

45

Upgrading to Visual Studio 2005

Moving to a new development environment can sometimes be daunting. Visual Basic 6 users came head to head with a significant learning curve when Visual Basic .NET was first released in 2002. In fact, the transition to the new language was so overwhelming that some high-profile gurus in the language refused to move over and started campaigns to bring back the original language constructs.

Many corporations faced with the effort required in converting to .NET also decided to hold off, and as a result the current Visual Basic programming world is divided into two sizable camps, Visual Basic 6 programmers and Visual Basic .NET developers, with the rare few using both tools. The crux of the problem in transitioning to the new language was that the core Visual Basic language evolved to more closely suit object-oriented principles and to synchronize with the .NET Framework.

Therefore, instead of a language that had its origins in an interpretive compiler, Visual Basic .NET utilized the paradigm of everything being an object and therefore being treated as such; even forms and their controls were transformed into “just another” object that inherited from a class in the .NET Framework. As a result of these functional changes to the language, it was difficult to automatically upgrade Visual Basic 6 code to the new .NET environment, even with the Upgrade Wizard.

Besides the significant language changes, there were new concepts to consider, including proper inheritance and features such as method overloading. Coupled with these language-side issues, the environment itself also changed. Mostly the changes were improvements over the old way of doing things, but in the process of creating a new development environment, Microsoft left out some major features on which Visual Basic 6 developers depended. The most prominent feature that got left behind was the edit-and-continue process that enabled programmers to edit their code while the program was running in debug mode and then allow the program to continue running, taking advantage of the new altered code without having to restart the program.

When Microsoft began work on Visual Studio 2005, they told the programming world that their intention was to make this version of Visual Studio full of features that developers wanted, as opposed to those features that Microsoft itself thought they needed. Two such features that were on the lips of every Visual Basic programmer were the return of the edit-and-continue feature and a more convenient way of upgrading Visual Basic 6 code. Edit-and-continue is dealt with in Section IX, where the debugging features of Visual Studio 2005 are discussed, but this chapter is all about upgrading Visual Basic 6 code to Visual Basic 2005.

The Upgrade Process

You should approach the upgrading process carefully. In fact, the first serious decision you have to make is whether to upgrade at all. Visual Studio 2005 applications can interact with Visual Basic 6 applications through the COM Interop layer, so if you have an existing application that just needs to talk to new programs, then your best option may actually be to leave the VB6 project as is. However, if you need to do a major rewrite of an existing project, then it might be a great time to convert it up to Visual Basic 2005 to take advantage of the many benefits of producing applications in .NET 2.0.

Either way, besides the most basic program, it is most likely that the upgrade process will include a list of tasks as well as compilation errors and warnings, all of which need to be addressed before the converted program will run as expected. In addition, you'll need to check every conversion point in the application code to ensure that it still acts as you expect.

If you're going through a total rewrite of an existing application, you may even want to start a new project in Visual Studio 2005 and use the Upgrade Visual Basic 6 Code tool to migrate the old program code a section at a time to make it more manageable.

Getting Ready to Upgrade

To give the upgrade process the best chance of success, there are some guidelines to follow in preparing the Visual Basic 6 code for the conversion. These optional tasks minimize the impact of some common issues that the Upgrade Wizard faces, such as default properties.

Things That Just Won't Work

Some features of Visual Basic 6 are so incompatible with Visual Basic 2005's new language model that the Upgrade Wizard can't handle them at all. For instance, code that uses the graphic methods on forms (or even the `Line` and various `Shape` controls) cannot be converted at all and needs to be completely rewritten.

Similarly, applications that take advantage of the drag-and-drop commands in Visual Basic 6 need to have those sections recreated using the new techniques within .NET 2.0.

A large number of Windows API definitions cannot be converted over to Visual Basic 2005 because they rely on `Variant` and `Any`. Variants will be converted to `Objects`, which means in some cases the behavior of the application will change, while the Upgrade Wizard can't replicate the behavior of `Any`. However, the .NET Framework contains many functions that eliminate the need for calling the Windows API directly. Therefore, while you'll need to rewrite those sections of code that call Windows APIs, you can take advantage of the managed code functionality found in the framework instead.

One last issue worth mentioning is that if the Visual Basic 6 application uses DAO for its data access (such as to a Microsoft Access database), then it will require a reworking of the data access layer because Visual Studio 2005 doesn't support data-binding to DAO.

Changing the Original Code

The many issues that can occur during the upgrade can be loosely grouped into two sets. The first set of problems is items that work fine in Visual Basic 6 but don't work at all in Visual Basic 2005; however, changing them in the original code doesn't necessarily make sense. For instance, the graphics methods for a form such as `Cls` and `PSet` are not available in Visual Basic 2005. However, you can't easily change the code in Visual Basic 6 before you perform the upgrade to the new graphics methods, because the .NET Framework-based methods aren't available from the Visual Basic 6 code. Control examples include the `Line` and `Shape` controls mentioned previously; these don't have a corresponding component in .NET and are converted to empty `Label` controls at best.

In both of these situations, you're forced to accept your fate and perform development on the converted code to regain your original application's behavior.

However, other features of Visual Basic 6 that do not convert well in the Upgrade Wizard have alternatives in Visual Basic 6 itself. If you're more comfortable with working in Visual Basic 6, a little bit of work in the original code base can help immensely with the upgrade process.

For example, the `Timer` control in Visual Basic 6 has an `Enabled` property, but many programmers chose to simply set the `Interval` property to 0, which effectively paused the `Timer`. However, whenever you set the `Interval` value to 0 in Visual Basic 2005, it implicitly converts the setting to 1, which means the `Timer` continues to run and fire. To avoid this issue cropping up when you upgrade your application code, make sure you use the `Enabled` property instead of setting `Interval` to 0.

This problem is a great example of code that will be converted without error but have unexpected results. If you leave the code with the improperly coded `Interval = 0` statement, your `Timer` may fire a little more frequently than you ever intended it to!

Upgrading other Visual Basic 6 code features may produce warning messages that can be easily avoided by changing the code before the upgrade process. The most common issue of this type is the capability to use default properties in Visual Basic 6. This is the practice of simply referring to the object or control and letting Visual Basic determine what property should be used. For example, a `TextBox` has a default property of `Text`, so instead of assigning a string to the `Text` property explicitly like this

```
TextBox1.Text = myString
```

you can simply specify the `TextBox1` control and Visual Basic implicitly assumes the `Text` property; the following line is the equivalent of the previous line of code in Visual Basic 6:

```
TextBox1 = myString
```

Visual Basic 2005 does not support default properties at all, requiring you to explicitly specify the property to which you're referring. The Upgrade Wizard does a great job of trying to determine what property it should use when converting the code, but sometimes it just doesn't know what to do. Figure 45-1 shows an example of this kind of problem—the Upgrade Wizard couldn't determine the default property of the `ListView` control's `Item` object.

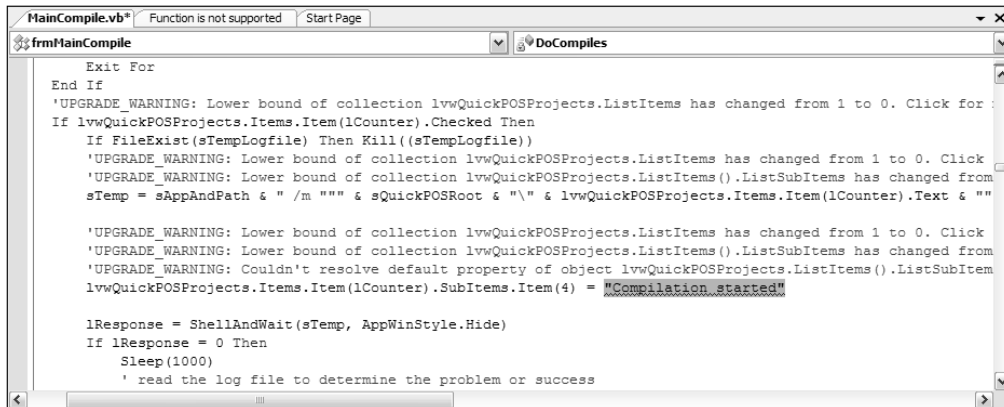


Figure 45-1

Explicitly coding the property names in the original Visual Basic 6 code in the first place avoids this problem entirely. The MSDN Library documentation that comes with Visual Studio has many other recommendations (look for the topics entitled *Preparing a Visual Basic 6.0 Application for Upgrading* and *Language Recommendations for Upgrading*).

Using the Upgrade Project Wizard

When you are ready to convert a Visual Basic 6 application project to Visual Basic 2005, you can do so by simply opening the project file (with a .vbj extension) in Visual Studio 2005. The Visual Studio IDE determines that the project is from an older version of Visual Basic and automatically starts the Visual Basic Upgrade Wizard.

After reviewing the goals of the Upgrade Wizard, click the Next button to progress to the first step. The first choice you have is whether the application should be converted to a Windows application (.exe) or a binary library such as a DLL or custom control. Normally this choice is taken out of your hands, as the wizard attempts to determine the project type by reading the original project information.

The next step is to select the destination for the Visual Basic 2005 version of the project (see Figure 45-2). Note that the Upgrade Wizard will not convert the project in place. In fact, it won't allow you to convert the application into anything but an empty folder. If you specify a location that doesn't exist, the wizard will ask you if you want it to create the folder automatically for you.

The default location for the wizard is a subfolder underneath the original project file's location, but you can set it to anywhere within your local file system.

Your work is now done. Click Next to get to the confirmation page, and then click Next one more time to start the actual Upgrade Wizard. As the wizard progresses through the conversion, the dialog updates the status message with the current module and member being upgraded (see Figure 45-3). The status bar of the main Visual Studio IDE also dynamically updates, showing the percentage of completion for the upgrade process.

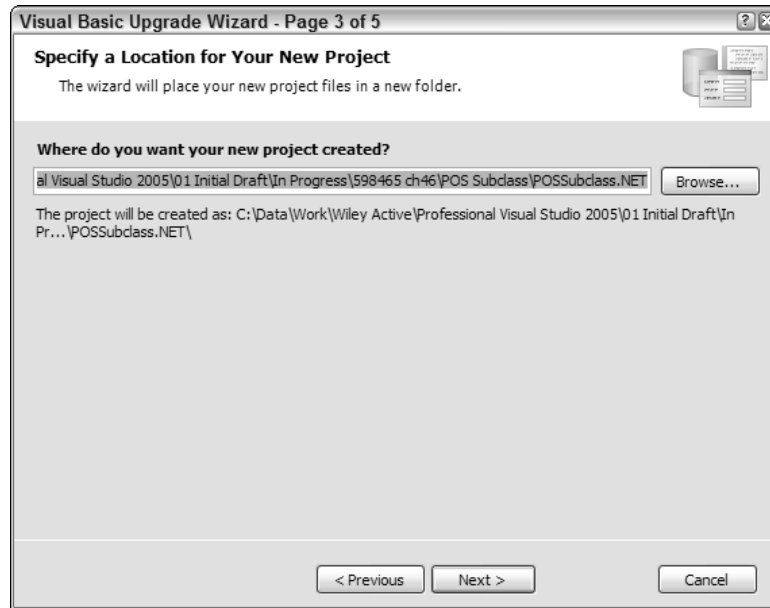


Figure 45-2

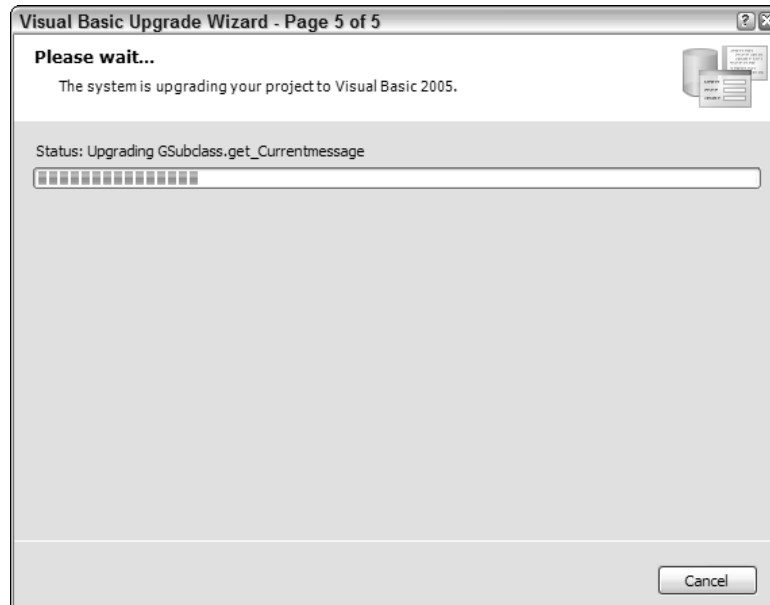


Figure 45-3

Chapter 45

Most of the time, the Upgrade Wizard will complete the process and present you with a list of errors, warnings, and additional tasks for you to check (this is discussed in the next section “Checking the Upgrade Output”). However, some programs cannot be converted at all, and will present you with an error message. Usually, when this occurs it is because the original code itself wouldn’t compile.

Figure 45-4 shows an example of the kind of message that can be produced. This example application didn’t convert because the referenced DLLs in the project were not registered on the machine doing the conversion. As noted at the bottom of the message text, if this kind of error occurs, your first step should be to make sure the program compiles and runs in Visual Basic 6 before trying the Upgrade Wizard again.



Figure 45-4

Unfortunately, sometimes you’ll get a blank error message. This kind of error is extremely rare, and normally points to a project that is somehow incomplete, such as missing files. When you encounter this kind of error, you can still determine where the error occurred by viewing the log file that is produced as the Upgrade Wizard does its work.

This log file is located in the destination project folder you specified in the wizard, and is made up of XML nodes detailing each file as it is upgraded. Scrolling through the log you can identify the point at which the upgrade failed, and then look at that specific file’s contents (see Figure 45-5). In this sample case, it turned out that the attempted conversion of the source file failed because the file didn’t exist where the project file expected it to.

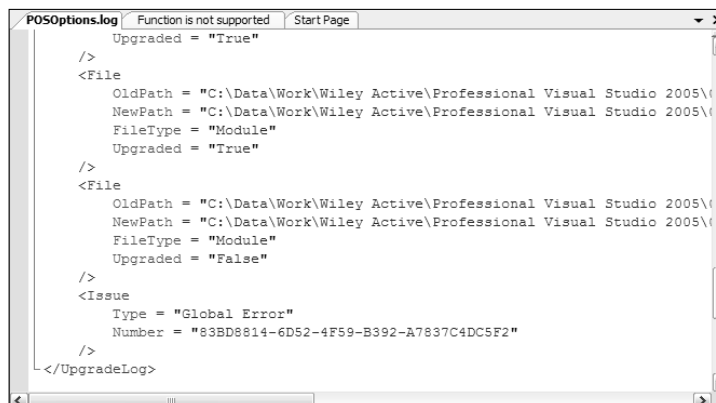


Figure 45-5

Checking the Upgrade Output

When the Upgrade Wizard is complete, it opens the new Visual Basic 2005 project in the IDE. If there are any issues to address, they are added to the Task List and the Error List with all the associated IntelliSense formatting that comes with warnings and errors applied to the code listings themselves. In addition, an Upgrade Report is generated and added to the project's file list in the Solution Explorer. Examining each of these upgrade output elements is the best way to make sure your application was converted successfully.

Task List

The first port of call when reviewing the upgrade output is the Task List. Visual Studio 2005 should display this automatically, but if it's not shown, use the View⇨Other Windows⇨Task List menu command to display it. The result of a typical conversion will appear similar to what is shown in Figure 45-6.

Description	File	Line
TODO: Review the values of the assembly attributes	AssemblyInfo.vb	10
UPGRADE_WARNING: Class instantiation was changed to public. Click for more: 'ms-help://MS.VSCC.v80/dv_commoner/local/redirect.htm?keyword=ED41034B-3890-49FC-8076-8D6FC2F42A85"	ISubclass.vb	3
UPGRADE_WARNING: App property App.EXENAME has a new behavior. Click for more: 'ms-help://MS.VSCC.v80/dv_commoner/local/redirect.htm?keyword=6BA9B8D2-2A32-4B6E-8D36-44949974A5B4"	Subclass_bas.vb	262
UPGRADE_WARNING: Add a delegate for AddressOf WindowProc Click for more: 'ms-help://MS.VSCC.v80/dv_commoner/local/redirect.htm?keyword=E9E157F7-EF0C-4016-87B7-7D7FB8C6EE08"	Subclass_bas.vb	354
UPGRADE_WARNING: Couldn't resolve default property of object ivwPT. Click for more: 'ms-help://MS.VSCC.v80/dv_commoner/local/redirect.htm?keyword=6A50421D-15FE-4896-8A1B-2EC21E9037B2"	Subclass_bas.vb	554
UPGRADE_WARNING: Couldn't resolve default property of object ivwPT. Click for more: 'ms-help://MS.VSCC.v80/dv_commoner/local/redirect.htm?keyword=6A50421D-15FE-4896-8A1B-2EC21E9037B2"	Subclass_bas.vb	557
UPGRADE_WARNING: Class instantiation was changed to public. Click for more: 'ms-help://MS.VSCC.v80/dv_commoner/local/redirect.htm?keyword=ED41034B-3890-49FC-8076-8D6FC2F42A85"	Subclass_cls.vb	3
UPGRADE_WARNING: Lower bound of array aTimers was changed from 1 to 0. Click for more: 'ms-help://MS.VSCC.v80/dv_commoner/local/redirect.htm?keyword=0F1C9BE1-AF9D-476E-83B1-17D438ECCFF20"	Timer_bas.vb	12
UPGRADE_WARNING: Add a delegate for AddressOf TimerProc Click for more: 'ms-help://MS.VSCC.v80/dv_commoner/local/redirect.htm?keyword=E9E157F7-EF0C-4016-87B7-7D7FB8C6EE08"	Timer_bas.vb	32
UPGRADE_WARNING: Couldn't resolve default property of object StoreTimer. Click for more: 'ms-help://MS.VSCC.v80/dv_commoner/local/redirect.htm?keyword=6A50421D-15FE-4896-8A1B-2EC21E9037B2"	Timer_bas.vb	151
UPGRADE_WARNING: App property App.EXENAME has a new behavior. Click for more: 'ms-help://MS.VSCC.v80/dv_commoner/local/redirect.htm?keyword=6BA9B8D2-2A32-4B6E-8D36-44949974A5B4"	Timer_cls.vb	35

Figure 45-6

Generally, the warnings and TODO tasks are added to the code where actual compilation warnings or errors will not occur. This is because while the Upgrade Wizard was able to successfully convert the code to what it expects is the correct Visual Basic 2005 code, you are encouraged to review its changes.

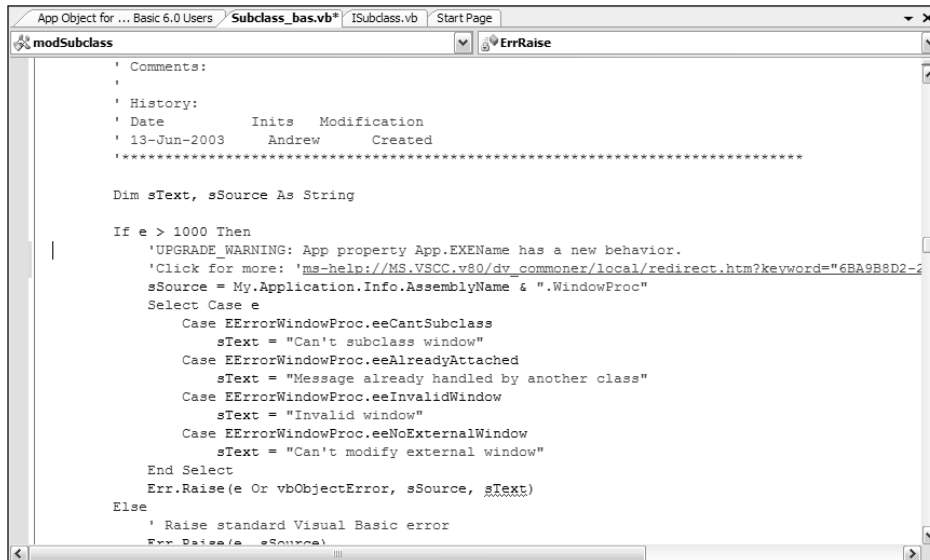
The majority of messages in this list will be of the variety that you can simply acknowledge and delete. For instance, Figure 45-6 shows a warning about the lower bound of an array being changed from 1 to 0. This is because arrays in Visual Basic 6 are 1-based, skipping over the 0th element, while Visual Basic 2005 follows the more common mechanism of including a 0th element in the array. While most of the time you can simply ignore the extra element in the array, some particular code constructs may exhibit different behavior.

The other warnings in the list may require more interaction. Again, looking at the sample list in Figure 45-6, there are some warnings about default properties for certain objects not being resolved. This situation causes a compilation error and needs to be addressed before the converted project can be built successfully.

Each entry in the Task List links to the associated line of code containing the problem. Double-clicking the entry in the list takes you directly to the line in question and the full warning message (see Figure 45-7).

Chapter 45

Every warning message also contains a link to a documentation entry in the MSDN Library explaining why there could be an issue with the particular statement. For example, the warning shown in Figure 45-7 indicates that the `App.EXENAME` was converted to `My.Application.Info.AssemblyName`, but that there are differences between the two objects. Clicking the link ultimately takes you to a help topic that explains in great detail any differences between the old Visual Basic 6 `App` object and its equivalent in Visual Basic 2005.



```
App Object for ... Basic 6.0 Users Subclass_bas.vb* ISubclass.vb Start Page
modSubclass
' Comments:
'
' History:
' Date      Inits  Modification
' 13-Jun-2003  Andrew  Created
'.....
Dim sText, sSource As String

If e > 1000 Then
  'UPGRADE_WARNING: App property App.EXENAME has a new behavior.
  'Click for more: 'ms-help://MS.VSCC.v80/dv_commoner/local/redirect.htm?keyword='6BA9B8D2-2...
  sSource = My.Application.Info.AssemblyName & ".WindowProc"
  Select Case e
    Case EErrorWindowProc.eeCantSubclass
      sText = "Can't subclass window"
    Case EErrorWindowProc.eeAlreadyAttached
      sText = "Message already handled by another class"
    Case EErrorWindowProc.eeInvalidWindow
      sText = "Invalid window"
    Case EErrorWindowProc.eeNoExternalWindow
      sText = "Can't modify external window"
  End Select
  Err.Raise(e Or vbObjectError, sSource, sText)
Else
  ' Raise standard Visual Basic error
  Err.Raise(e, sSource)
```

Figure 45-7

Errors and Warnings

Once you've reviewed the Task List, you should next turn to the Error List tool window (accessible via the `View` → `Error List` menu command). Because Visual Basic 2005 performs background compilation of code, any compilation errors that have been produced in the process of creating the new project are automatically flagged, added to the Error List, and marked in the code through the IntelliSense engine.

Figure 45-8 shows a sample Error List after the upgrade of a Visual Basic DLL project. This project was very small but contained a sizeable number of errors due to the purpose of the code. As mentioned previously, the `As Any` construct is not supported in Visual Basic and cannot be upgraded (you would need to define individual declarations for each variable type you were going to pass to the function). In addition, the code is using the unsupported `ObjPtr` function.

Some errors will be easier to fix. While the Upgrade Wizard may not be able to determine the default property of the `Item` object in a `ListView`, a human programmer can easily determine that the `Text` property was intended to be used, and correct it in just a few moments.

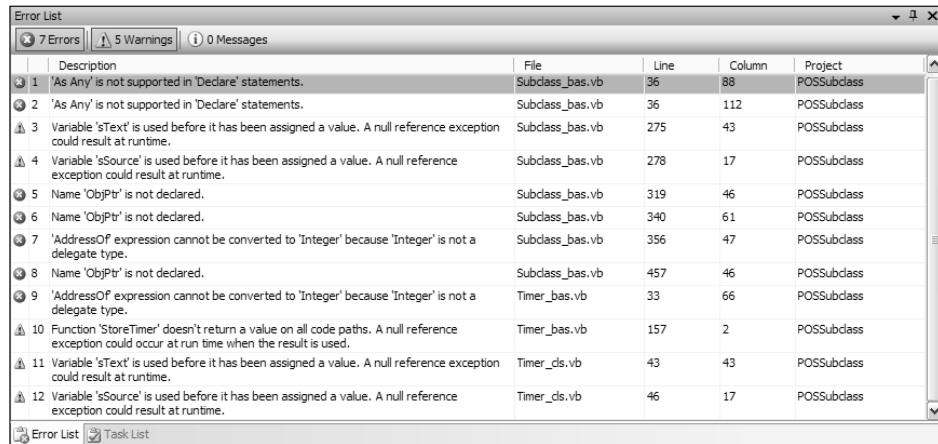


Figure 45-8

While compilation warnings don't stop your application from running, you should seriously consider and investigate each warning after an upgrade process to ensure that the expected behavior will occur. In the sample shown in Figure 45-8, errors 3 and 4 occur because `String` variables are accessed without first being assigned a value (you could leave these, or correct them by setting them to `vbNullString` in the declaration of each variable).

However, error 10 may be more serious depending on how the Upgrade Wizard handled the function. In some cases the wizard would be able to convert the appropriate statements to `Return` values, but if the return type is an unusual type it may not even be able to do that. Either way, you should definitely investigate the code that has produced the warning.

Upgrade Report

If you prefer a more visually pleasing view of the upgrade output, you can view the Upgrade Report. This report is generated after the Upgrade Wizard has completed and is placed in the project's folder with a name of `_UpgradeReport.htm`. It is then added to the Files list in the Solution Explorer.

Opening the page in a browser shows a nicely formatted report, as shown in Figure 45-9. At the top of the report is a summary showing how many errors and warnings were found during the upgrade process and of what type.

Underneath this summary section is a details table containing the errors per physical file contained in the project. This details view is handy for determining which parts of the project require the most work. Each of the tables can be collapsed and expanded to manage the detail.

Printing the report will only include the visible information; if you need to print the report in its entirety, make sure you first expand all nodes in both tables.

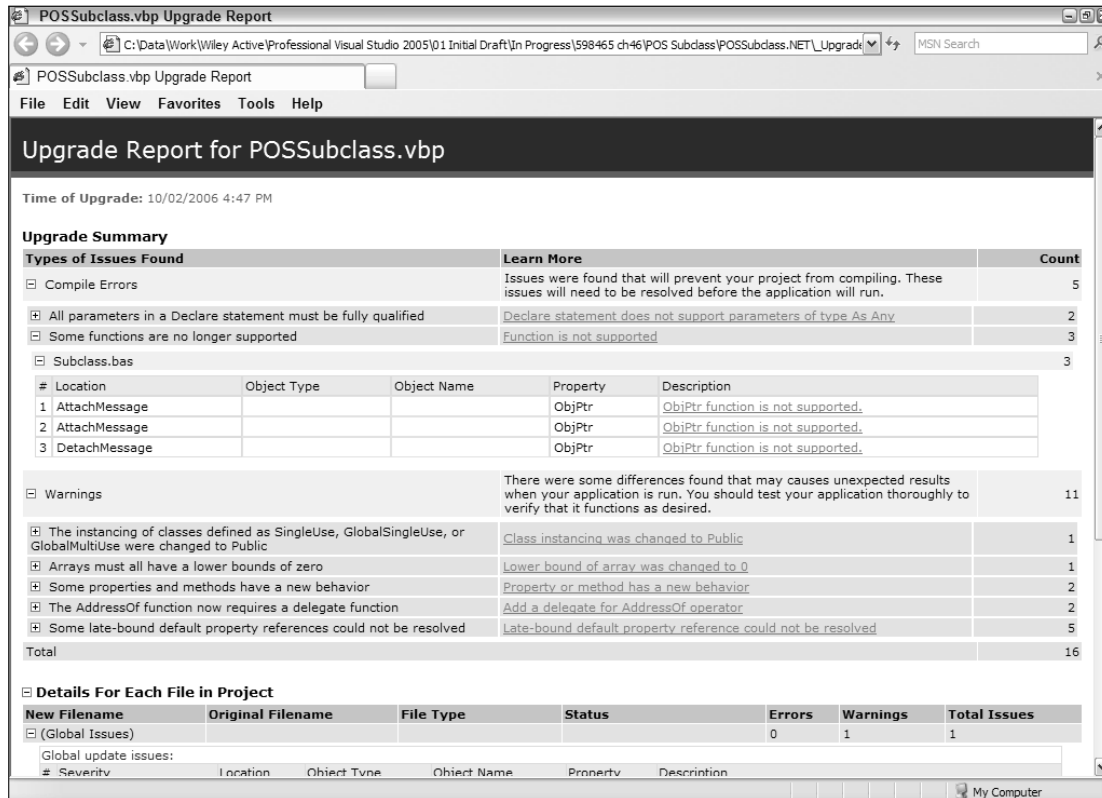


Figure 45-9

Unflagged Issues

Some issues can crop up in your conversion that are not flagged at all. These are usually related to situations that the Upgrade Wizard cannot determine, such as differences in literal value meanings.

A great example of this is the kinds of errors that can occur at runtime. When trapping errors using the `On Error` statement, often code is hard-coded with specific error numbers. Because the behavior may be different in Visual Basic 2005, different errors may be produced, which can cause unexpected results. The MSDN documentation has the following sample listing that illustrates this issue:

```

On Local Error GoTo Result
Dim x() As Boolean
Dim y As Variant

y = x(10)

Result:
If Err.Number = 9 Then
    ' Do something.
Else
    ' Do something else.
End If
    
```

When this code is run in Visual Basic 6 it will generate an `Err` object with a value of 9, which means “Subscript out of range.” However, because Visual Basic 2005 requires you to initialize any object (which includes arrays) before using it, you will get a different error number.

Because this kind of unseen issue can occur, it is always highly recommended to do a complete code review of upgraded projects.

The Upgrade Visual Basic 6 Tool

If you need only to convert a block of code instead of an entire project, whether it’s a single code statement or a full module, you will find the Upgrade Visual Basic 6 Code tool extremely useful. New to Visual Studio 2005, this utility is available from the Tool menu when you are editing a class or module in Code view.

When you start the utility, it displays a small dialog window containing a blank text area in which you can enter the Visual Basic 6 code to be converted (as shown in Figure 45-10). If the code uses COM objects, you can add the references to each COM object you require in the References tab.

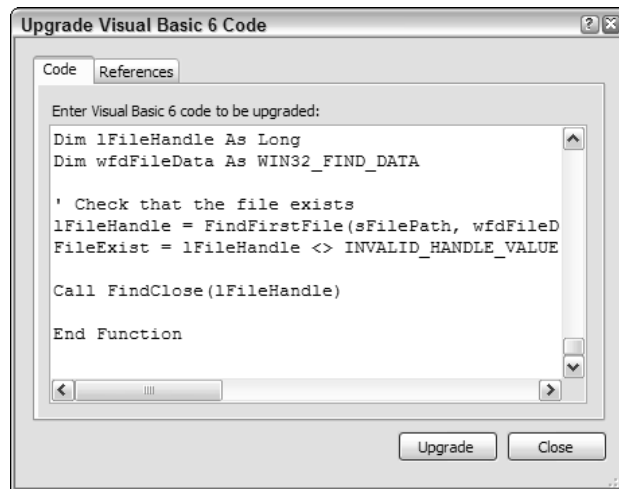


Figure 45-10

When the code and references are ready for conversion, click the Upgrade button to start the conversion process. This uses the same internal engine as the Upgrade Wizard, and automatically inserts the newly converted Visual Basic 2005 code into the module at the current insertion point. Any necessary COM references are automatically added to the project’s reference list.

Some additional errors may occur when converting a small block of Visual Basic code, such as converting a statement that includes an undefined variable. To avoid these, try to include everything that the particular code will need prior to clicking the Upgrade button.

Summary

Visual Studio 2005 comes with two utilities to migrate your Visual Basic 6 code over to the new language constructs — an Upgrade Wizard to convert complete projects to Visual Basic 2005, and a code snippet converter utility for converting individual functions or single modules of code. The two tools in conjunction enable you to upgrade Visual Basic 6 code more easily than ever, and along with the new features of Visual Studio 2005 (such as edit-and-continue), pave the way for Visual Basic 6 programmers to transition to .NET.

Contents

Acknowledgments	ix
Introduction	xxxv
Who This Book Is For	xxxv
What This Book Covers	xxxv
A Brief History of Visual Studio	xxxvi
One Comprehensive Environment	xxxvi
How This Book Is Structured	xxxviii
What You Need to Use This Book	xxxix
Conventions	xxxix
Source Code	xl
Errata	xl
p2p.wrox.com	xli
Part I: The Integrated Development Environment	1
Chapter 1: A Quick Tour of the IDE	3
Where to First?	3
IDE Structure	5
Getting Familiar with the IDE Structure	6
Basic Layout	6
Additional Windows	13
Summary	14
Chapter 2: Options	15
One with the Lot	15
Environment Options	16
Projects and Solutions	21
Text Editor	23
Debugging	25
Summary	27

Contents

Chapter 3: The Toolbox	29
Describing the Toolbox	29
Arranging Components	31
Adding Components	33
Commonly Used Elements	35
Summary	37
Chapter 4: The Solution Explorer	39
Solution Explorer Structure	39
Showing Hidden Items	40
Temporary Solutions	41
Web Solutions	42
Common Project and Solution Tasks	43
Adding Windows References	45
Adding Web References	46
Setting Solution Properties	46
Summary	47
Chapter 5: Customizing the IDE	49
Tool Window Customization	49
Working with Tool Windows	49
Moving Tool Windows	52
Importing and Exporting IDE Settings	55
Splitting Up the Workspace	57
Summary	58
Chapter 6: Form Design	59
The Form Itself	59
Form Design Preferences	63
Adding Controls to Your Form	64
Guidelines for Controls	65
Vertically Aligning Text Controls	66
Automatic Formatting of Multiple Controls	67
Setting Control Properties	69
Service-Based Components	71
Smart Tag Tasks	71
Additional Commands	72

Container Controls	73
Panel and SplitContainer	73
FlowLayoutPanel	74
TableLayoutPanel	75
Summary	76
Part II: Project and Solution Design	77
Chapter 7: Projects and Solutions	79
<hr/>	
Solution Structure	79
Solution File Format	81
Solution Properties	81
Common Properties	82
Configuration Properties	82
Project Types	84
Project File Format	84
Project Properties	84
Application	85
Compile	88
Debug	89
References	90
Resources	91
Settings	91
Signing	92
Security	93
Publish	94
Code Analysis	94
Creating a Custom Settings Provider	95
Summary	96
Chapter 8: Source Control	97
<hr/>	
Selecting a Source Control Repository	98
Environment Settings	99
Plug-In Settings	99
Accessing Source Control	99
Creating the Repository	100
Adding the Solution	101
Solution Explorer	101
Checking In and Out	102

Contents

Pending Changes	102
Merging Changes	103
History	104
Pinning	104
Source Control with Team Foundation	105
Source Control Explorer	105
Pending Changes	106
Shelving	108
Summary	109
Chapter 9: Application Configuration Files	111
Config Files	111
Machine.config	111
Web.config	111
App.config	112
Security.config	112
Configuration Schema	112
Configuration Attributes	113
Section: startup	114
Section: runtime	114
Section: system.runtime.remoting	115
Section: system.net	115
Section: cryptographySettings	116
Section: configurationSections	116
Section: system.diagnostics	116
Section: system.web	117
Section: webserver	117
Section: compiler	118
Application Settings	118
Using appSettings	118
Dynamic Properties	118
Custom Configuration Sections	119
Automation Using SCDL	122
IntelliSense	122
Summary	123
Chapter 10: XML Resource Files	125
Resourcing Your Application	125
What Are Resources?	127
Text File Resources	127
ResxResource Files	128

Adding Resources	129
Embedding Files as Resources	130
Accessing Resources	130
Resource Naming	130
Satellite Resources	130
Cultures	131
Creating Culture Resources	131
Loading Culture Resource Files	132
Satellite Culture Resources	132
Accessing Specifics	133
My Namespace	133
Bitmap and Icon Loading	133
ComponentResourceManager	133
Coding Resource Files	134
ResourceReader and ResourceWriter	135
ResxResourceReader and ResxResourceWriter	135
Custom Resources	136
Designer Files	140
Summary	140
Part III: Documentation and Research	141
Chapter 11: Help and Research	143
Accessing Help	143
Document Explorer	145
Dynamic Help	147
The Search Window	148
Sorting Results	149
Filtering Results	150
Keeping Favorites	151
Customizing Help	151
Ask a Question	152
Summary	153
Chapter 12: XML Comments	155
What Are XML Comments?	155
How to Add XML Comments	156
XML Comment Tags	156
The <c> Tag	157
The <code> Tag	157
The <example> Tag	158

Contents

The <exception> Tag	159
The <include> Tag	160
The <list> Tag	162
The <para> Tag	163
The <param> Tag	163
The <paramref> Tag	164
The <permission> Tag	165
The <remarks> Tag	165
The <returns> Tag	165
The <see> Tag	166
The <seealso> Tag	166
The <summary> Tag	168
The <typeparam> Tag	168
The <value> Tag	168
Using XML Comments	168
IntelliSense Information	169
Summary	170
Chapter 13: Control and Document Outline	171
Document Outline	171
Control Outline	173
Extra Commands in Control Outline Mode	174
Summary	175
Part IV: Security and Modeling	177
Chapter 14: Code Generation	179
Class Designer	179
Design Surface	180
Toolbox	181
Class Details	182
Properties Window	183
Layout	184
Exporting	184
Other Code-Generation Techniques	185
Snippets	185
Refactoring	185
Project and Item Templates	186
Strongly Typed Datasets	186
Forms	187
My Namespace	188

Taking Charge of the Class Designer	189
Class Diagram Schema	190
IntelliSense Code Generation	191
Object Test Bench	191
Invoking Static Methods	191
Instantiating Entities	192
Accessing Fields and Properties	193
Invoking Instance Methods	193
Summary	194
Chapter 15: Security Concepts	195
<hr/>	
Application Security	195
Code-Based Security	196
Role-Based Security	197
Summary	199
Chapter 16: Cryptography	201
<hr/>	
General Principles	201
Techniques	202
Hashing	202
Symmetric (Secret) Keys	202
Asymmetric (Public/Private) Keys	203
Signing	203
Summary of Goals	204
Applying Cryptography	204
Creating Asymmetric Key Pairs	204
Creating a Symmetric Key	206
Encrypting and Signing the Key	207
Verifying Key and Signature	209
Decrypting the Symmetric Key	210
Sending a Message	212
Receiving a Message	214
Miscellaneous	215
SecureString	216
Key Containers	217
Summary	218
Chapter 17: Obfuscation	219
<hr/>	
MSIL Disassembler	219
Decompilers	221

Contents

Obfuscating Your Code	222
Dotfuscator	222
Words of Caution	225
Attributes	227
ObfuscationAssembly	227
Obfuscation	228
Summary	229
Part V: Coding	231
Chapter 18: IntelliSense	233
IntelliSense Explained	233
General IntelliSense	234
Completing Words and Phrases	235
Parameter Information	238
Quick Info	238
IntelliSense Options	238
General Options	238
C#- and J#-Specific Options	240
Extended IntelliSense	241
Code Snippets	241
XML Comments	242
Adding Your Own IntelliSense	242
Summary	242
Chapter 19: Code Snippets	243
Code Snippets Revealed	243
Original Code Snippets	243
“Real” Code Snippets	244
Using Snippets in Visual Basic	245
Using Snippets in C# and J#	248
Creating Snippets Manually	249
Code Snippets Manager	254
Creating Snippets with VB Snippet Editor	256
Summary	261
Chapter 20: Regions and Bookmarks	263
Regions	263
Creating Regions	264
Using Regions	265
Introducing Outlining Commands	266

Visual Indicators	267
Color Coding	267
Margin Icons	268
Bookmarks and the Bookmark Window	269
Summary	271
Chapter 21: Refactoring	273
Accessing Refactoring Support	274
C# — Visual Studio 2005	274
VB.NET — Refactor!	274
Refactoring Actions	275
Extract Method	275
Encapsulate Field	277
Extract Interface	279
Reorder Parameters	280
Remove Parameters	281
Rename	282
Promote to Parameter	282
Generate Method Stub	283
Surround with Snippet	283
Summary	284
Chapter 22: Generics, Nullable Types, and Partial Types	285
Generics	285
Consumption	286
Creation	287
Constraints	288
Nullable Types	289
Partial Types	291
Form Designers	292
Operator Overloading	292
Operators	292
Type Conversions	293
Why Static Methods Are Bad	294
Predefined Delegates	295
Action	296
Comparison	296
Converter	297
Predicate	297
EventHandler	298

Contents

Property Accessibility	299
Custom Events	300
Summary	301
Chapter 23: Language-Specific Features	303
<hr/>	
C#	303
Anonymous Methods	303
Iterators	304
Static Classes	305
Naming Conflicts	306
Namespace Alias Qualifier	307
Global	307
Extern Aliases	308
Pragma	309
VB.NET	309
Continue	310
IsNot	310
Global	311
TryCast	311
Summary	312
Chapter 24: The My Namespace	313
<hr/>	
What Is the My Namespace?	314
The Main Components	315
Using My in Code	316
Using My in C#	316
Contextual My	317
Default Instances	320
My.Application	320
My.Computer	321
My.Computer.Audio	322
My.Computer.Clipboard	322
My.Computer.Clock	322
My.Computer.FileSystem	323
My.Computer.Info	323
My.Computer.Keyboard and My.Computer.Mouse	323
My.Computer.Network	324
My.Computer.Ports	324
My.Computer.Registry	324

My.Forms and My.WebServices	325
My For the Web	325
My.Resources	325
Other My Classes	327
Summary	327
Part VI: Automation	329
Chapter 25: Code Generation Templates	331
Creating Templates	331
Item Template	331
Project Template	335
Template Structure	335
Extending Templates	337
Template Project Setup	337
IWizard	339
Starter Template	342
Summary	344
Chapter 26: Macros	345
The Macro Explorer	345
Running Macros	346
Creating Macros	347
Recording Temporary Macros	348
Recording Issues	348
The Visual Studio Macros Editor	349
The DTE Object	351
Sample Macros	353
Building and Deploying	354
Summary	355
Chapter 27: Connection Strings	357
Data Source Connection Wizard	357
SQL Server Format	362
In-Code Construction	363
Encrypting Connection Strings	364
Summary	366

Contents

Chapter 28: Assembly Signing	367
Strong-Named Assemblies	367
The Global Assembly Cache	368
Signing an Assembly in VS 2005	368
Summary	369
Chapter 29: Preemptive Error Correction	371
Smart Compile Auto Correction	371
Customizing Warnings in Visual Basic	374
Warnings Not Displayed by Default	376
Other Customizable Warnings	377
Customizing Warnings in C#	380
Summary	381
Chapter 30: Strongly Typed DataSets	383
DataSet Overview	383
Adding a Data Source	384
DataSet Designer	387
Working with Data Sources	390
Web Service Data Source	391
Browsing Data	392
Summary	394
Chapter 31: Data Binding and Object Data Sources	395
Data Binding	395
BindingSource	397
BindingNavigator	398
Data Source Selections	400
BindingSource Chains	401
Saving Changes	407
Inserting New Items	409
Validation	410
DataGridView	417
Object Data Source	419
IDataErrorInfo	423
Application Settings	423
Summary	424

Chapter 32: Add-Ins	425
The Add-In Manager	425
Types of Add-Ins	426
Creating a Simple Add-In with the Wizard	427
Common Classes, Objects, and Methods	432
IDTExtensibility2	432
IDTCommandTarget	433
AddNamedCommand2	435
CreateToolWindow2	436
Debugging	436
Registration and Deployment	436
Summary	437
Chapter 33: Third-Party Extensions	439
Development Environment Enhancements	439
CoolCommands for VS2005	439
MZ-Tools	440
Code Aids	442
Imports Sorter	443
CodeKeep	443
Documentation	445
Testing and Debugging	446
Regex Visualizer	446
TestDriven.NET	446
Summary	447
Chapter 34: Starter Kits	449
The Card Game Starter Kit	450
The Screensaver Starter Kit	451
The Movie Collection Starter Kit	452
The Personal Web Site Starter Kit	453
Creating Your Own Starter Kit	454
Summary	454
Part VII: Other Time Savers	455
Chapter 35: Workspace Control	457
Visual Studio 2005 Windows	457
Start Page	457
Code/Designer	458

Contents

Solution Explorer	458
Properties	459
Toolbox	459
Server Explorer	460
Error List	460
Object Browser	461
Task List	461
Class View	462
Code Definition	462
Output	463
Find Results	463
Call Browser	463
Command Window	464
Document Outline	464
Object Test Bench	465
Performance Explorer	465
Property Manager	465
Resource View	466
History	466
Source Control Explorer	467
Pending Changes	467
Macro Explorer	468
Web Browser	468
Team Explorer	469
Breakpoints	469
Immediate	470
Script Explorer	470
Registers	470
Disassembly	471
Memory	471
Processes	471
Modules	472
Threads	472
Call Stack	472
Autos, Locals, and Watch	473
Code Coverage	473
Test Results	473
Test Manager	474
Test View	474
Team Builds	474
Test Runs	475
Bookmarks	475
Data Sources	475

Workspace Navigation	476
Full Screen Mode	476
Navigation Keys	476
Summary	478
Chapter 36: Find and Replace	479
<hr/>	
Introducing Find and Replace	479
Quick Find	480
Quick Replace	481
Quick Find and Replace Dialog Options	481
Find in Files	484
Find Dialog Options	484
Results Window	485
Replace in Files	486
Incremental Search	488
Find Symbol	489
Find and Replace Options	489
Summary	490
Chapter 37: Server Explorer	491
<hr/>	
The Servers Node	492
Event Logs	492
Management Classes	494
Management Events	496
Message Queues	499
Performance Counters	501
Services	504
Summary	505
Chapter 38: Visual Database Tools	507
<hr/>	
Database Windows in Visual Studio 2005	507
Server Explorer	508
Table Editing	510
Relationship Editing	512
Views	512
Stored Procedures and Functions	513
Database Diagrams	514
Data Sources Window	515

Contents

Using Databases	518
Editing Data Source Schema	518
Data Binding Controls	520
Data Controls	522
Managing Test Data	524
Previewing Data	525
Database Projects	526
Script-Based Database Projects	526
Managed Code Language-Based Database Projects	527
Summary	528
Chapter 39: Regular Expressions	529
Where Can Regular Expressions Be Used?	530
Regular Expression Programming	530
Find and Replace	530
Visual Studio Tools for Office Smart Tags	531
What Are Regular Expressions?	532
Using Regular Expressions to Replace Data	533
Regular Expression Syntax	534
Regular Expressions in .NET Programming	536
Regex	536
Match	537
MatchCollection	537
Replacing Substrings	538
Summary	538
Chapter 40: Tips, Hacks, and Tweaks	539
IDE Shortcuts	539
The Open With Dialog	539
Accessing the Active Files List	540
Changing Font Size	541
Making Rectangular Selections	542
Go To Find Combo	543
Forced Reformat	544
Word Wrapping	544
Registry Hacks	544
Vertical Guidelines	544
Right-Click New Solution	545
Keyword Color-Coding	547

Other Tips	548
Disable Add-Ins Loading on Startup	548
Multi-Monitor Layouts	548
Summary	549
Chapter 41: Creating Web Applications	551
<hr/>	
Creating Web Projects	551
Dynamic Compilation	554
Web Services	555
Personal Web Site Starter Kit	555
Web Development Options	556
HTML Text Editor Options	556
HTML Designer Options	557
Website Menu	558
Web Controls	558
General Property Settings	559
The Controls	560
Master/Detail Content Pages	568
Finalizing and Deployment	569
Deploying the Site	570
Site Administration	571
Security	572
Application Settings	574
ASP.NET 2.0 Configuration Settings	574
Summary	575
Chapter 42: Additional Web Techniques	577
<hr/>	
Web Development Revisited	577
The Sitemap	579
web.sitemap	579
The SiteMapPath Control	581
The SiteMapResolve Event	582
The Web Menu Control	584
Web Parts	585
WebPartManager	586
EditorZone	588
CatalogZone	590
Summary	592

Contents

Chapter 43: Building Device Applications **593**

Getting Started	593
.NET Compact Framework Versions	594
Solution Explorer	595
Design Skin	596
Orientation	596
Buttons	597
Toolbox	598
Common Controls	598
Mobile Controls	599
Debugging	605
Emulator	605
Device	606
Device Emulator Manager	607
Connecting	608
Cradling	608
Project Settings	609
Device Options	610
Summary	611

Chapter 44: Advanced Device Application Programming **613**

Data Source	613
DataSet	615
ResultSet	623
Windows Mobile 5.0	623
SDK Download	623
Managed APIs	624
Notification Broker	626
Deployment	627
CAB Files	628
MSI Installer	629
OpenNetCF Smart Devices Framework	632
Summary	633

Part VIII: Build and Deployment **635**

Chapter 45: Upgrading to Visual Studio 2005 **637**

The Upgrade Process	638
Getting Ready to Upgrade	638
Using the Upgrade Project Wizard	640
Checking the Upgrade Output	643

The Upgrade Visual Basic 6 Tool	647
Summary	648
Chapter 46: Build Customization	649
General Build Options	649
Batch Building	652
Manual Dependencies	652
Visual Basic Compile Page	654
Advanced Compiler Settings	654
Build Events	656
C# Build Pages	657
Advanced	658
MSBuild	660
How Visual Studio Uses MSBuild	660
MSBuild Schema	663
Summary	664
Chapter 47: Deployment: ClickOnce and Other Methods	665
Installers	665
Building an Installer	665
Customizing the Installer	669
Adding Custom Actions	673
Web Project Installers	675
Service Installer	676
ClickOnce	677
Click to Deploy	678
Click to Update	683
Other Techniques	684
XCopy	684
Publish Website	684
Copy Web Project	684
Summary	685
Part IX: Debugging and Testing	687
Chapter 48: Using the Debugging Windows	689
Code Window	689
Breakpoints	689
DataTips	690
Breakpoint Window	690

Contents

Output Window	691
Immediate Window	692
Script Explorer	692
Watch Windows	693
QuickWatch	693
Watch Windows 1–4	694
Autos and Locals	694
Call Stack	694
Threads	695
Modules	695
Processes	696
Memory Windows	696
Memory Windows 1–4	696
Disassembly	697
Registers	697
Exceptions	698
Customizing the Exception Assistant	699
Unwinding an Exception	700
Summary	701
Chapter 49: Debugging Breakpoints	703
<hr/>	
Breakpoints	703
Setting a Breakpoint	703
Adding Break Conditions	706
Working with Breakpoints	708
Tracepoints	709
Creating a Tracepoint	709
Tracepoint Actions	710
Execution Point	710
Stepping Through Code	711
Moving the Execution Point	712
Edit and Continue	712
Rude Edits	712
Stop Applying Changes	712
Summary	713
Chapter 50: Debugging Proxies and Visualizers	715
<hr/>	
Attributes	715
DebuggerBrowsable	715
DebuggerDisplay	716
DebuggerHidden	717

DebuggerStepThrough	717
DebuggerNonUserCode	718
Type Proxies	718
The Full Picture	720
Visualizers	720
Advanced Techniques	723
Saving Changes to Your Object	723
Summary	723
Chapter 51: Maintaining Web Applications	725
<hr/>	
Debugging	725
Breaking on Errors Automatically	727
Debugging an Executing Web Application	727
Error Handling	728
Tracing	729
Page-Level Tracing	729
Application-Level Tracing	731
Trace Output	731
Trace Viewer	732
Custom Trace Output	732
Summary	733
Chapter 52: Other Debugging Techniques	735
<hr/>	
Debugging Options Pages	735
General Options	735
Debug Page in My Project	738
Exception Assistant	739
Debugging Macros	741
Debugging Database Stored Procedures	742
Summary	742
Chapter 53: Unit Testing	743
<hr/>	
Your First Test Case	743
Test Attributes	748
Test Attributes	749
Asserting the Facts	750
Assert	751
StringAssert	751
CollectionAssert	752
ExpectedException Attribute	752

Contents

Initializing and Cleaning Up	753
More Attributes	753
Testing Context	753
Data	754
Writing Test Output	755
Advanced	756
Custom Properties	756
Testing Private Members	758
Summary	760
Part X: Extensions for Visual Studio 2005	761
Chapter 54: InfoPath 2003 Toolkit	763
Creating Managed InfoPath Solutions	763
The Generated Solution	765
Switching Between Visual Studio and InfoPath	767
Adding Code to InfoPath Forms	768
Form-Related Events	768
Field Events	773
The Button Click Event	774
Other Considerations	776
Summary	776
Chapter 55: Visual Studio Tools for Office	777
The New Visual Studio Tools for Office	778
The Visual Designer	780
Control Design	781
Writing Code	782
The Actions Pane	784
Smart Tags	785
Microsoft Outlook Add-Ins	787
The VSTO 2005 Sample Project	788
Summary	800
Chapter 56: Visual Studio Team System	801
Team System Editions	801
For Everyone	801
For Software Architects	807
For Software Developers	811
For Software Testers	818

Advanced	825
Writing Custom Code Analysis Rules	825
Customizing the Process Templates	828
Summary	830
Index	831

Professional Visual Studio® 2005

Published by

Wiley Publishing, Inc.

10475 Crosspoint Boulevard

Indianapolis, IN 46256

www.wiley.com

Copyright © 2006 by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN-13: 978-0-7645-9846-3

ISBN-10: 0-7645-9846-5

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

1MA/QT/QY/QW/IN

Library of Congress Cataloging-in-Publication Data

Parsons, Andrew, 1970-

Visual studio 2005 / Andrew Parsons and Nick Randolph.

p. cm.

Includes index.

ISBN-13: 978-0-7645-9846-3 (paper/website)

ISBN-10: 0-7645-9846-5 (paper/website)

1. Microsoft Visual studio.
 2. Microsoft .NET Framework.
 3. Web site development—Computer programs.
 4. Application software—Development—Computer programs.
- I. Randolph, Nick. 1978- II. Title.

TK5105.8885.M57P38 2006

006.786—dc22

2006014685

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Legal Department, Wiley Publishing, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, (317) 572-3447, fax (317) 572-4355, or online at <http://www.wiley.com/go/permissions>.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

For general information on our other products and services please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Trademarks: Wiley, the Wiley logo, Wrox, the Wrox logo, Programmer to Programmer, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. Visual Studio is a registered trademark of Microsoft Corporation in the United States and/or other countries. All other trademarks are the property of their respective owners. Wiley Publishing, Inc., is not associated with any product or vendor mentioned in this book.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

About the Authors

Andrew Parsons

Andrew Parsons is an accomplished programmer, journalist, and author. He created, launched, and served as chief editor for *Australian Developer* magazine, which was so successful that it expanded globally and is now known as *International Developer*. Subsequent to that success, Parsons launched the local Australian and New Zealand edition of *MSDN* magazine. In addition, he has written a variety of technical books, including topics as diverse as HTML and CSS, Photoshop, and Visual Basic Express. When not writing, Parsons consults on .NET programming implementations for a number of clients, and currently serves as a senior consultant at Readify Pty, Ltd (www.readify.net), as well as running his own business, Parsons Designs (www.parsonsdesigns.com), and GAMEparents (www.gameparents.com), a website dedicated to helping parents understand and enjoy computer and video games.

Nick Randolph

Nick Randolph is an experienced .NET developer and solution architect. During his time with Software Engineering Australia, a not-for-profit industry body, Nick founded the Perth .NET Community of Practice and has been integrally involved in the local .NET community since. When Nick joined AutumnCare (www.autumncare.com.au) as Development Manager, he was responsible for their product architecture, which incorporated best practices around building smart client applications using the .NET Framework. Nick is currently a solutions architect with SoftTeq (<http://softteq.com>), which provides consulting, training, and mentoring services. Outside of his consulting role, Nick takes a proactive approach toward technology, ever seeking to learn, use, and present on beta products. As a Microsoft MVP, Nick has been invited to present at IT conferences such as TechEd, MEDC, and Code Camp, and has been a worldwide finalist judge for the Microsoft Imagine Cup for the last two years.