpgrade\p
Misc
Properties
Dynamic Help
Done

Figure 5-11 An upgrade report.

The upgrade report captures in one place the issues found while upgrading the project to Visual Basic .NET. Like the comments in code, each issue is also a hyperlink that navigates to a Help topic describing how to fix the problem. For more on upgrade issues, see Chapter 7

You now have your Visual Basic .NET project. The Upgrade Wizard keeps the form layout the same, makes necessary syntax changes, and warns you about any issues found during the upgrade. After you get your project working, you can enhance it using some of the exciting new features of Visual Basic .NET. We'll put off a discussion of these features until later, because we still need to look at how to upgrade project groups and snippets of code.

Upgrading Project Groups

The Upgrade Wizard upgrades only projects. But what about project groups? In Visual Basic 6, you could create a project group that contained several projects. When you pressed F5, all of the projects in the group were compiled together, and the debugger could step from code in one project into code in another. Project groups are useful when one project references another.

The Visual Basic .NET equivalent of a project group is a solution. Each solution contains one or more projects. As in Visual Basic 6, they are all compiled together when you press F5, projects can reference other projects in the same group, and the debugger can step from code in one project into code in another. Not all projects in a solution have to be Visual Basic .NET projects. A solution can contain projects written in different languages, such as Visual Basic .NET, C#, and Visual C++.

To upgrade a project group to Visual Basic .NET, you need to upgrade one project at a time, starting with the most "downstream" project and working your way up the dependency hierarchy. For example, if your project group contains an EXE project that references a DLL project, you should upgrade the EXE first, followed by the DLL. Let's walk through an example to see how this is done. First we will create a project group in Visual Basic 6, and then we will upgrade it one project at a time to Visual Basic .NET. Our project group will be simple: it will show "Hello World" in a message box when you click a CommandButton on a form.

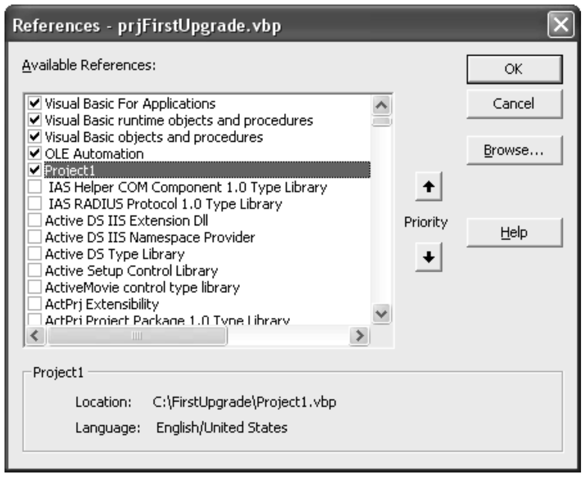
1. Open the project you created earlier in Visual Basic 6 (prjFirstUpgrade.vbp). Add a new DLL project by choosing Add Project from the File menu and selecting ActiveX DLL from the Add Project dialog box. A DLL project called Project1 is added to the Project Explorer. The two projects are now part of a project group.

2. Visual Basic 6 automatically adds a new class in Project1, *Class1*. Add the following code to the class:

```
Sub sayHelloWorld()
   MsgBox "Hello World"
End Sub
```

3. Close the *Class1* Code Editor, and save all the files by choosing Save Project Group from the File menu. Make sure you save all the files in the same directory as the first project. Accept the default filenames; the project group will be called Group1.vbp, and the new project will be called Project1.vbp.

4. Now we will make prjFirstUpgrade reference Project1. Select prj-FirstUpgrade in the Project Explorer, and choose References from the Project menu. In the References dialog box, select Project1, and click OK. Figure 5-12 shows the References dialog box.


References - prjFirstUpgrade.vbp

Available References:

- [✓] Visual Basic For Applications
- [✓] Visual Basic runtime objects and procedures
- [✓] Visual Basic objects and procedures
- [✓] OLE Automation
- [✓] Project1
- [ ] IAS Helper COM Component 1.0 Type Library
- [ ] IAS RADIUS Protocol 1.0 Type Library
- [ ] Active DS IIS Extension Dll
- [ ] Active DS IIS Namespace Provider
- [ ] Active DS Type Library
- [ ] Active Setup Control Library
- [ ] ActiveMovie control type library
- [ ] ActPrj Extensibility
- [ ] ActPrj Project Package 1.0 Type Library

Project1

Location: C:\FirstUpgrade\Project1.vbp

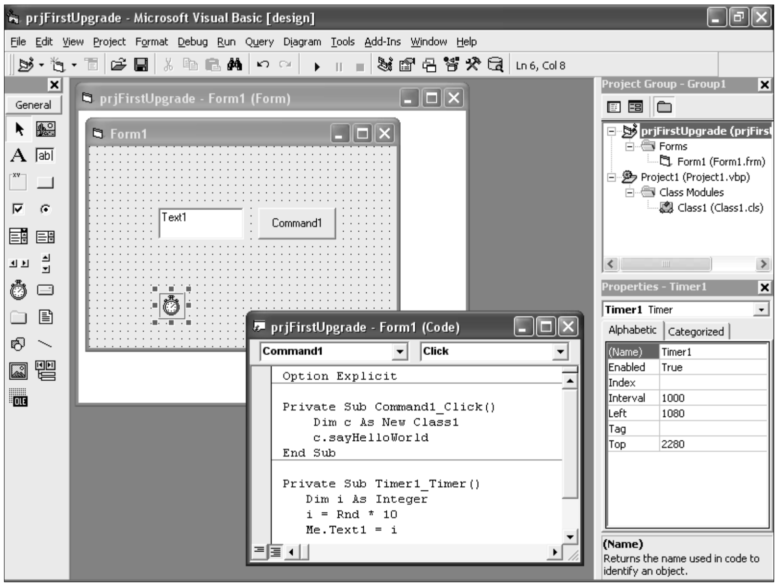
Language: English/United States

Figure 5-12 References dialog box.

5. Let’s use the Project1 *Class1* from prjFirstUpgrade. Open prjFirst-Upgrade.Form1 in the Form window, and double-click the Command1 CommandButton to edit the *Click* event code. Change the *Click* event code to the following:

```
Private Sub Command1_Click()
   Dim c As New Class1
   c.sayHelloWorld
End Sub
```

After you've added this code, the project group will look like Figure 5-13.


prjFirstUpgrade - Microsoft Visual Basic [design]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
Ln 6, Col 8
General
prjFirstUpgrade - Form1 (Form)
Form1
Text1
Command1
prjFirstUpgrade - Form1 (Code)
Command1
Click
Project Group - Group1
prjFirstUpgrade (prjFirst
Forms
Form1 (Form1.frm)
Project1 (Project1.vbp)
Class Modules
Class1 (Class1.cls)
Properties - Timer1
Timer1 Timer
Alphabetic
Categorized

```
Option Explicit

Private Sub Command1_Click()
    Dim c As New Class1
    c.sayHelloWorld
End Sub

Private Sub Timer1_Timer()
   Dim i As Integer
   i = Rnd * 10
   Me.Text1 = i
```

(Name)	Timer1
Enabled	True
Index	
Interval	1000
Left	1080
Tag	
Top	2280

(Name)
Returns the name used in code to identify an object.

Figure 5-13 Project group in Visual Basic 6.

6. Let's test it out. Press F5 to run the project group. When you click the CommandButton Command1, a message box pops up with the text "Hello World."

7. We'll do two more things before we upgrade the group. First, save the group by choosing Save Project Group from the File menu. Next, compile the group by choosing Make Project Group from the File menu. Compiling is necessary because the Upgrade Wizard checks the timestamp of the Project1 project against the timestamp of the Project1 DLL. If the project file is newer than the DLL, the wizard assumes that the DLL is out of date and asks you to recompile it.

Now that we've created a project group in Visual Basic 6, let's upgrade it to Visual Basic .NET. First we'll upgrade the EXE prjFirstUpgrade. Then we'll upgrade the DLL Project1. Finally, we'll change prjFirstUpgrade so that it references the upgraded Project1.

1. Start Visual Basic .NET, and upgrade prjFirstUpgrade. (If you need a recap on how to upgrade a project, see the section "Upgrade Walkthrough" earlier in this chapter.) You won't need to make any modifications to the upgraded code, since we replaced *Timer.Interval* = *0* with code that shows a message box.

2. Now let's upgrade Project1 to be part of this solution. In Visual Basic .NET, from the File menu choose Add Project and then Existing Project. This opens the Add Existing Project dialog box. Select Project1.vbp and click the Open button to upgrade Project1. Accept the default upgrade options, as you did with prjFirstUpgrade.
3. Is that all there is to do? Not quite. The project prjFirstUpgrade still references the Visual Basic 6 version of Project1. We need to delete the reference to the Visual Basic 6 Project1 DLL and add a reference to the upgraded Visual Basic .NET Project1. First let's remove the reference. Expand the prjFirstUpgrade references node and remove the reference *Interop.Project1_1_1* by right-clicking it and selecting Remove. Note that in your project, this reference might be called something slightly different. Look for the reference with "Project1" in the name. For information on adding and removing references, see Chapter 6
4. Add a reference to the upgraded Project1 by right-clicking References, selecting Add Reference, and then selecting Project1 in the Add Reference dialog box.
5. That's it! Press F5 to run the solution. When you click the Command1 button, you will see a message box, as shown in Figure 5-14.

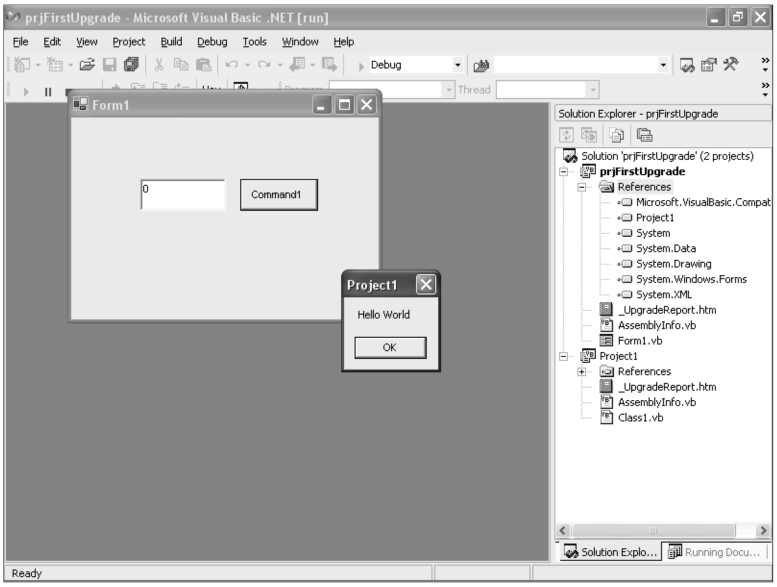
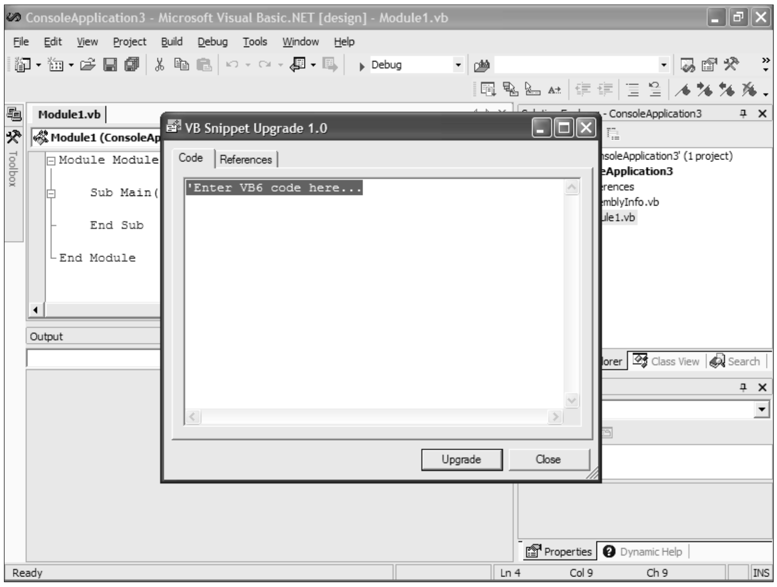


Figure 5-14 Upgraded project group.

You now know the basic procedure for upgrading project groups. If your Visual Basic 6 project group contains many projects, you simply repeat these steps for every project in the group. Start with the most downstream project, and work your way up the dependency hierarchy.

Using the VB Snippet Upgrade Add-In

What is a snippet? A **snippet** is a fragment of code, such as a procedure or even just two or three lines of code. A Visual Basic 6 snippet is a fragment of Visual Basic 6 code. It may be sample code that you received by e-mail, or it might be a function you use all the time. The VB Snippet Upgrade add-in is designed for upgrading snippets. It is useful when you want to upgrade only a few lines of code, not an entire project. To use it, you must have a Code Editor window open in Visual Basic .NET. Choose VB Snippet Upgrade from the Tools menu. The VB Snippet Upgrade window opens, as shown in Figure 5-15.



```
Module Module
    Sub Main(
    End Sub
End Module
```

```
'Enter VB6 code here...
```

Figure 5-15 The VB Snippet Upgrade add-in.

Simply type or paste Visual Basic 6 snippet code into this window, and click the Upgrade button. Let's try upgrading a simple example. Type the following into the window:

```
Dim x As Integer
Debug.Print x
```

Now click the Upgrade button. The snippet is upgraded and inserted into the Code Editor at the current insertion point:

```
Dim x As Short
System.Diagnostics.Debug.WriteLine(x)
```

The VB Snippet Upgrade add-in is a great way to familiarize yourself with how code upgrades from Visual Basic 6 to Visual Basic .NET. The add-in can upgrade several lines of code, a function, or even a whole module. The only limitation is that the snippet can't refer to functions or objects that are not included as part of the snippet—the VB Snippet Upgrade add-in needs to examine all of the objects and functions you use in that snippet. If they are defined elsewhere, the add-in won't know how to interpret them. You can add references to the snippet, however. A list box on the References tab allows you to add or remove COM libraries that your snippet references. Any libraries you add here are also added to your Visual Basic .NET project.

Getting Updates

As this book went to press, the VB Snippet Upgrade add-in was still in development, and Microsoft had not yet decided whether it would be ready to ship with Visual Basic .NET or be made available later as a download. If it is not installed with Visual Basic .NET, you will be able to get it from the Visual Basic Web site, *http://msdn.microsoft.com/vbasic/.* From time to time this Web site is updated with new versions of the Upgrade Wizard, the VB Snippet Upgrade add-in, and other resources for upgrading. These updates may contain bug fixes, whitepapers, or new upgrade capabilities.

Upgrading Using the Command Line

Those of you who prefer using command-line tools over graphical wizards will be pleased to know that Visual Basic .NET ships with a command-line upgrade tool: VBUpgrade.exe. If you accept the default file locations during installation, you will find it installed to the following directory:

<*drive*>:\Program Files\Microsoft Visual Studio .NET\Vb7\VBUpgrade\

The command-line upgrade tool produces exactly the same result as the Upgrade Wizard—in fact, they share the same underlying upgrade engine. Let's see it in action. Assuming that the tool resides on drive C, open a Command Prompt window and change the current directory to the location of the tool, using the CD command, as follows:

```
CD "C:\Program Files\Microsoft Visual Studio .NET\Vb7\VBUpgrade"
```

Now type the following to see the command-line options:

```
vbupgrade /?
```

The upgrade tool shows you its available options:

```
Microsoft (R) Visual Basic.NET Upgrade Tool Version 7.00.9238.0
Copyright (C) Microsoft Corp 2000-2001.
Portions copyright ArtinSoft S.A.
All rights reserved.
Usage: VBUpgrade <filename> [/Out <directory>] [/NoLog | /LogFile
<filename>] [/Verbose] [/GenerateInterfaces]
/?                      Display this message
/Out                    Target directory (default is ".\OutDir")
/Verbose                Outputs status and results
/NoLog                  Don't write a log file
/LogFile                Log file name (default is
                        "<ProjectFileName>.log")
/GenerateInterfaces    Generates interfaces for public classes
```

To upgrade a project, you need to specify the project filename and the destination directory for the upgraded project. If the destination directory doesn't exist already, the upgrade tool will create it for you. For example, the following statement upgrades C:\Project1.vbp to the C:\Project1.NET directory:

```
vbupgrade c:\Project1.vbp /Out c:\Project1.NET
```

Because it is a command-line tool, the upgrade tool doesn't show any wizard pages while upgrading. The one difference between the command-line tool and the Upgrade Wizard is that the command-line tool does not delete any files in the destination directory. If the destination directory already contains files, the command-line tool stops with an error.

Conclusion

This chapter has covered the basic mechanics of upgrading projects, project groups, and code snippets. Doing each is easy, once you know how. The next chapter takes a more detailed look at how to work with your upgraded project in Visual Basic .NET.