19

Code Snippets

Code snippets are small chunks of code that can be inserted into an application's code base and then customized to meet the application's specific requirements. They are usually generic in nature and serve one specific purpose. Code snippets do not generate full-blown applications or whole form definitions — project and item templates are used for such purposes. Instead, code snippets shortcut the programming task by automating frequently used code structures or obscure program code blocks that are not easy to remember.

In this chapter you'll see how code snippets have matured in Visual Studio 2005 to be powerful tools that can improve coding efficiency enormously, particularly for programmers who perform repetitive tasks with similar behaviors.

Code Snippets Revealed

Code snippet functionality has been around in many forms for a long time, but Microsoft has not included it in their development environments natively until the release of Visual Studio 2005, preferring to let third parties create various add-ins for languages such as Visual Basic 6 and the early versions of Visual Studio .NET.

Visual Studio 2005 marks the introduction of a proper code snippet feature built directly into the IDE. It allows code snippets that include not only blocks of code, but also multiple sections of code to be inserted in different locations within the module. In addition, a type of variable can be defined that makes it clear to see what parts of the snippet are to be customized and what sections can be left as is.

Original Code Snippets

The original code snippets from previous versions of Visual Studio were simple at best. These snippets were used to store a block of plain text that could be then inserted into a code module when desired. The process to create and use them was simple as well: Select a section of code and

drag it over to the Toolbox. This creates an entry for it in the Toolbox with a default name equal to the first line of the code. You can rename it like any other element in the Toolbox, and to use it, simply drag the code to the desired location in the Code view and release the left mouse button (see Figure 19-1).

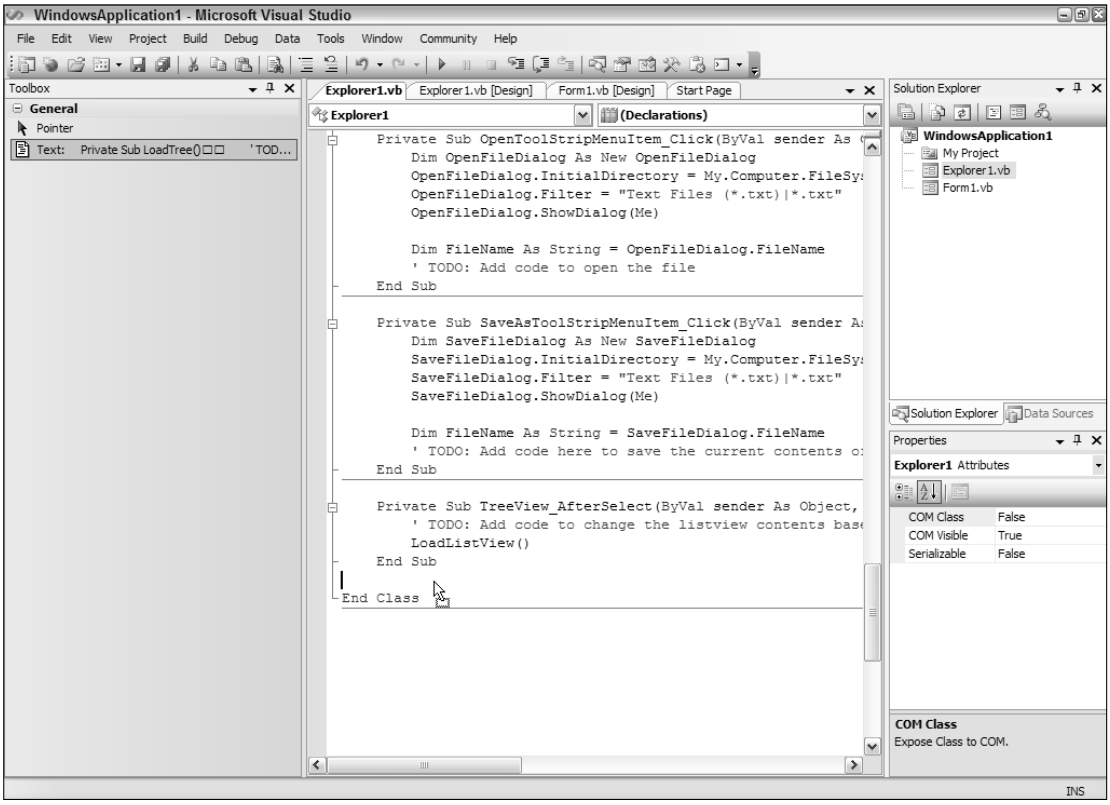

Figure 19-1

Many speakers used this simple technology to more easily display large code blocks in presentations, but in a real-world situation it was not as effective as it could have been, because often you had to remember to use multiple items to generate code that would compile.

It was also hard to share these so-called snippets, and equally hard to modify them. Nevertheless, this method of keeping small sections of code is still available to programmers in Visual Studio 2005, and it can prove useful when you don't need a permanent record of the code, but rather want to copy a series of code blocks for use immediately or in the near future.

"Real" Code Snippets

Now, in Visual Studio 2005, code snippet technology refers to something completely different. Code snippets are XML-based files containing sections of code that can include not only normal source code, but references and `Imports` statements and replaceable parameters as well.

Visual Studio 2005 ships with many predefined code snippets in the three main languages — Visual Basic, C#, and J#. These snippets are arranged hierarchically in a logical fashion so that you can easily locate the appropriate snippet. Rather than locate the snippet in the Toolbox, you can use menu commands or keyboard shortcuts to bring up the main list of groups.

New code snippets can be created to automate almost any coding task and then stored in this code snippet library. Because each snippet is stored in a special XML file, you can even share them with other developers.

Using Snippets in Visual Basic

Code snippets are a natural addition to the Visual Basic developer's toolset. They provide a shortcut way to insert code that either is difficult to remember or is used often with minor tweaks. One common problem some programmers have is remembering the correct references and `Imports` statements required to get a specific section of code working properly; code snippets in Visual Basic solve this problem by including all the necessary associations as well as the actual code.

To use a code snippet you should first locate where you want the generated code to be placed in the program listing and position the cursor at that point. You don't have to worry about the associated references and `Imports` statements, as they will be placed in the correct location.

There are three scopes under which a snippet can be inserted:

- **Class Declaration:** The snippet will actually include a class declaration, so it should not be inserted into an existing class definition.
- **Member Declaration:** This snippet scope will include code that defines members, such as functions and event handler routines. This means it should be inserted outside an existing member.
- **Member Body:** This scope is for snippets that are inserted into an already defined member, such as an event handler routine.

Once you've determined where the snippet is to be placed, the easiest way to bring up the Insert Snippet dialog is to use the keyboard shortcut chord of Ctrl+K, Ctrl+X (remember that a chord is a way of stringing multiple keyboard shortcuts together). There are two alternative methods to start the Insert Snippet process. The first is to right-click at the intended insertion point in the code window and select Insert Snippet from the context menu that is displayed. The other option is to use the Edit➪IntelliSense➪Insert Snippet menu command.

The Insert Snippet dialog is a special kind of IntelliSense (hence its location in the menu structure) that appears inline in the code window. Initially it displays the words Insert Snippet along with a drop-down list of code snippet groups from which to choose. Once you select the group that contains the snippet you require, it will show you the list of snippets from which you simply double-click the one you need.

Because you can organize the snippet library into many levels, you may find that the snippet you need is multiple levels deep in the Insert Snippet dialog. Figure 19-2 displays an Insert Snippet dialog in which the user has navigated through two levels of groups and then located a snippet named Draw a Pie Chart.

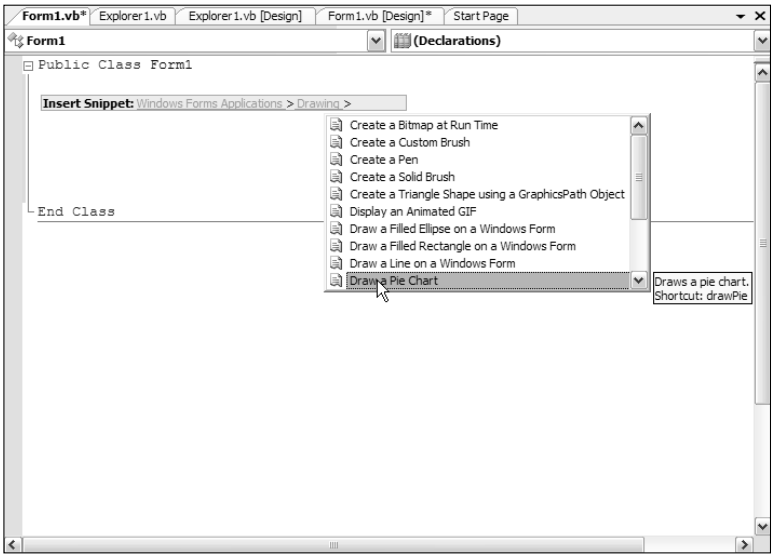

Figure 19-2

Double-clicking this entry will tell Visual Studio 2005 that you want the snippet to be generated at the current location. Figure 19-3 displays the result of selecting the Draw a Pie Chart snippet. This example shows a snippet with member declaration scope because it adds the definition of two subroutines to the code. To help you modify the code to your own requirements, the sections you would normally need to change are highlighted (the default is a green background).

When changing the variable sections of the generated code snippet, Visual Studio helps you even further. Select the first highlighted literal to be changed and enter the new value. Pressing the Tab key will move to the next literal and highlight it, ready for you to override the value with your own. Shift+Tab will navigate backward, so you have an easy way of accessing the sections of code that need changing without needing to manually select the next piece to modify.

Some code snippets use the same variable for multiple pieces of the code snippet logic. This means changing the value in one place will result in it changing in all other instances. A great example of this is can be found by selecting Windows Forms Applications➪Forms➪Add a Windows Forms Control At Run Time. The code that is generated through this snippet is shown in Figure 19-4, with all occurrences of `MyTest` referring to the same variable. Changing the first instance of `MyTest` in the line `Dim MyTest As New TextBox()` will result in the other two instances also changing automatically.

You might have noticed in Figure 19-2 that the tooltip text includes the words "Shortcut: drawPie." This text indicates that the selected code snippet has a text shortcut that you can use to automatically invoke the code snippet behavior without bringing up the IntelliSense dialog.

```
Public Class Form1

    ' Shows how to call the DrawPieChart method
    Public Sub DrawPieChartHelper()
        Dim percents() As Integer = {10, 20, 70}
        Dim colors() As Color = {Color.Red, Color.CadetBlue, Color.Khaki}
        Dim graphics As Graphics = Me.CreateGraphics
        Dim location As Point = New Point(10, 10)
        Dim size As Size = New Size(150, 150)
        DrawPieChart(percents, colors, graphics, location, size)
    End Sub


    ' Draws a pie chart.
    Public Sub DrawPieChart(ByVal percents() As Integer, ByVal colors() As Color, _
    ByVal surface As Graphics, ByVal location As Point, ByVal pieSize As Size)
        ' Check if sections add up to 100.
        Dim sum As Integer = 0
        For Each percent As Integer In percents
            sum += percent
        Next

        If sum <> 100 Then
            Throw New ArgumentException("Percentages do not add up to 100.")
        End If

        If percents.Length <> colors.Length Then
            Throw New ArgumentException("There must be the same number of percents and colors.")
        End If

        Dim percentTotal As Integer = 0
        For percent As Integer = 0 To percents.Length() - 1
            surface.FillPie( _
                New SolidBrush(colors(percent)), _
```

Figure 19-3

Of course, you need to know what the shortcut is before you can use this feature, but for those that you are aware of, all you need to do is type the shortcut into the code editor and press the Tab key. In Visual Basic the shortcut isn't even case sensitive, so this example can be generated by typing the term "draw-pie" and pressing Tab.

```
Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load
        Dim MyTest As New TextBox()
        With MyTest
            .Location = New Point(64, 40)
            .Size = New Size(100, 20)
            .TabIndex = 0
            .Text = "TextBox1"
        End With
        Controls.Add(MyTest)

    End Sub
End Class
```

Figure 19-4

Note that in some instances the IntelliSense engine may not recognize this kind of shortcut. If this happens to you, press Ctrl+Tab to force the IntelliSense to intercept the Tab key.

Using Snippets in C# and J#

The code snippets in C# and J# are not as extensive as those available for Visual Basic but are inserted in the same way. Only Visual Basic supports the advanced features of the code snippet functionality, such as references and `Imports` statements. First, locate the position where you want to insert the generated code and then use one of the following methods:

- The keyboard chord Ctrl+K, Ctrl+X
- Right-click and choose Insert Snippet from the context menu
- Run the Edit➪IntelliSense➪Insert Snippet menu command

At this point, Visual Studio will bring up the Insert Snippet list for the current language, as Figure 19-5 shows. As you scroll through the list and hover the mouse pointer over each entry, a tooltip will be displayed to indicate what the snippet does.

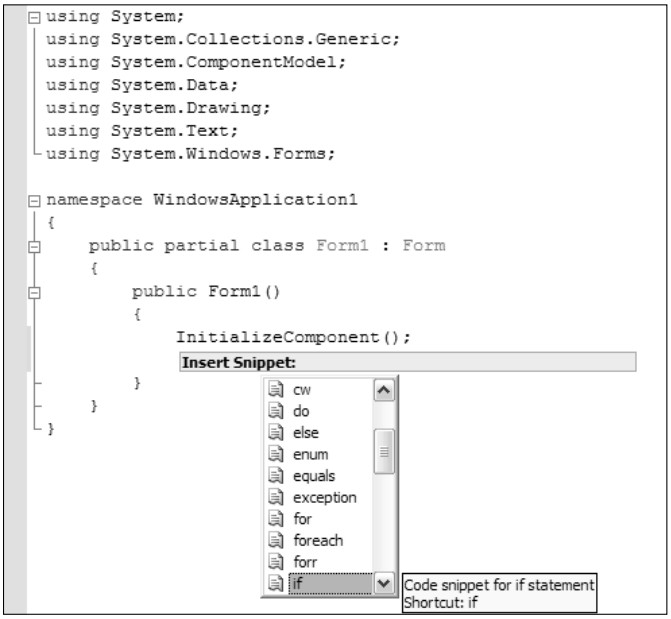

Figure 19-5

Although the predefined C# and J# snippets are limited in nature, you can create more functional and complex snippets for them.

Creating Snippets Manually

Visual Studio 2005 does not ship with a code snippet creator or editor. During the development of the IDE, Microsoft determined that a third-party tool, simply called the Snippet Editor, performed this functionality well enough that there was no reason to include a built-in editor in the IDE. Later in this chapter you'll learn how to use the Snippet Editor to create your own snippets, but it's worth taking a look at how code snippets are structured by looking at the manual method of creating one.

Each code snippet is simply an individual XML file with a file extension of `.snippet`. The contents of the file are written in plain text and follow the standard XML structure of a hierarchy of tags containing attributes and values. The remainder of this section deals with the structure of the code snippet XML schema.

Every snippet file must start with the `CodeSnippets` tag, identifying the namespace that defines the code snippet schema. This is written in the following form:

```
<CodeSnippets
    xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
</CodeSnippets>
```

Within these tags, each snippet is defined using the `CodeSnippet` tag, which will in turn contain the definition of the snippet itself:

```
  <CodeSnippet Format="1.0.0">
  </CodeSnippet>
```

Similar to HTML files, each code snippet has a header area and a body area, known as the `Header` and `Snippet`, respectively. The `Header` area can contain any combination of three separate tags, each defining a different attribute of the snippet:

- `Title`: The name of the snippet
- `Description`: The description of the snippet
- `Shortcut`: A shortcut term used to insert the snippet automatically

The `Header` layout looks like the following:

```
    <Header>
      <Title>The Name Of The Snippet</Title>
      <Description>The description of the snippet. (Optional)</Description>
      <Shortcut>The shortcut for the snippet. (Optional)</Shortcut>
    </Header>
```

Within the main `Snippet` tag you need to define the actual code to be inserted into the module. A `Code` tag is included with an attribute of `Language` (containing `VB`, `C#`, or `J#` depending on the language for which the snippet is intended). The actual code needs to be defined within a custom data tag with the format `<![CDATA[ code ]]>`. For example, the most basic `Snippet` tag looks like this:

```
<Snippet>
  <Code Language="VB">
    <![CDATA[The Code Goes Here]]>
  </Code>
</Snippet>
```

In addition to this code, you can define references and `Imports` statements in Visual Basic code snippets. Rather than insert the code at the selected entry point, Visual Studio will associate the references correctly as well as place the `Imports` statements at the top of the code module. These are placed at the top of the `Snippet` tag before the `Code` tag:

```
<Snippet>
<References>
   <Reference>
     <Assembly>AssemblyName.dll</Assembly>
   </Reference>
 </References>
 <Imports>
   <Import>
     <Namespace>Namespace.Name</Namespace>
   </Import>
 </Imports>
 <Code Language="VB">
   <![CDATA[The Code Goes Here]]>
 </Code>
</Snippet>
```

As shown in the preceding example, code snippets can also have variable sections marked with special aliases so that the developer using the snippet knows which bits he or she should customize. To include such an alias, you first need to define it using a `Literal` tag. The `Literal` tag structure consists of the following:

- `ID`: An `ID` tag to uniquely identify the variable
- `Type`: The type of data to be inserted in this variable. This is optional.
- `ToolTip`: If defined, the user will see a tooltip containing this text. This is optional.
- `Default`: A default value to be placed in the automatically generated code. This is optional.

Following is a sample `Literal` tag:

```
<Literal>
  <ID>MyID</ID>
  <Type>String</Type>
  <ToolTip>The tooltip text</ToolTip>
  <Default>MyVarName</Default>
</Literal>
```

Object variables can also be included in the same way as literals, but use the `Object` *tab instead.*

To use `Object` and `Literal` aliases in the code to be inserted, enclose the ID of the required variable with dollar signs ($) and include it at the intended location in the code. The following code includes references to a literal and an object called `controlName` and `controlType`, respectively:

```
        <Code Language="VB">
          <![CDATA[
              Dim $controlName$ As $controlType$
          ]]>
        </Code>
```

You can use the same variable multiple times in the code. When you change the value after the code is generated, the code snippet IntelliSense engine will automatically update any other occurrences of the `Literal` or `Object` with the new value.

The final code snippet structure appears like this:

```
<CodeSnippets
    xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <CodeSnippet Format="1.0.0">
    <Header>
      <Title>The Name Of The Snippet</Title>
      <Description>The description of the snippet. (Optional)</Description>
      <Shortcut>The shortcut for the snippet. (Optional)</Shortcut>
    </Header>
     <Snippet>
     <References>
        <Reference>
          <Assembly>AssemblyName.dll</Assembly>
        </Reference>
      </References>
      <Imports>
        <Import>
          <Namespace>Namespace.Name</Namespace>
        </Import>
      </Imports>
        <Literal>
          <ID>MyID</ID>
          <Type>String</Type>
          <ToolTip>The tooltip text</ToolTip>
          <Default>MyVarName</Default>
        </Literal>
        <Object>
          <ID>MyType</ID>
          <Type>Control</Type>
          <ToolTip>The tooltip text</ToolTip>
          <Default>Button</Default>
        </Object>
      <Code Language="VB">
        <![CDATA[
            Dim $myID$ As $MyType$
        ]]>
      </Code>
    </Snippet>
  </CodeSnippet>
</CodeSnippets>
```

The best way to illustrate how code snippets can make your life easier is to walk through the creation of a simple example, adding it to the code snippets library and then using it in code. This next exercise

does just that, creating a snippet that in turn creates three subroutines, including a helper subroutine that is intended to show the developer using the snippet how to call the functionality properly:

1. Start Notepad and add the following stub of XML (you're using Notepad to show that code snippets are simply XML written in plain text):

```
<?xml version="1.0"?>
<CodeSnippets xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <CodeSnippet Format="1.0.0">
  </CodeSnippet>
</CodeSnippets>
```

2. The first task is to define the header information. This is what's used to define the name of the snippet in the snippet library, and it also enables you to define a shortcut and a brief description of what the code snippet does. In between the `CodeSnippet` tags, insert the XML to create a `Header` tag that contains `Title`, `Description`, and `Shortcut` tags, like so:

```
<?xml version="1.0"?>
<CodeSnippets xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <CodeSnippet Format="1.0.0">
    <Header>
      <Title>CreateAButtonSample</Title>
      <Description>This snippet adds code to create a button control and
          hook an event handler to it.</Description>
      <Shortcut>createAButton</Shortcut>
    </Header>
  </CodeSnippet>
</CodeSnippets>
```

3. Now that the header information is present, you can begin creating the snippet itself. Start by defining the `Snippet` tag with a `Declaration` section and the main `Code` tag, setting attributes to `VB` (for Visual Basic) and `method decl` for the `Kind` so that Visual Studio knows that the scope of this snippet is a member declaration:

```
<?xml version="1.0"?>
<CodeSnippets xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <CodeSnippet Format="1.0.0">
    <Header>
      <Title>CreateAButtonSample</Title>
      <Description>This snippet adds code to create a button control and
          hook an event handler to it.</Description>
      <Shortcut>createAButton</Shortcut>
    </Header>
    <Snippet>
      <Declarations>
      </Declarations>
      <Code Language="VB" Kind="method decl">
      </Code>
    </Snippet>
  </CodeSnippet>
</CodeSnippets>
```

4. Define the `Literal` tags for the `Name` and `Text` properties that will be used to customize the button's creation. These properties will be used in the `Helper` subroutine so you know what you need to change to make the other subroutines work. `Literal` tags need an `ID` to identify

the alias used in the code snippet; and they can have a default value as well as an explanatory tooltip. You'll use all three tags to create your `Literal` tags, which should be included in the `Declarations` section:

```
  <Declarations>
    <Literal>
      <ID>controlName</ID>
      <ToolTip>The name of the button.</ToolTip>
      <Default>"MyButton"</Default>
    </Literal>
    <Literal>
      <ID>controlText</ID>
      <ToolTip>The Text property of the button.</ToolTip>
      <Default>"Click Me!"</Default>
    </Literal>
  </Declarations>
```

5. As mentioned earlier, the code to be inserted when this snippet is activated needs to be inserted in a custom data tag in the following form:

```
<![CDATA[code goes here]]>
```

Type the following code in between the opening and closing `Code` tags. It defines the three subroutines and is straight Visual Basic code other than the use of the aliased `Literal` tags. Note that these are enclosed by dollar signs ($) to tell Visual Studio that they are aliases — to use the `Literal controlName`, the alias `$controlName$` is used:

```
<Code Language="VB" Kind="method decl">
<![CDATA[Private Sub CreateButtonHelper
    CreateAButton($controlName$, $controlText$, Me)
End Sub

Private Sub CreateAButton(ButtonName As String, ButtonText As String, _
    Owner As Form)
    Dim MyButton As New Button

    MyButton.Name = ButtonName
    MyButton.Text = ButtonName
    Owner.Controls.Add(MyButton)

    MyButton.Top = 0
    MyButton.Left = 0
    MyButton.Text = ButtonText
    MyButton.Visible = True

    AddHandler MyButton.Click, AddressOf ButtonClickHandler
End Sub

Private Sub ButtonClickHandler(ByVal sender As System.Object, _
    ByVal e As System.EventArgs)
    MessageBox.Show("The " & sender.Name & " button was clicked")
End Sub
]]>
      </Code>
```

6. Save the file as `CreateAButton.snippet` somewhere where you can locate it easily and switch to Visual Studio 2005. Bring up the code snippets library with the keyboard shortcut chord Ctrl+K, Ctrl+B. Once the library is displayed, click the Import button and browse to the snippet file you just saved.
7. Choose a suitable location for the snippet — the My Snippets group is the usual place for custom-built snippets — and click Finish. Click OK to close the library. Your snippet is now saved and stored in Visual Studio 2005, ready for use.
8. To test that the code snippet was properly defined and installed, create a new Windows Forms application and switch to the Code view of `Form1`. Display the Code Snippet IntelliSense dialog by using the keyboard chord Ctrl+K, Ctrl+X, and then browse to the CreateAButton snippet you just imported and double-click it. Visual Studio should insert the Visual Basic code to define three subroutines, with two variables highlighted.
9. Add the following code to the bottom of the `Form1` class definition:

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

This will execute the `CreateButtonHelper` subroutine when the form is first loaded, which in turn will call the other subroutines generated by the code snippet and create a button with default text and a default behavior. Run the application and click the button that is created, and you should get similar results to those shown in Figure 19-6.

Figure 19-6

While this sample shows the creation of a simple code snippet, you can use the same technique to create complex snippets that include `Imports` statements, code definitions, and markup for sections within the code snippet text to be replaced by the developer using it.

Code Snippets Manager

The Code Snippets Manager is the central library for the code snippets known to Visual Studio 2005. You can access it via the Tools➪Code Snippet Manager menu command or the keyboard shortcut chord, Ctrl+K, Ctrl+B.

When it is initially displayed, the Code Snippets Manager will show the snippets for the language you're currently using. Figure 19-7 shows how it will look when you're editing a Visual Basic project. The hierarchical folder structure follows the same set of folders on the PC by default, but as you add snippet files from different locations and insert them into the different groups, the new snippets slip into the appropriate folders.

If you have an entire folder of snippets to add to the library, such as when you have a corporate setup and need to import the company-developed snippets, you use the Add button. This brings up a dialog that you use to browse to the required folder. Folders added in this fashion will appear at the root level of the tree view — on the same level as the main groups of default snippets. However, you can add a folder that contains subfolders, which will be added as child nodes in the tree view.

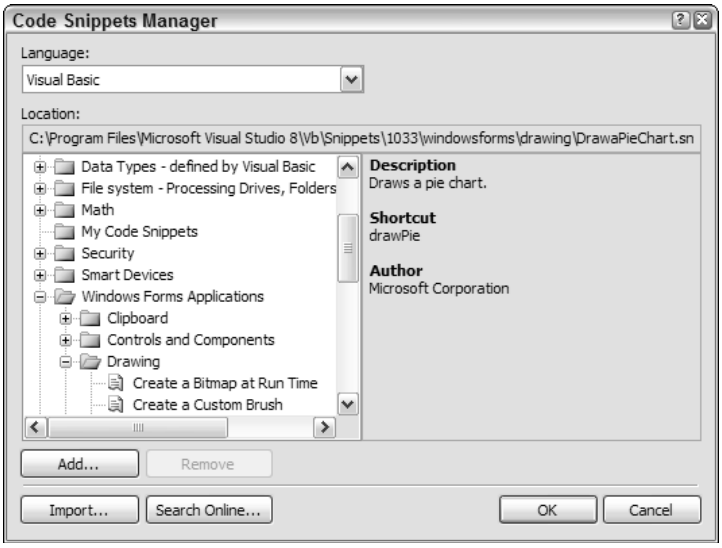

Figure 19-7

Removing a folder is just as easy — in fact, it's dangerously easy. Select the root node that you want to remove and click the Remove button. Instantly the node and all child nodes and snippets will be removed from the Snippets Manager without a confirmation window. You can add them back by following the steps explained in the previous walkthrough, but it can be frustrating trying to locate a default snippet folder that you inadvertently deleted from the list.

The location for the code snippets that are installed with Visual Studio 2005 is deep within the installation folder. By default, the code snippet library will be installed in `C:\Program Files\Microsoft Visual Studio 8\VB\Snippets\1033`.

Individual snippet files can be imported into the library using the Import button. The advantage of this method over the Add button is that you get the opportunity to specify the location of each snippet in the library structure.

Figure 19-8 shows the Import Code Snippet dialog for a sample snippet file `HelloPersonName.snippet`. By default, Visual Studio 2005 suggests that snippets added in this fashion be inserted into the custom My Code Snippets folder, but you can put the snippet in any folder that seems appropriate by finding it in the Location list.

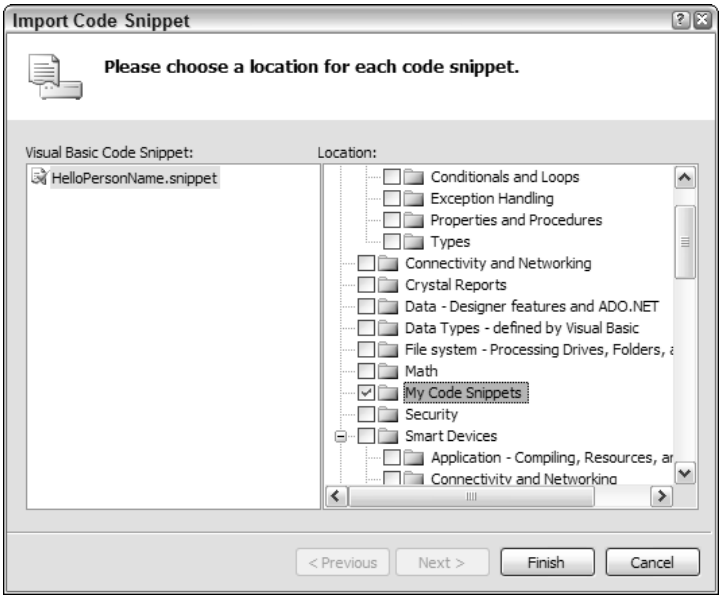

Figure 19-8

Creating Snippets with VB Snippet Editor

Creating code snippets by manually editing XML files can be tedious. It can also result in errors that are hard to track down. Fortunately, a third-party tool called Snippet Editor can make your life a lot easier. You'll find the Snippet Editor at `http://msdn.microsoft.com/vbasic/downloads/tools/snippeteditor/`. Download it and install it in a location that you can locate easily, as it doesn't create an entry in the Start menu. The default location is `C:\Documents and Settings\username\My Documents\MSDN\Code Snippet Editor`.

You may also want to create a desktop shortcut to the program if you'll be using it frequently.

When you start the Snippet Editor, it will display a welcome screen showing you how to browse and create new snippets. The left side of the screen is populated with a tree view containing all the Visual Basic snippets defined in your system known to Visual Studio 2005. Initially the tree view is collapsed, but by expanding it you'll see a set of folders similar to those in the code snippet library (see Figure 19-9).

Reviewing Existing Snippets

An excellent feature of the Snippet Editor is the view it offers of the structure of any snippet file in the system. This means you can browse the default snippets installed with Visual Studio, which can provide insight into how to better build your own snippets.

Browse to the snippet you're interested in and double-click its entry to display it in the Editor window. Figure 19-9 shows a simple Hello World snippet. You'll notice two tabs at the top of the editing side of the form — Editor and Preview. Editor is where you'll do most of your work, while switching over to Preview shows how the snippet will look when you insert it into your application code.

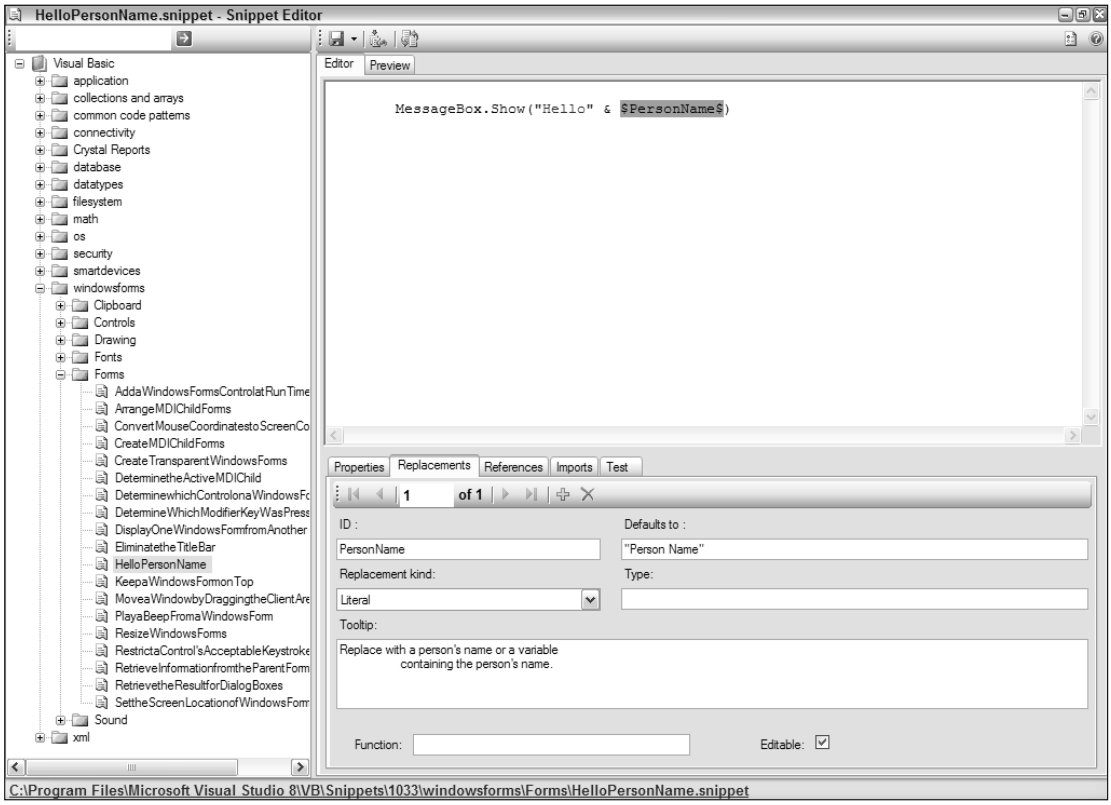

Figure 19-9

The lower area of the Editor pane contains all of the associated information about the snippet. From left to right, these tabs contain the following:

Tab	Function
Properties	The main properties for the snippet, including title, shortcut, and description
Replacements	All `Literal` and `Object` aliases are defined in this tab.
References	If your snippet will require system framework references, this tab allows you to define them.
Imports	Similar to the References tab, this tab enables you to define any `Imports` statements that are required in order for your snippet to function correctly.
Test	This tab attempts to analyze your snippet to confirm that it will work properly as is.

Browsing through these tabs enables you to analyze an existing snippet for its properties and replacement variables. In the example shown in Figure 19-9, the Replacements tab is displayed, showing that one replacement is defined as a `Literal` with an ID of `PersonName` and a default value of `"Person Name"`.

Be aware that the results shown in the Test tab are not always accurate. As shown in Figure 19-10, even the predefined snippet templates produce compilation errors when tested. This is because the Snippet Editor is not aware of the full context for which this snippet is intended. However, it's still a handy step to perform, as it will show you real errors in your code snippet as well.

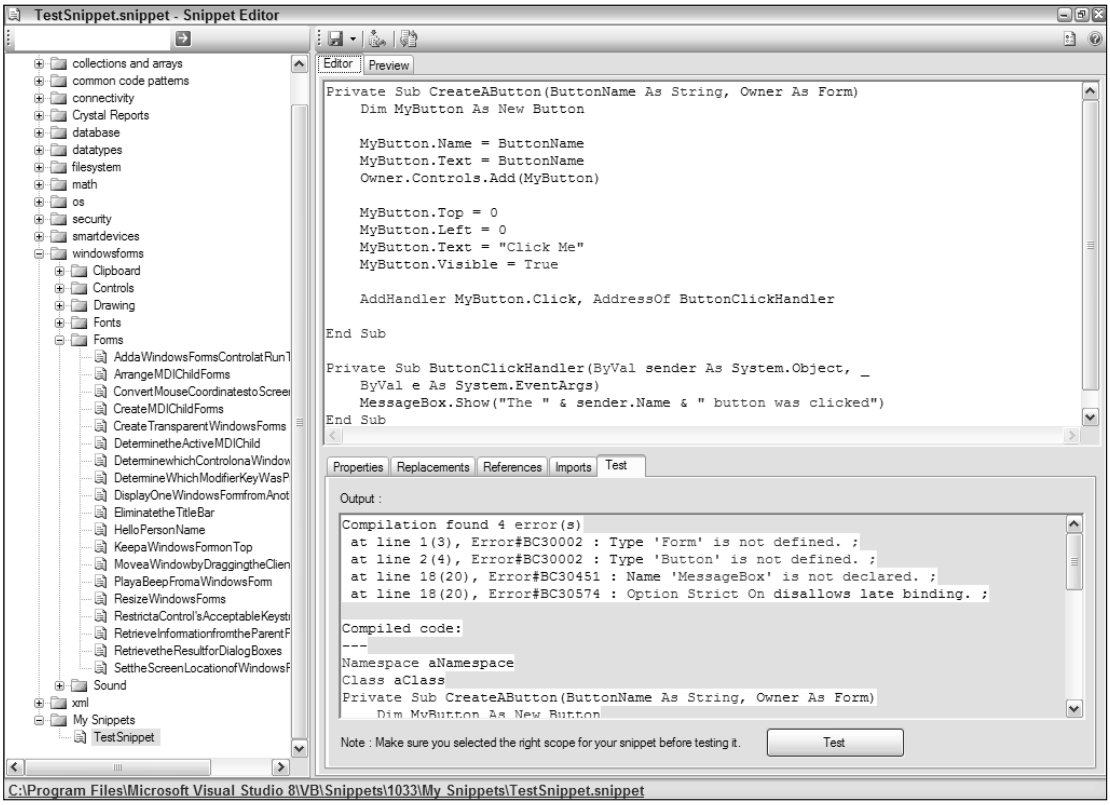

Figure 19-10

Both Figure 19-9 and Figure 19-10 show that editing snippets using the Snippet Editor is a much more pleasant process than editing the raw XML. The code to be inserted in the snippet is color-coded and formatted in a similar fashion to the Visual Basic editor in Visual Studio, giving you a familiar environment in which to write.

Replacements Explained

When defining `Literal` and `Object` aliases, you would normally define them in the XML using `Literal` and `Object` tags, and then refer to them in the code with special alias formatting. The Snippet Editor operates on a similar paradigm; use the Replacements tab to first define the replacement's properties. When the Add button is clicked in the Replacements tab, it will insert the default ID into the Editor window and populate the properties in the lower half.

You need to change the ID in the lower section, not in the Editor window.

To demonstrate how the Snippet Editor makes creating your own snippets a lot more straightforward, follow this next exercise in which you will create the same snippet you created earlier in this chapter, but this time taking advantage of the Snippet Editor's features:

1. Start the Snippet Editor and create a new snippet. To do this, locate the My Snippets folder in the tree view (or any other folder of your choice), right-click, and select Add New Snippet from the context menu that is displayed.
2. When prompted, name the snippet CreateAButtonSample2 and click OK. Double-click the new entry to open it in the Editor pane.

Note that creating the snippet will not automatically open the new snippet in the Editor — don't overwrite the properties of another snippet by mistake!

3. The first thing you need to do is edit the Title, Description and Shortcut fields so they match the previous sample (see Figure 19-11):
   - `Title`: CreateAButtonSample2
   - `Description`: This snippet adds code to create a button control and hook an event handler to it.
   - `Shortcut`: createAButton

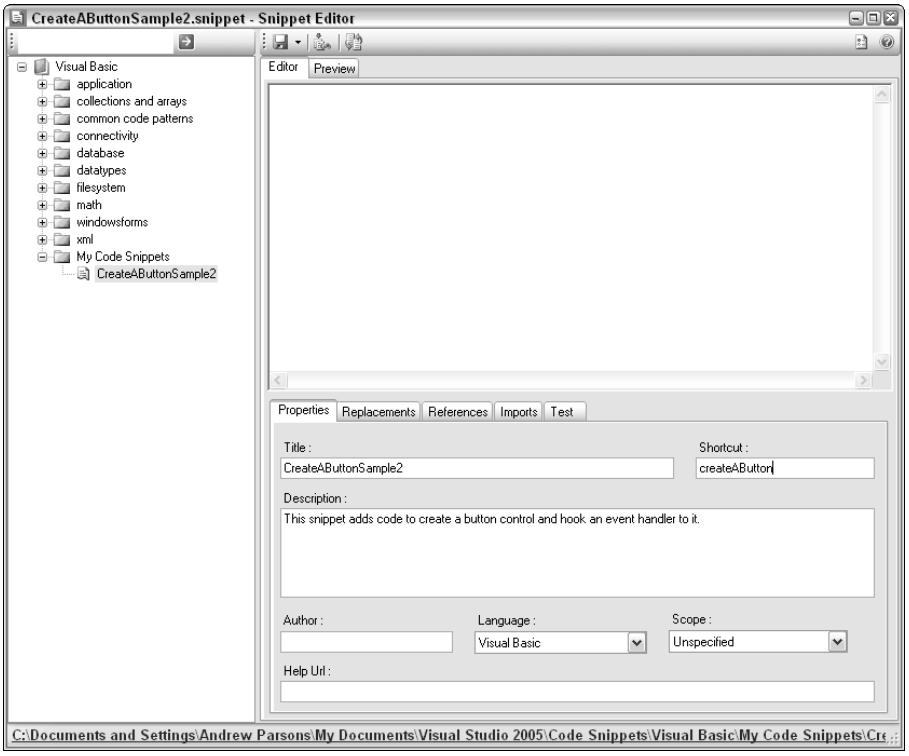

Figure 19-11

4. Because this snippet contains member definitions, set the Scope to `Member declaration`.
5. In the Editor window, insert the code necessary to create the three subroutines as before. Note that you don't have to include the custom data tag `CDATA`, as the Snippet Editor will do that for you in the background:

```
Private Sub CreateButtonHelper
    CreateAButton(, , Me)
End Sub

Private Sub CreateAButton(ButtonName As String, ButtonText As String, Owner As
Form)
    Dim MyButton As New Button

    MyButton.Name = ButtonName
    MyButton.Text = ButtonName
    Owner.Controls.Add(MyButton)

    MyButton.Top = 0
    MyButton.Left = 0
    MyButton.Text = ButtonText
    MyButton.Visible = True

    AddHandler MyButton.Click, AddressOf ButtonClickHandler
End Sub

Private Sub ButtonClickHandler(ByVal sender As System.Object, _
    ByVal e As System.EventArgs)
    MessageBox.Show("The " & sender.Name & " button was clicked")
End Sub
```

6. You'll notice that the call to CreateAButton is incomplete, because you haven't defined the `Literal` aliases yet, so switch over to the Replacements tab. Position the cursor immediately after the opening parenthesis on the CreateAButton function call and click the Add button to create a new replacement.

The Snippet Editor will immediately insert the default name for the new replacement in the code, but don't worry: It will be changed when you set the ID.

7. Change the replacement properties like so (note that the default values should include the quotes (") so they are generated in the snippet:
   - `ID`: `controlName`
   - `Defaults to`: `"MyButton"`
   - `Tooltip`: The name of the button
8. Notice that the code window changed the alias to the new ID. Position the cursor after the first comma and repeat the process of creating a new replacement. Set the properties of the new replacement as follows:
   - `ID`: `controlText`
   - `Defaults to`: `"Click Me!"`
   - `Tooltip`: The text property of the button

Your snippet is now done and ready to be used (compare it to Figure 19-12). You can use the Preview tab to check it against the code generated by the previous code snippet exercise or use Visual Studio 2005 to insert the snippet into a code window.

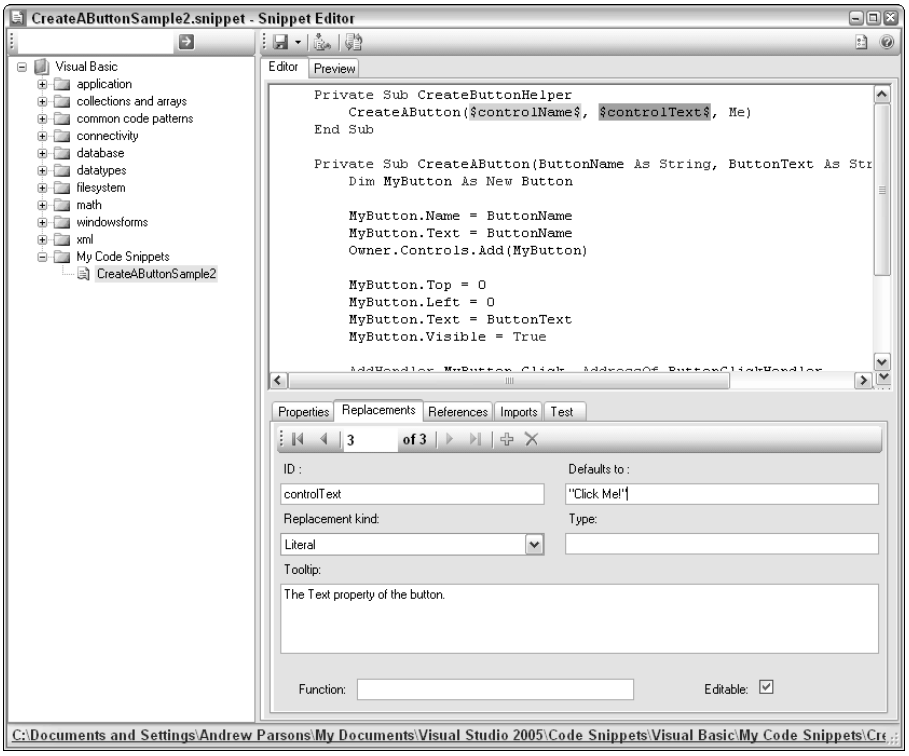

Figure 19-12

Note that if you added your snippet to a known folder, Visual Studio 2005 will automatically find it and recognize its shortcut without you needing to import it manually.

Summary

Code snippets are a valuable inclusion in the Visual Studio 2005 feature set. You learned in this chapter how to use them, and, more important, how to create your own, including variable substitution and `Imports` and reference associations for Visual Basic snippets. With this information you'll be able to create your own library of code snippets from functionality that you use frequently, saving you time in coding similar constructs later.

Contents

Professional Visual Studio® 2005

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

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
006.7'86—dc22
2006014685

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.