Certified Visual Basic 2005 Programmer Sample Material

V-Skills Certifications

A Government of India
&
Government of NCT Delhi Initiative

V-Skills

Skills for a secure future

1. VISUAL STUDIO

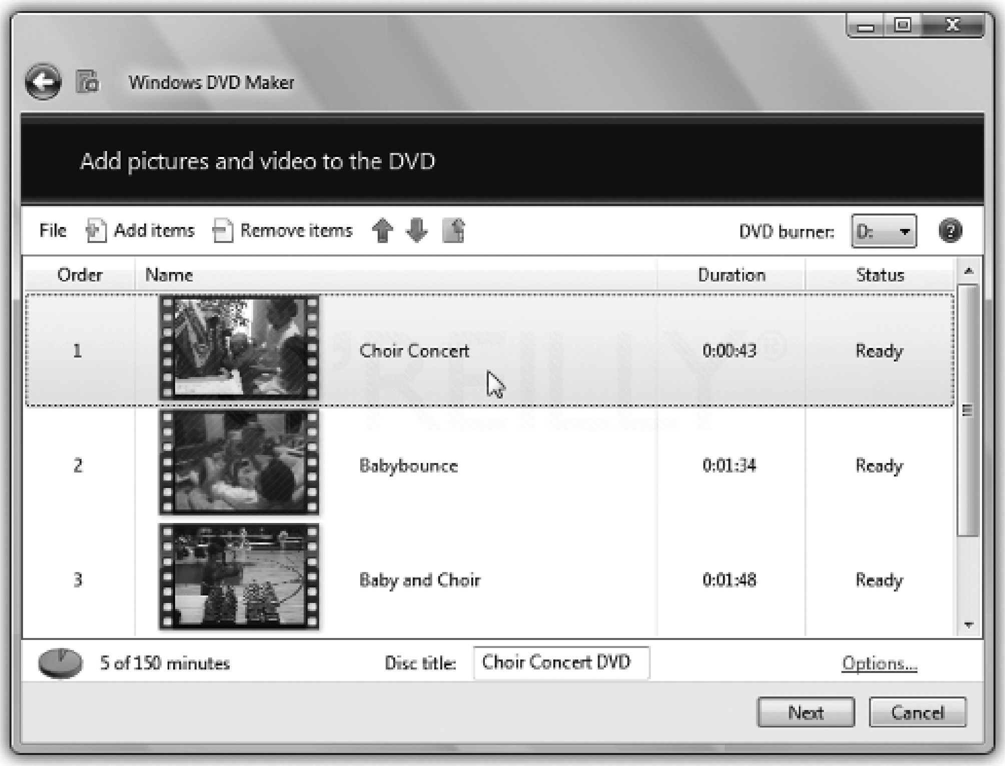
The new features of Visual Basic 2005 are actually provided by three separate components: the enhanced Visual Studio 2005 IDE, a new version of the VB compiler (vbc.exe), and the revamped .NET 2.0 Framework. In this chapter, you'll start by taking Visual Studio 2005 for a spin.

At first glance, Visual Studio hasn't changed too radically in its latest incarnation. However, it's worth taking a moment to orient yourself to Microsoft's newest IDE.

Visual Studio 2005 is the direct successor to Visual Studio .NET, and it provides the most complete set of tools and features. Visual Basic 2005 Express Edition allows you to build Windows applications, console applications, and DLL components (but not web applications). Visual Web Developer 2005 Express Edition allows you to build only web applications. However, all three of these programs are really variations of the same tool—Visual Studio. As a result, the menus, toolbars, and behavior of these applications are essentially the same.

1.1. How do I do that?

To get started and create a new project, select File → New Project from the Visual Studio menu. You'll see a slightly revamped New Project dialog box, as shown in Figure 1-1. Depending on the version of Visual Studio you're using, you may see a different set of available project types.



Order	Name	Duration	Status
1	Choir Concert	0:00:43	Ready
2	Babybounce	0:01:34	Ready
3	Baby and Choir	0:01:48	Ready

5 of 150 minutes

Disc title: Choir Concert DVD

Figure 1-1. Creating a new project

To continue, select the Windows Application project type and click OK to create the new project. In the Solution Explorer, you'll see that the project contains a single form, an application configuration file, and a My Project node (which you can select to configure project and build settings). However, the list of assembly references won't appear in the Solution Explorer, unless you explicitly choose Project → Show All Files. Figure 1-2 shows both versions of the Solution Explorer.

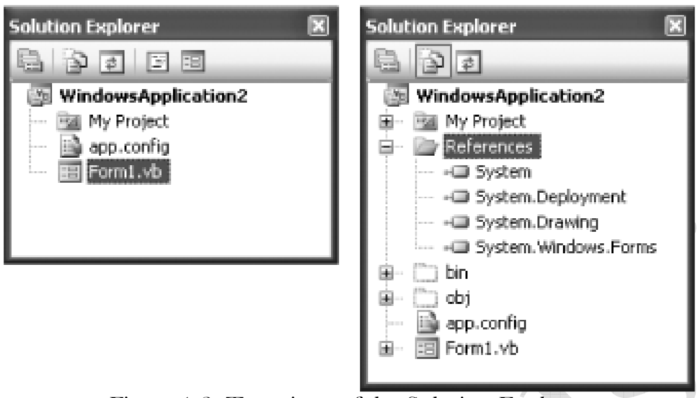


Figure 1-2. Two views of the Solution Explorer

To save your project, choose File → Save [ProjectName] from the menu. One change you're likely to notice is that Visual Studio no longer asks you to specify a directory path when you create a new project. That's because Visual Studio, in a bid to act more like Visual Basic 6, doesn't save any files until you ask it to.

This behavior actually depends on the Visual Studio environment settings. When you first install Visual Studio, you have the chance to choose your developer profile. If you choose Visual Basic Development Settings, you won't be asked to save your project when you first create it.

Of course, as a savvy programmer you know that files need to reside somewhere, and if you dig around you'll find a temporary directory like C:\Documents and Settings\[UserName]\Local Settings\Application Data\Temporary Projects\[ProjectName] that's used automatically to store new, unsaved projects. Once you save a project, it's moved to the location you choose.
The process of creating web applications has also changed subtly in Visual Studio 2005, and you no longer need IIS and a virtual directory to test your web site. You'll learn more about web projects in Chapter 4.

You can use the simple Windows application you created to try out the other labs in this chapter and tour Visual Studio's new features.

What about...

...the real deal of differences between different Visual Studio flavors? You can get the final word about what each version does and does not support from Microsoft's Visual Studio 2005 developer center, at http://lab.msdn.microsoft.com/vs2005. This site provides downloads of the latest Visual Studio betas and white papers that explain the differences between the express editions and the full-featured Visual Studio 2005.

1.2. Code, Debug, and Continue Without Restarting Your Application

Visual Basic 6 developers are accustomed to making changes on the fly, tweaking statements, refining logic, and even inserting entirely new blocks of code while they work. But the introduction of a new compile-time architecture with the .NET 1.0 common language runtime (CLR) caused this feature to disappear from Visual Studio .NET 2002 and 2003. Fortunately, it's returned in Visual Basic 2005, with a few enhancements and one major caveat—it won't work with ASP.NET. The single most requested feature from VB 6 returns to . NET: a debugger that lets you edit code without restarting your application.

How do I do that?

To see edit-and-debugging at its simplest, it's worth looking at an example where a problem sidelines your code—and how you can quickly recover. Figure 1-3 shows a financial calculator application that can calculate how long it will take you to become a millionaire, using Visual Basic's handy Pmt( ) function.

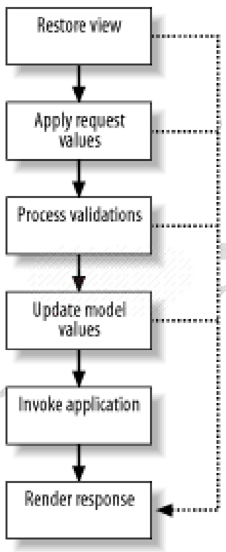


Figure 1-3. A simple form for a financial calculation

To create this program, first add four text boxes (the labels are optional), and then name them txtInterestRate, txtYears, txtFutureValue, and txtMonthlyPayment (from top to bottom). Then, add a button with the following event handler:

```
Private Sub btnCalculate_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles btnCalculate.Click

    Dim InterestRate, Years, FinalValue, MonthlyPayment As Double
    InterestRate = Val(txtInterestRate.Text)
    FinalValue = Val(txtFutureValue.Text)
    MonthlyPayment = Pmt(InterestRate / 12 / 100, _
        Years * 12, 0, -FinalValue, DueDate.BegOfPeriod)
    txtMonthlyInvestment.Text = MonthlyPayment.ToString("C")

End Sub
```

Now run the application, enter some sample values, and click the button. You'll receive a runtime exception (with the cryptically worded explanation "Argument NPer is not a valid value") when your code tries to calculate the MonthlyPayment value. One way to discover the source of the problem is to move the mouse over all the parameters in the statement and verify that they reflect what you expect. In this case, the problem is that the Years variable is never set, and so contains the value 0.

Thanks to edit-and-continue debugging, you can correct this problem without restarting your application. When the error occurs, click the "Enable editing" link in the error window. Then, add the missing line:

```
Private Sub btnCalculate_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles btnCalculate.Click

    Years = Val(txtYears.Text)
    ...

End Sub
```

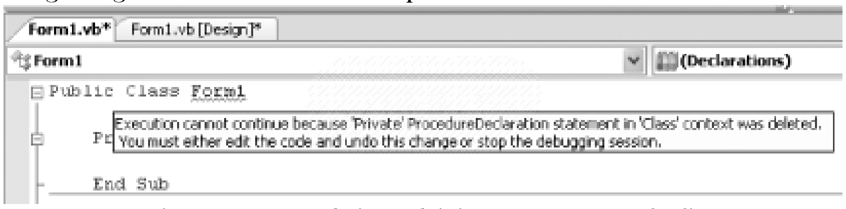
Now, look for the yellow arrow in the margin that indicates where the debugger is in your code. Click and drag this yellow arrow up to the newly added line so that it will be executed next. When you press F5 or click the Start button, the code resumes from this point and the calculation completes without a hitch.

You don't need to wait for an error to occur to use edit-and-continue debugging. You can also set a breakpoint in your code or select Debug → Break from the menu at any time.

What about...

...changes that the edit-and-continue debugger doesn't support? For the most part, edit-and-continue debugging in Visual Basic 2005 supports a greater variety of edits than supported in Visual Basic 6. However, there are still some types of edits that require you to restart your application. One example is if you delete the method in which your code is currently executing. For the full list, refer to the MSDN help, under the index entry "Edit and Continue → unsupported declaration edits" (which describes disallowed changes to declarations, like properties, methods, and classes) and "Edit and Continue → unsupported property and method body edits" (which describes disallowed changes inside your actual code routines).

To alert you when you make an unsupported edit, Visual Studio underlines the declaration of the current class with a green squiggly line. If you hover over that line, a ToolTip appears that explains the offending change. Figure 1-4 shows an example.



```
Public Class Form1

    Pr
    
    End Sub
```

Execution cannot continue because 'Private' ProcedureDeclaration statement in 'Class' context was deleted. You must either edit the code and undo this change or stop the debugging session.

Figure 1-4. A ToolTip explaining an unsupported edit

At this point, you can either undo this change, or continue (knowing that you'll need to restart the program). If you attempt to continue execution (by pressing F5 or F8), Visual Studio asks whether you want to stop debugging or want to cancel the request and continue editing your code.

A more significant limitation of the new edit-and-continue feature is that it doesn't support ASP.NET web applications. However, Visual Basic (and C#) developers still receive some improvement in their web-application debugging experience. Visual Studio 2005 compiles each web page separately, rather than into a single assembly (as was the model in previous versions). As a result, when you find some misbehaving code in a web page, you can pause the debugger, edit the code, and refresh the web page by clicking Refresh in your browser. This behavior gives you an experience that's similar to edit-and-continue, but it only works on a per-page basis. Unfortunately, this feature won't help you if you're in the middle of debugging a complex routine inside a web page. In that case, you'll still need to re-request the web page after you make the change and start over.

1.3. Look Inside an Object While Debugging

Visual Studio has always made it possible for you to peer into variables while debugging your code, just by hovering over them with the mouse pointer. But there were always limitations. If the variable was an instance of an object, all you could see was the value returned by the ToString( ) method, which more often than not was simply the fully qualified name of the class itself. Moreover, you couldn't see the content of public properties and indexers. The Watch and Locals windows provided some improvement, but they weren't quite as convenient or intuitive. Visual Studio 2005 changes the picture with a new feature called debugger DataTips.

In Visual Studio 2005, it's even easier to take a look at the content of complex objects while debugging.

How do I do that?

To use debugger DataTips, it helps to have a custom class to work with. The code in Example 1-1 shows the declaration for two very simple classes that represent employees and departments, respectively.

Example 1-1. Two simple classes
```
Public Class Employee
   Private _ID As String
   Public ReadOnly Property ID( ) As String
      Get
         Return _ID
      End Get
   End Property

   Private _Name As String
   Public ReadOnly Property Name( ) As String
      Get
         Return _Name
      End Get
```

```
    End Property

    Public Sub New(ByVal id As String, ByVal name As String)
        _ID = id
        _Name = name
    End Sub
End Class

Public Class Department
    Private _Manager As Employee
    Public ReadOnly Property Manager( ) As Employee
        Get
            Return _Manager
        End Get
    End Property

    Private _DepartmentName As String
    Public ReadOnly Property Name( ) As String
        Get
            Return _DepartmentName
        End Get
    End Property

    Public Sub New(ByVal departmentName As String, ByVal manager As Employee)
        _DepartmentName = departmentName
        _Manager = manager
    End Sub
End Class
```

Now you can add some code that uses these objects. Add the following event handler to any form to create a new Employee and Department object when the form first loads.

```
Private Sub Form1_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load

    Dim Manager As New Employee("ALFKI", "John Smith")
    Dim Sales As New Department("Sales", Manager)

End Sub
```

Now place a breakpoint on the final End Sub, and run the application. When execution stops on the final line, hover over the Sales variable. An expanded ToolTip will appear that lists every private and public member of the object.

Even better, if one object references another, you can drill into the details of both objects. To try this out, click the plus sign (+) sign next to the Manager property to see the linked Employee object. Figure 1-5 shows the DataTip you'll see.

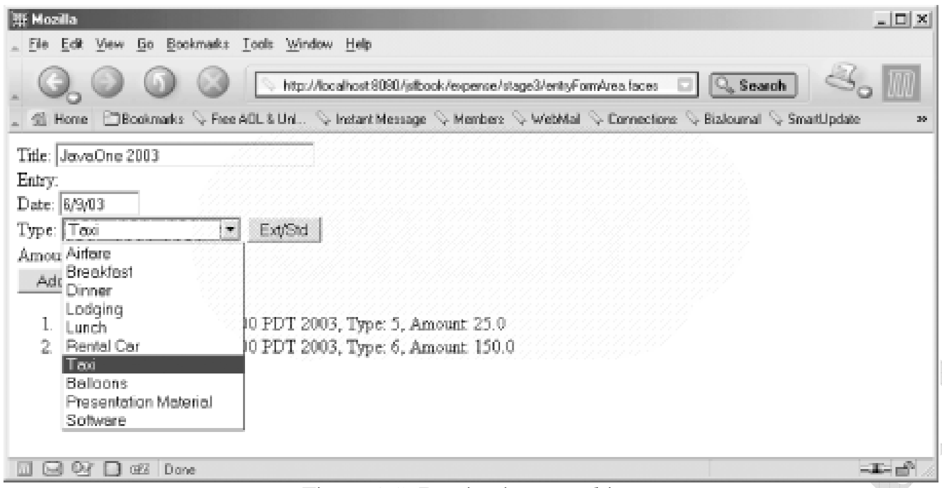

Mozilla
File Edit View Go Bookmarks Tools Window Help
http://localhost:8080/jsfbook/expense/stage3/entryFormArea.faces
Search
Home Bookmarks Free AOL & Unl.. Instant Message Members WebMail Connections BizJournal SmartUpdate
Title: JavaOne 2003
Entry:
Date: 6/9/03
Type: Taxi
Ext/Std
Airfare
Breakfast
Dinner
Lodging
Lunch
Rental Car
Taxi
Balloons
Presentation Material
Software
0 PDT 2003, Type: 5, Amount: 25.0
0 PDT 2003, Type: 6, Amount: 150.0
Done

Figure 1-5. Peering into an object

Using debugger DataTips, you can also edit simple data types on the fly. Just double click the property or variable, and edit the value. In Figure 1-5, the private variable _Name is currently being edited.

What about...

...working with exotic types of data? Using the .NET Framework, it's possible to create design-time classes that produce customized visualizations for specific types of data. While this topic is outside the scope of this book, you can see it at work with the three built-in visualizers for text, HTML, and XML data.

For example, imagine you have a string variable that holds the content of an XML document. You can't easily see the whole document in the single-line ToolTip display. However, if you click the magnifying glass next to the content in the ToolTip, you'll see a list of all the visualizers you can use. Select XML Visualizer, and a new dialog box will appear with a formatted, color-coded, scrollable, resizable display of the full document content, as shown in Figure 1-6.

Mozilla
File Edit View Go Bookmarks Tools Window Help
http://localhost:8080/jsfbook/expense/stage1/menuArea.faces
Search
Home Bookmarks Free AOL & Unl.. Instant Message Members WebMail Connections BizJournal SmartUpdate
New Delete Submit
You're logged in as "hans" [Logout] [Preferences]

Figure 1-6. Viewing XML content while debugging

1.4. Diagnose and Correct Errors on the Fly

Visual Studio does a great job of catching exceptions, but it's not always as helpful at resolving them. The new Exception Assistant that's hardwired into Visual Studio 2005 gives you a head start. Stumbled into a head-scratching exception? Visual Studio 2005 gives you a head start for resolving common issues with its Exception Assistant.

How do I do that?

You don't need to take any steps to activate the Exception Assistant. Instead, it springs into action as soon as your program encounters an unhandled exception. To see it in action, you need to create some faulty code. A good test is to add the following event handler to any form, which tries to open a non-existent file:

```
Private Sub Form1_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load

    Dim XMLText As String = My.Computer.FileSystem.ReadAllText( _
      "c:\FileDoesNotExist")

End Sub
```

Now run the application. When the error occurs, Visual Studio switches into break mode and highlights the offending statement. The Exception Assistant then appears, with a list of possible causes for the problem. Each suggestion appears as a separate link in the pop-up window. If you click one of these links, the full MSDN help topic will appear. Figure 1-7 shows the result with the faulty file-reading code; the Exception Assistant correctly identifies the reason that the attempt to open the file failed.

Figure 1-7. Getting help with an exception

This example uses a new VB language feature—the My object. You'll learn much more about My objects in the next chapter.

If you want to see the low-level exception information, click the View Detail link at the bottom of the window. This pops up a dialog box with a PropertyGrid showing all the information of the associated exception object. This change alone is a great step forward from Visual Studio .NET 2003, where you needed to write a Catch exception handler and set a breakpoint to take a look at the underlying exception object.

What about...

...solving complex problems? The Exception Assistant isn't designed to help you sort through issues of any complexity. Instead, it works best at identifying the all-too-common "gotchas," such as trying to use a null reference (usually a result of forgetting to use the New keyword) and failing to convert a data type (often a result of an inadvertent type cast).

Where can I learn more?

For help in the real world, consult a colleague or one of the many .NET discussion groups. Some good choices include http://lab.msdn.microsoft.com/vs2005/community (for the latest on Visual Basic 2005) and—once Visual Basic 2005 enters its release phase—http://www.windowsforms.net/Forums (for Windows Forms questions), http://www.asp.net/Forums (for ASP.NET issues), and http://discuss.develop.com/advanced-dotnet.html (for more advanced .NET queries).

1.5. Rename All Instances of Any Program Element

Symbolic rename allows you to rename all instances of any element you declare in your program, from classes and interfaces to properties and methods, in a single step. This technique, which is decidedly not a simple text search-and-replace feature by virtue of its awareness of program syntax, solves many knotty problems found in previous releases of Visual Basic. For example, imagine you want to rename a public property named FirstName. If you use search-and-replace, you'll also inadvertently affect a text box named txtFirstName, an event handler named cmdFirstName_Click, a database field accessed through row("FirstName"), and even your code comments. With symbolic rename, the IDE takes care of renaming just what you want, and it completes all of its work in a single step.

Need to rename a method, property, or variable without mangling other similar names in the same file? Visual Studio 2005 includes the perfect antidote to clumsy search-and-replace.

How do I do that?

You can use symbolic rename from any code window. To understand how it works, create a form that has a single text box named TextBox1 and a button named cmdText. Finally, add the form code in Example 1-2.

Example 1-2. A simple form that uses the word "Text" heavily

```
Public Class TextTest

  Private Sub TextTest_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
      ' Get the text from the text box.
      Dim Text As String = TextBox1.Text
```

```
        ' Convert and display the text.
        Text = ConvertText(Text)
        MessageBox.Show("Uppercase Text is: " & Text)
    End Sub

    Public Function ConvertText(ByVal Text As String) As String
        Return Text.ToUpper( )
    End Function

End Class
```

This code performs a relatively mundane task: converting a user-supplied string to uppercase and displays it in a message box. What's notable is how many places it uses the word "Text." Now, consider what happens if you need to rename the local variable Text in the event handler for the Form.Load event. Clearly, this is enough to confuse any search-and-replace algorithm. That's where symbolic rename comes in.

To use symbolic rename, simply right-click on the local Text variable, and select Rename from the context menu. In the Rename dialog box, enter the new variable name LocalText and click OK. All the appropriate instances in your code will be changed automatically without affecting other elements in your code (such as the text box, the comments, the literal text string, the form class name, the Text parameter in the ConvertText function, and so on). Here's the resulting code:

```
Public Class TextTest

    Private Sub cmdTest_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles cmdText.Click
        ' Get the text from the text box.
        Dim LocalText As String = TextBox1.Text

        ' Convert and display the text.
        LocalText = ConvertText(LocalText)
        MessageBox.Show("Uppercase Text is: " & LocalText)
    End Sub

    Public Function ConvertText(ByVal Text As String) As String
        Return Text.ToUpper( )
    End Function

End Class
```

Symbolic rename works with any property, class, or method name you want to change. Here are a few important points to keep in mind about how symbolic rename works:

- ✓ If you rename a class, all the statements that create an instance of that class are also changed.
- ✓ If you rename a method, all the statements that call that method are also changed.

- ✓ If you change a variable name that is the same as a method name, only the variable is changed (and vice versa).
- ✓ If you change a local variable name that is the same as a local variable name with different scope (for example, in another method), only the first variable is affected.

The symbolic rename feature isn't immediately impressive, but it's genuinely useful. Particularly noteworthy is the way it properly observes the scope of the item you want to rename. For example, when you rename a local variable, your changes don't spread beyond the current procedure. On the other hand, renaming a class can affect every file in your project.

Note that if you change the name of a control variable, your code will also be updated accordingly. However, there's one exception—the names of event handlers are never modified automatically. For example, if you change Button1 to Button2, all the code that interacts with Button1 will be updated, but the event handler subroutine Button1_Click will not be affected. (Remember, the name of the event handler has no effect on how it works in your application, as long as it's connected with the Handles clause.)

In Visual Studio 2005, when you rename a .vb file in the Solution Explorer, the name of the class in the file is also renamed, as long as the file contains a class that has the old name. For example, if you rename Form1.vb to Form2.vb and the file contains a class named Form1, that class will be renamed to Form2. Any code statements that create an instance of Form1 will also be updated, no matter where they reside in the project. However, if you've already changed the class name to something else (like MyForm), the class name won't be affected when you rename the file. In Visual Studio 2002 and 2003, the same action of renaming a form file has no effect on your code, so it's worth noting.

What about...

...support in Visual Basic 2005 for C# refactoring? Unfortunately, many of the additional refactoring features that Visual Studio provides to C# programmers don't appear in Visual Basic at all. Symbolic rename is one of the few new refactoring features that's alive and well for VB programmers in this release.

1.6. Use IntelliSense Filteringand AutoCorrect

IntelliSense is one of the great conveniences of Visual Studio, and it continues to improve in Visual Studio 2005, with two new features that make it more useful: IntelliSense filtering and AutoCorrect. IntelliSense filtering restricts the number of options you see to those that make sense in the current context. AutoCorrect goes one step further by recommending ways to resolve syntax errors, rather than simply reporting them.

Visual Studio 2005 makes IntelliSense more intelligent by restricting class names that aren't relevant and suggesting corrections you can apply to resolve syntax errors.

How do I do that?

There's no need for any extra steps when you use IntelliSense filtering—it's at work automatically. As you enter code, IntelliSense prompts you with lists of classes, properties, events, and more. In Visual Studio 2005, this list is tailored to your immediate needs, based on various contextual details. For example, if you're selecting an attribute to apply to a method, the IntelliSense list will show only classes that derive from the base Attribute class.

To see the new IntelliSense in action, start typing an exception-handling block. When you enter the Catch block, the IntelliSense list will show only classes that derive from the base Exception class (as shown in Figure 1-8). Select the Common or All tab at the bottom of the list, depending on whether you want to see the most commonly used classes or every possibility.

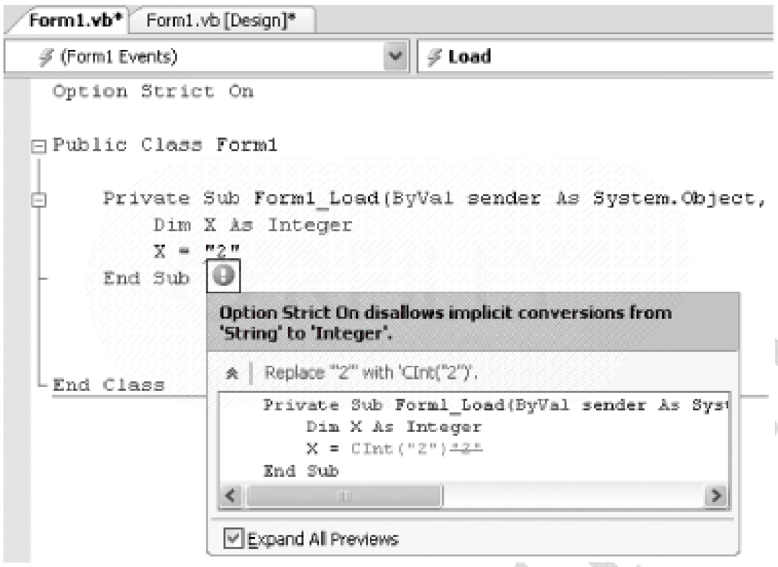
Figure 1-8. Filtering for Exception classes only

AutoCorrect is an IntelliSense improvement that targets syntax errors. Every time Visual Studio discovers a problem, it underlines the offending code in blue. You can hover over the problem to see a ToolTip with error information. With AutoCorrect, Visual Studio also adds a red error icon that, when clicked, shows a window with the suggested correction.

To see AutoCorrect in action, enter the following code (which attempts to assign a string to an integer without proper type-casting code):

```
Dim X As Integer
X = "2"
```

Option Strict catches data type conversion errors at compile time. To switch it on, double-click My Project in the Solution Explorer, click the Compile tab, and look for the Option Strict drop-down listbox. Assuming you have Option Strict switched on, you'll see a red error icon when you hover over this line. Click the red error icon. The AutoCorrect window that appears shows your code in blue, code to be added in red, and code to be removed crossed out with a solid line. Figure 1-9 shows the correction offered for this code snippet.



```
Option Strict On

Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object,
        Dim X As Integer
        X = "2"
    End Sub

End Class
```

Option Strict On disallows implicit conversions from 'String' to 'Integer'.

Replace "2" with 'CInt("2")'.

```
Private Sub Form1_Load(ByVal sender As Sys
    Dim X As Integer
    X = CInt("2")~~"2"~~
End Sub
```

Figure 1-9. Intellisense AutoCorrect in action

Other problems that AutoCorrect can resolve include class names that aren't fully qualified, misspelled keywords, and missing lines in a block structure. In some cases, it will even show more than one possible correction.

What about...

...doing more? There's still a lot of additional intelligence that IntelliSense could provide, but doesn't. For example, when assigning a property from a class to a string variable, why not show only those properties that return string data types? Or when applying an attribute to a method, why not show attribute classes that can be applied only to methods? As computer processors become faster and have more and more idle cycles, expect to see new levels of artificial intelligence appearing in your IDE.

1.7. Edit Control Properties in Place

The Properties window in Visual Studio makes control editing easy, but not always fast. For example, imagine you want to tweak all the text on a form. In previous versions of Visual Studio, the only option was to select each control in turn and modify the Text property in the Properties window one at a time. Although this approach isn't necessarily awkward, it certainly isn't as easy as it could be. In Visual Studio 2005, you can adjust a single property for a series of controls directly on the form.

When you need to update a single property for a number of different controls, in-place property editing makes it easy.

How do I do that?

To try in-place property editing, create a new form and add an assortment of controls. (The actual controls you use don't really matter, but you should probably include some text boxes, buttons, and labels.) Then, select View → Property Editing View. Finally, choose the property you want to change from the drop-down list above the form design surface. By default, the Name property is selected, but Figure 1-10 shows an example with the Text property.

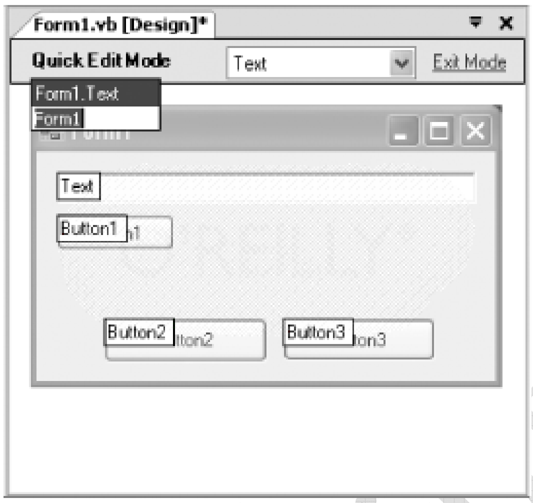


Figure 1-10. Editing a single property for multiple controls

In property-editing view, an edit box appears over every control on the form with the contents of the selected property. You can edit the value of that property by simply clicking on the edit box and entering the new value. You can also jump from one control to the next by pressing the Tab key.
When you are finished with your work, again select View → Property Editing View, or click the Exit Mode link next to the property drop-down list.

What about...

...editing tab order? Visual Studio allows you to easily edit tab order by clicking controls in the order that you want users to be able to navigate through them. Select a form with at least one control, and choose View → Tab Order to activate this mode, which works the same as it did in Visual Studio 2003.

1.8. Call Methods at Design Time

Although Visual Studio .NET 2003 included the Immediate window, you couldn't use it to execute code at design time. Longtime VB coders missed this feature, which was a casualty of the lack of a background compiler. In Visual Studio 2005, this feature returns along with the return of a background compiler.

Need to try out a freshly written code routine? Visual Studio 2005 lets you run it without starting your project.

How do I do that?

You can use the Immediate window to evaluate simple expressions, and even to run subroutines in your code. To try out this technique, add the following shared method to a class:

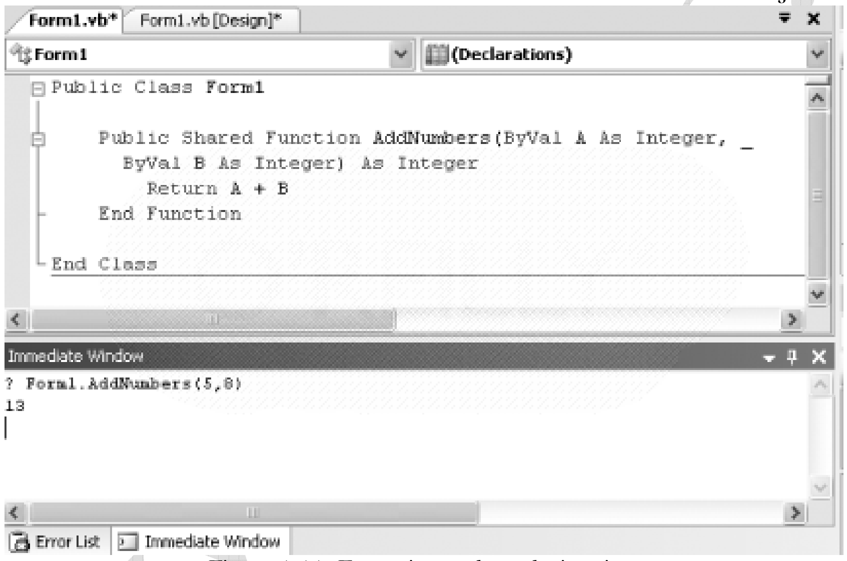
```
Public Shared Function AddNumbers(ByVal A As Integer, _
  ByVal B As Integer) As Integer

    Return A + B

End Sub
```

By making this a shared method, you ensure that it's available even without creating an instance of the class. Now, you can call it easily in the design environment.

By default, the Immediate window isn't shown at design time. To show it, select Debug → Windows → Command from the menu. Statements inside the Immediate window usually start with ? (a shorthand for Print, which instructs Visual Studio to display the result). You can enter the rest of the statement like any other line of code, with the benefit of IntelliSense. Figure 1-11 shows an example in which the Command window is used to run the shared method just shown.



```
Public Class Form1

    Public Shared Function AddNumbers(ByVal A As Integer, _
      ByVal B As Integer) As Integer
        Return A + B
    End Function

End Class
```

```
? Form1.AddNumbers(5,8)
13
```

Figure 1-11. Executing code at design time

When you execute a statement like the one shown in Figure 1-11, there will be a short pause while the background compiler works (and you'll see the message "Build started" in the status bar). Then the result will appear.

The expression evaluation in the beta release of Visual Studio 2005 is a little quirky. Some evaluations won't work, and will simply launch your application without returning a result. Look for this to improve in future builds.

1.9. Insert Boilerplate Code Using Snippets

Some code is common and generic enough that programmers everywhere write it again and again each day. Even though developers have the help of online documentation, samples, and books like the one you're reading, useful code never seems to be at your fingertips when you need it. Visual Studio 2005 includes a new code snippet feature that allows you to insert commonly used code and quickly adapt it to suit your purposes. Early beta builds of Visual Studio 2005 included a tool

for building your own snippets. Although this feature isn't in the latest releases, Microsoft has suggested that it might appear as a separate add-on tool at a later time.
Looking for the solution to an all-too-common nuisance? Visual Studio code snippets might already have the answer.

How do I do that?

You can insert a code snippet anywhere in your code. Just move to the appropriate location, right-click the mouse on the current line, and select Insert Snippet. A pop-up menu will appear with a list of snippet categories, such as Math, Connectivity and Networking, and Working with XML. Once you select a category, a menu will appear with a list of snippets. Once you select a snippet, the code will be inserted.

For example, suppose you want to add the ability to send and receive email messages to your application. Just create a new event handler or a standalone method, and right-click inside it. Then, choose Insert Snippet, and select Connectivity and Networking → Create an Email Message. Figure 1-12 shows the code that's inserted.

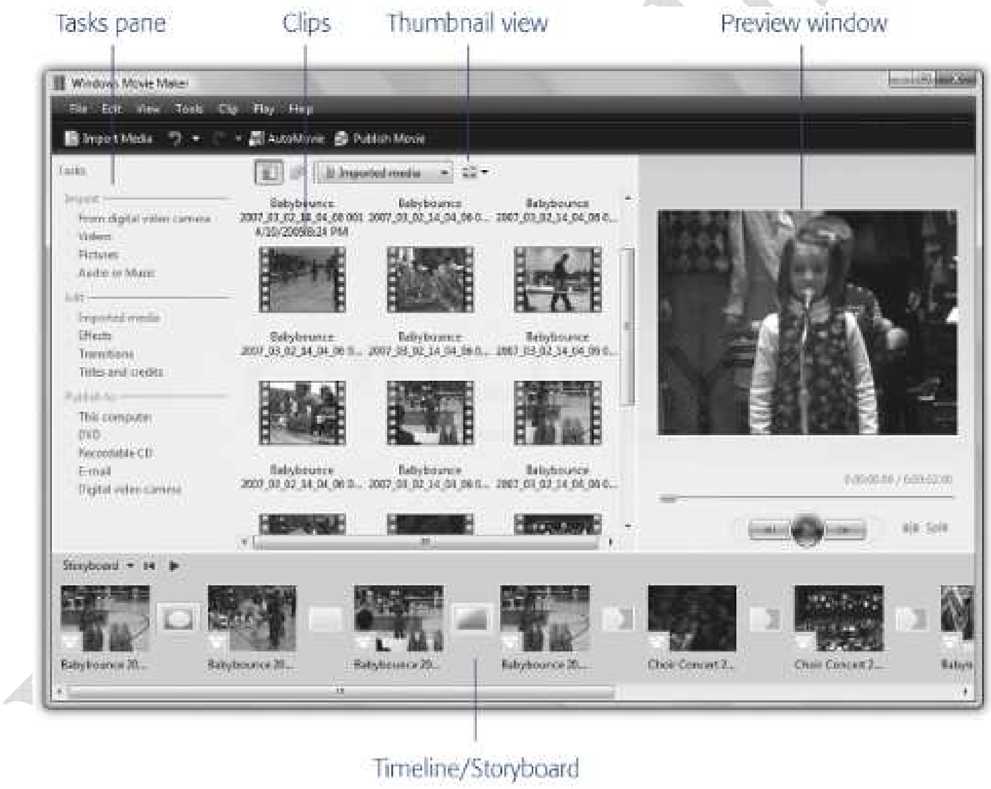


Figure 1-12. Customizing a code snippet

The shaded portions of code are literal values (like file paths and control references) that you need to customize to adapt the code to your needs. By pressing the Tab key, you can move from one shaded region to the next. Additionally, if you hover over a shaded region, a ToolTip will appear with a description of what content you need to insert.

What about...

...getting more snippets? The basic set of code snippets included with Visual Studio .NET is fairly modest. It includes some truly useful snippets (e.g., "Find a Node in XML Data") and some absurdly trivial ones (e.g., "Add a Comment to Your Code"). However, many useful topics, such as encryption, aren't dealt with at all.

Where can I learn more?

Thanks to the pluggable nature of snippets, you may soon be able to add more snippets to your collection from community web sites, coworkers, third-party software developers, and even sample code from a book like this.

1.10. Create XML Documentation for Your Code

Properly commenting and documenting code takes time. Unfortunately, there's no easy way to leverage the descriptive comments you place in your code when it comes time to produce more detailed API references and documentation. Instead, you typically must create these documents from scratch.

Use XML comments to effortlessly create detailed code references,a feature C# programmers have had since . NET 1.0.

Visual Studio 2005 changes all this by introducing a feature that's been taken for granted by C# programmers since .NET 1.0—XML comments. With XML comments, you comment your code using a predefined format. Then, you can use other tools to extract these comments and use them to build other documents. These documents can range from help documentation to specialized code reports (for example, a list of unresolved issues, legacy code, or code review dates).

How do I do that?

XML comments are distinguished from ordinary comments by their format. First of all, XML comments start with three apostrophes (rather than just one). Here's an example:
''' <summary>This is the summary.</summary>

As you can see, XML comments also have another characteristic—they use tag names. The tag identifies the type of comment. These tags allow you to distinguish between summary information, information about a specific method, references to other documentation sections, and so on.
The most commonly used XML comment tags include:

- ✓ <summary> - Describes a class or another type. This is the highest-level information for your code.
- ✓ <remarks> - Allows you to supplement the summary information. This tag is most commonly used to give a high-level description of each type member (e.g., individual methods and properties).
- ✓ <param> - Describes the parameters accepted by a method. Add one <param> tag for each parameter.
- ✓ <returns> - Describes the return value of a method.
- ✓ <exception> - Allows you to specify which exceptions a class can throw.
- ✓ <example> - Lets you specify an example of how to use a method or other member.

- ✓ <see> - Allows you to create a link to another documentation element.

In addition, there are tags that are usually used just to format or structure blocks of text. You use these tags inside the other tags. They include:

- ✓ <para> - Lets you add structure to a tag (such as a <remarks> tag) by separating its content into paragraphs.
- ✓ <list> - Starts a bulleted list. You must tag each individual list item with the <item> tag.
- ✓ <c> - Indicates that text within a description should be marked as code. Use the <code> tag to indicate multiple lines as code.
- ✓ <code> - Allows you to embed multiple lines of code, as in an example of usage. For example, you would commonly put a <code> tag inside an <example> tag.

In addition, you can define custom tags that you can then use for your own purposes. Visual Studio helps you out by automatically adding some XML tags—but only when you want them. For example, consider the code routine shown here, which tests if two files are exactly the same using a hash code. In order to use this sample as written, you need to import the System.IO and System.Security.Cryptography namespaces:

```
Public Function TestIfTwoFilesMatch(ByVal fileA As String, _
  ByVal fileB As String) As Boolean

    ' Create the hashing object.
    Dim Hash As HashAlgorithm = HashAlgorithm.Create( )

    ' Calculate the hash for the first file.
    Dim fsA As New FileStream(fileA, FileMode.Open)
    Dim HashA( ) As Byte = Hash.ComputeHash(fsA)
    fsA.Close( )

    ' Calculate the hash for the second file.
    Dim fsB As New FileStream(fileB, FileMode.Open)
    Dim HashB( ) As Byte = Hash.ComputeHash(fsB)
    fsB.Close( )

    ' Compare the hashes.
    Return (Convert.ToString(HashA) = Convert.ToString(HashB))

End Function
```

Now, position your cursor just before the function declaration, and insert three apostrophes. Visual Studio will automatically add a skeleton set of XML tags, as shown here:

```
''' <summary>
'''
''' </summary>
''' <param name="fileA"></param>
''' <param name="fileB"></param>
''' <returns></returns>
''' <remarks></remarks>
Public Function TestIfTwoFilesMatch(ByVal fileA As String, _
  ByVal fileB As String) As Boolean
...
```

Now, you simply need to fill in some sample content within the tags:

```
''' <summary>
''' This function tests whether two files
''' contain the exact same content.
''' </summary>
''' <param name="fileA">Contains the full path to the first file.</param>
''' <param name="fileB">Contains the full path to the second file.</param>
''' <returns>True if the files match, false if they don't.</returns>
''' <remarks>
''' The implementation of this method uses cryptographic classes
''' to compute a hash value. This may not be the most performant
''' approach, but it is sensitive to the minutest differences,
''' and can't be practically fooled.
''' </remarks>
```

To make a more realistic example, put the TestIfTwoFilesMatch( ) method into a class, and add XML documentation tags to the class declaration. Here's a typical example, which uses cross-references that point to the available methods in a list:

```
''' <summary>
''' This class contains methods for comparing files.
''' </summary>
''' <remarks>
'''     <para>This class is stateless. However, it's not safe to use
''' it if the file in question may already be help open by another
''' process.</para>
'''     <para>The methods in this class include:</para>
'''     <list type="bullet">
'''        <item><see cref="FileComparer.TestIfTwoFilesMatch"/>
''' TestifTwoFilesMatch( ) uses hash codes to compare two files.</item>
'''     </list>
''' </remarks>
Public Class FileComparer
   ...
End Class
```

Unlike other comments, XML comments are added to the metadata of your compiled assembly. They'll automatically appear in the Object Browser when you examine a type.

Additionally, once you've created your XML tags, you can export them to an XML document. Just double-click the My Project node in the Solution Explorer, choose the Compile tab, and ensure that the "Generate XML documentation file" option is selected. The XML documentation is automatically saved into an XML file with the same name as your project and the extension .xml (in the bin directory).

The generated document will include all of the XML comments, with none of the code. You can then feed this document into some other type of application. For example, you might create your own custom application to scan XML comment files and build specialized reports.

What about...

...creating help documentation? Although Visual Studio 2005 doesn't include any tools of its own, the open source NDoc application provides a solution (http://ndoc.sourceforge.net). NDoc scans code and uses the XML documentation tags it finds to build MSDN-style web pages or Visual Studio-style (MS Help 2.0) documentation. At the time of this writing, NDoc doesn't yet support .NET 2.0.