TRAINING & REFERENCE

murach's ASP.NET 2.0 web programming with VB 2005

(Chapter 2)

MIKE MURACH & ASSOCIATES, INC.
1-800-221-5528 • (559) 440-9071 • Fax: (559) 440-0963
murachbooks@murach.com • **www.murach.com**

2

How to develop a one-page web application

In the last chapter, you were introduced to the basic concepts of web programming and ASP.NET. Now, this chapter shows you how to develop a one-page web application using Visual Studio 2005. If you've used Visual Studio to develop Windows applications, you'll soon see that you develop web applications in much the same way. As a result, you should be able to move quickly through this chapter.

How to work with ASP.NET web sites 30
How to start a new web site 30
How to work with the Visual Studio IDE 32
How to add folders and files to a web site 34
How to open or close an existing web site 36
How to use Design view to build a web form 38
The design of the Future Value form 38
How to use flow layout 40
How to add a table to a form 42
How to add text to the cells of a table 42
How to add server controls to a form 44
How to set the properties of the controls 44
Common properties for web server controls 46
How to work in Source view 48
How to use Source view to modify the design 48
The aspx code for the Future Value form 50
How to add validation controls to a form 52
An introduction to the validation controls 52
How to use the required field validator 54
How to use the range validator 54
How to add code to a form 56
How to use the Code Editor 56
How to use page and control events 58
The Visual Basic code for the Future Value form 60
How to test a web application 62
How to run a web site with the built-in development server 62
How to review the HTML that's sent to the browser 64
Perspective 66

How to work with ASP.NET web sites

This chapter starts by showing you how to start a new web application, how to work with the Visual Studio IDE, how to add folders and files to an application, and how to close and re-open an application. Once you're comfortable with those skills, you'll be ready to learn how to build your first ASP.NET application.

How to start a new web site

In Visual Studio 2005, a web application is called a *web site*, and figure 2-1 shows the dialog box for starting a new web site. After you open the New Web Site dialog box, you select the language you want to use for the web site and you specify the location where the web site will be created.

The Location drop-down list gives you three options for specifying the location of the web site. The simplest method is to create a *file-system web site*. This type of web site can exist in any folder on your local hard disk, or in a folder on a shared network drive. You can run a file-system web site using either Visual Studio's built-in development server or IIS. You'll learn how to do that later in this chapter.

You use the HTTP option to create a web site that runs under IIS on your local computer or on a computer that can be accessed over a local area network. To use this option, you must specify the IIS server where you want to create the web site. In addition, you must select or create the IIS directory that will contain the files for the web site, or you must select or create a virtual directory for the web site.

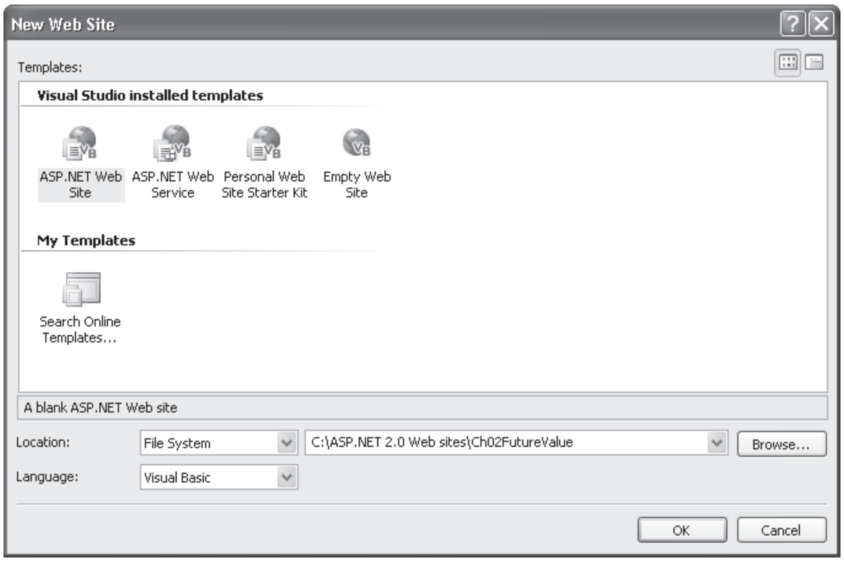
The third option, FTP, lets you create a web site on a remote server by uploading it to that server using FTP. To create this type of web site, you must specify at least the name of the FTP server and the folder where the web site resides. You'll learn more about how to use the HTTP and FTP options in chapter 4.

By default, Visual Studio 2005 creates a solution file for your web site in My Documents\Visual Studio 2005\Projects. This solution file is stored in this folder regardless of the location of the web site itself. To change the location where solutions are stored by default, choose Tools→Options. Then, expand the Projects and Solutions node, select the General category, and enter the location in the Visual Studio Projects Location text box.

In the dialog box in this example, I'm starting a new file-system web site named Ch02FutureValue in the ASP.NET 2.0 Web Sites folder on my own PC. Then, when I click the OK button, Visual Studio creates the folder named Ch02FutureValue and puts the starting files for the web site in that folder. It also creates a solution file in the default folder for those files.

The folders and files that are used for developing a web site can be referred to as a *web project*. So in practice, web sites are often referred to as web projects, and vice versa. In a moment, you'll see that Visual Studio often uses the term *project* in the commands for working with web sites.

The New Web Site dialog box



A blank ASP.NET Web site

Location: File System C:\ASP.NET 2.0 Web sites\Ch02FutureValue

Language: Visual Basic

Three location options for ASP.NET web sites

Option	Description
File System	A web site created in a folder on your local computer or in a shared folder on a network. You can run the web site directly from the built-in development server or create an IIS virtual directory for the folder and run the application under IIS.
HTTP	A web site created under the control of an IIS web server. The IIS server can be on your local computer or on a computer that's available over a local area network.
FTP	A web site created on a remote hosting server.

Description

- An ASP.NET web application is called a *web site* under ASP.NET 2.0, so you use the File→New Web Site command to create a new ASP.NET 2.0 web site.
- A *web project* is a project that's used for the development of a web site. In practice, web sites are often referred to as web projects, and vice versa.
- Unlike previous versions of ASP.NET, ASP.NET 2.0 web sites don't use project files. Instead, they use web.config files to store project information.
- When you start a new web site, Visual Studio creates a solution file for the web site in the default location for solution files, which is normally My Documents\Visual Studio 2005\Projects.

Figure 2-1 How to start a new web site

How to work with the Visual Studio IDE

When you start a new web site, ASP.NET provides the starting folders and files for the site, including two files for the first *web form* of the site. The file named Default.aspx contains the HTML and asp code that defines the form, and the file named Default.aspx.vb contains the Visual Basic code that determines how the form works. Then, Visual Studio displays the aspx file for the web form as shown in figure 2-2.

If you've used Visual Studio for building Windows applications, you should already be familiar with the *Toolbox*, *Solution Explorer*, and *Properties window*, as well as the Standard toolbar. They work much the same for web applications as they do for Windows applications. The Solution Explorer, for example, shows the folders and files of the web site. In this example, the Solution Explorer shows one folder named App_Data, plus the two files for the default web form.

To design a web form, you use the *Web Forms Designer* that's in the center of this Integrated Development Environment (IDE). When you start a new web site, this Designer is displayed in *Source view*, which shows the starting HTML code for the first (or only) web form of the application. Normally, though, you'll do most of the design in *Design view*, which you can switch to by clicking on the Design button at the bottom of the Designer window.

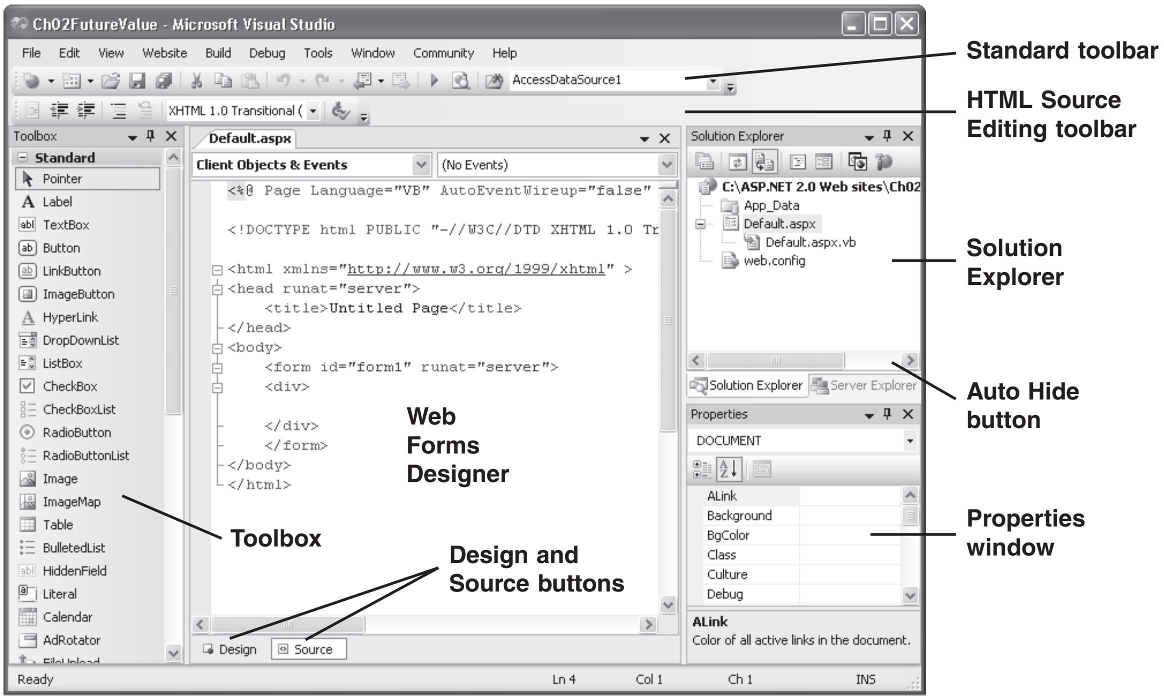
If you have your environment settings set to Web Developer, you'll notice that different toolbars are displayed depending on what view you're working in. In Source view, the Standard and HTML Source Editing toolbars are displayed. In Design view, the Standard and Formatting toolbars are displayed. This is typical of the way the Visual Studio IDE works. By the way, to change the environment settings, you use the Tools➔Import and Export Settings command.

As you go through the various phases of building a web site, you may want to close, hide, or size the windows that are displayed. You'll see some examples of this as you progress through this chapter, and this figure presents several techniques that you can use for working with the windows.

After you've designed a web form, you'll need to switch to the Code Editor, which replaces the Web Forms Designer in the center of the screen. Then, you can write the Visual Basic code in the code-behind file for the form. One way to switch to the Code Editor is to double-click on the code-behind file in the Solution Explorer. If, for example, you double-click on the file named Default.aspx.vb, you'll switch to the Code Editor and the starting code for that file will be displayed. Later in this chapter, you'll learn other ways to switch between the Web Forms Designer and the Code Editor.

As you work with Visual Studio, you'll see that it commonly provides several ways to do the same task. Some, of course, are more efficient than others, and we'll try to show you the best techniques as you progress through this book. Often, though, how you work is a matter of personal preference, so we encourage you to review and experiment with the toolbar buttons, the buttons at the top of the Solution Explorer, the tabs at the top of the Web Forms Designer or Code Editor, the shortcut menus that you get by right-clicking on an object, and so on.

The starting screen for a new web site



```
<%@ Page Language="VB" AutoEventWireup="false"

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Tr

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    </div>
    </form>
</body>
</html>
```

How to work with views and windows

- To change the Web Forms Designer from one view to another, click on the Design or Source button.
- To close a window, click on the close button in the upper right corner. To redisplay it, select it from the View menu.
- To hide a window, click on its Auto Hide button. Then, the window is displayed as a tab at the side of the screen, and you can display it by moving the mouse pointer over the tab. To restore the window, display it and click on the Auto Hide button again.
- To size a window, place the mouse pointer over one of its boundaries and drag it.

Description

- When you start a new web site, the primary window in the Visual Studio IDE is the Web Forms Designer window, or just *Web Forms Designer*. The three supporting windows are the *Toolbox*, the *Solution Explorer*, and the *Properties window*.
- You use the Web Forms Designer to design a *web form*. Later, to write the Visual Basic code for the form, you use the Code Editor as shown in figure 2-14.
- Visual Studio often provides several different ways to accomplish the same task. In this book, we'll try to show you the techniques that work the best.

Figure 2-2 How to work with the Visual Studio IDE

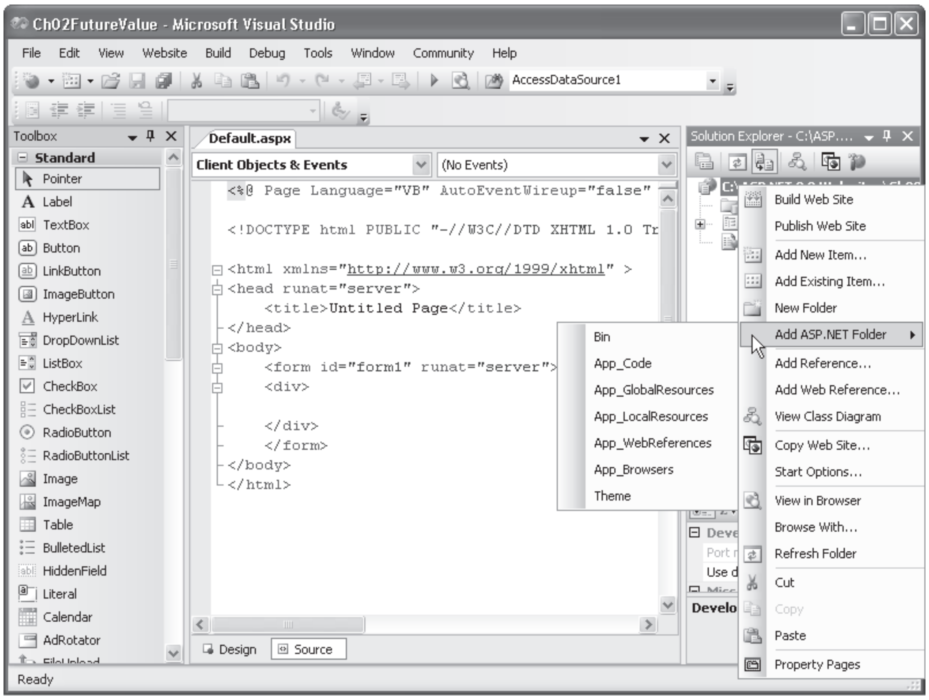
How to add folders and files to a web site

Right after you start a new web site, it often makes sense to add any other folders and existing files that the application is going to require. To do that, you can use the shortcut menus for the project or its folders in the Solution Explorer as shown in figure 2-3. As you can see, this menu provides a New Folder command as well as an Add Existing Item command.

For the Future Value application, I first added a folder named Images. To do that, I right-clicked on the project at the top of the Solution Explorer, chose the New Folder command, and entered the name for the folder. Then, I added an existing image file named MurachLogo.jpg to the Images folder. To do that, I right-clicked on the folder, chose Add Existing Item, and then selected the file from the dialog box that was displayed.

Those are the only other folders and files that I needed for the Future Value application, but often you'll need others. For instance, the application in the next chapter requires two existing business classes, an Access database, and a number of image files.

The Future Value project as a new folder is being added



```
<%@ Page Language="VB" AutoEventWireup="false"

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Tr

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    </div>
    </form>
</body>
</html>
```

How to add a folder to a web site

- To add a standard folder, right-click on the project or folder you want to add the folder to in the Solution Explorer and choose New Folder. Then, type a name for the folder and press Enter.
- To add a special ASP.NET folder, right-click on the project in the Solution Explorer and choose Add ASP.NET Folder. Then, select the folder from the list that's displayed.

How to add an existing item to a web site

- In the Solution Explorer, right-click on the project or on the folder that you want to add an existing item to. Then, select Add Existing Item and respond to the resulting dialog box.

Description

- When you create a new web form, Visual Studio generates the starting HTML for the form and displays it in Source view of the Web Forms Designer.
- Before you start designing the first web form of the application, you can use the Solution Explorer to add any other folders or files to the web site.

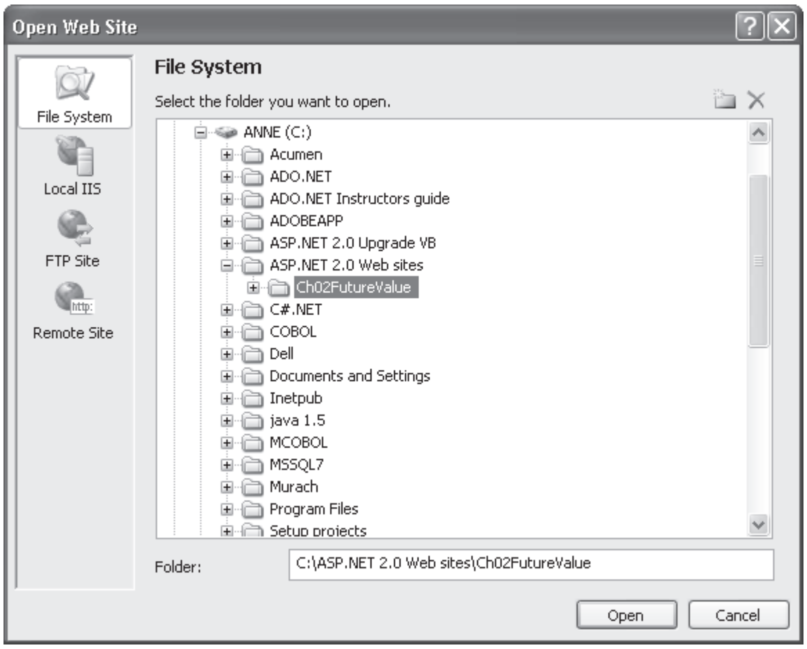
Figure 2-3 How to add folders and files to a web site

How to open or close an existing web site

Figure 2-4 presents three ways to open an existing web site. The Open Project and Recent Projects commands are the easiest to use, but the Open Web Site command provides more flexibility. In the Open Web Site dialog box, you can use the icons on the left to identify the type of web site that you're opening so you can open a web site directly from the web server on which it resides.

To close a project, you use the Close Project command. After you close a project for the first time, you'll be able to find it in the list of projects that you see when you use the Recent Projects command.

The Open Web Site dialog box



Folder: C:\ASP.NET 2.0 Web sites\Ch02FutureValue

Three ways to open a web site

- Use the File➔Open Project command.
- Use the File➔Recent Projects command.
- Use the File➔Open Web Site command.

How to use the Open Web Site dialog box

- To open a file-system web site, select File System on the left side of the dialog box, then use the File System tree to locate the web site.
- If a web site is managed by IIS on your own computer, you can open it by using the File System tree. Or, if you prefer, you can click Local IIS and select the web site from a list of sites available from IIS.
- The other icons on the left of the Open Web Site dialog box let you open web sites from an FTP site or from a remote IIS site.

How to close a project

- Use the File➔Close Project command.

Note

- The Recent Projects list and the Open Project and Open Web Site commands are also available from the Start page.

Figure 2-4 How to open or close an existing web site

How to use Design view to build a web form

Now that you know how to start, open, and close a web site, you're ready to learn how to build a web form. To start, I'll show you the web form that you're going to build. Then, I'll show you how to build it.

The design of the Future Value form

Figure 2-5 presents the design of a Future Value web form that calculates the future value of a fixed monthly investment. This form has enough features to give you a realistic idea of what's involved in the development of a form, but it's so simple that you won't be overwhelmed by detail. In this case, the entire application consists of this single form.

If you study the form, you can see that it contains six *web server controls*. These controls are derived from ASP.NET classes, and they have special features that make them suitable for web applications. This form contains one *drop-down list*, two *text boxes*, one *label*, and two *buttons*. These controls are analogous to the ones that you use for Windows applications.

When you click on a button in a web form, it automatically starts a postback to the server. When you click the Calculate button, for example, the application calculates the future value based on the values in the drop-down list and two text boxes. The result is displayed in the label when the form is returned to the browser. When you click on the Clear button, the text boxes and label are cleared and the value in the drop-down list is reset to 50.

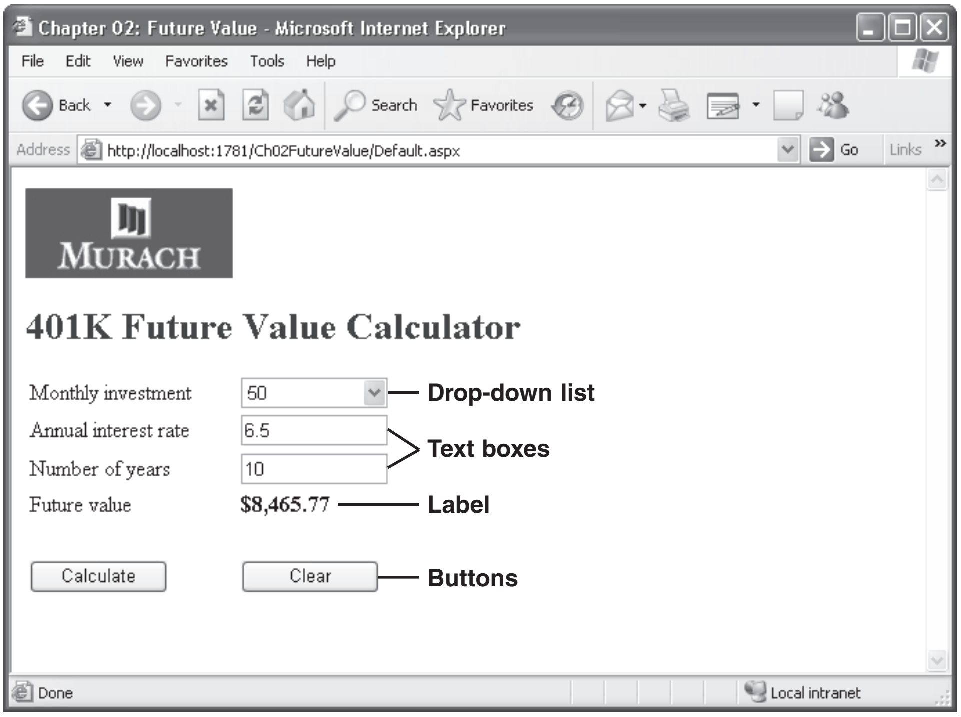
In contrast to the web server controls, the image at the top of this page (the Murach logo) is stored in an *HTML server control*. This is a second type of *server control* that you can use on a web form. The third type is a *validation control*, which you'll learn about later in this chapter.

The arrangement of the web server controls on this web form is done through an HTML table. Above the table, a heading has been entered and formatted. In the first column of the first four rows of the table, text has been entered that describes the data in the second column. The fifth row of the table contains no text or controls. And the six row contains the two buttons.

In the rest of this chapter, you'll learn how to build this web form, how to add validation controls to it, and how to write the code for its code-behind file. Then, you'll learn how to test a web application. At the end of this chapter, you'll find exercises that will walk you through the development of this Future Value application and help you experiment with other features of Visual Studio.

Throughout this chapter, please note that the term *page* is sometimes used to refer to a web form. That's because a web form represents a page that's sent to a browser.

The Future Value web form in a browser



The six web server controls used by the Future Value form

- The *drop-down list* can be used to select a monthly investment value ranging from 50 to 500.
- The two *text boxes* are used to enter values for the annual interest rate and the number of years that the monthly payments will be made.
- The *label* is used to display the future value that is calculated.
- The Calculate and Clear *buttons* are used to post the form back to the server and initiate the processing that needs to be done.

Description

- Besides the *web server controls*, the Future Value form uses an *HTML server control* to display the image at the top of the form, and it uses text to display the heading below the image. It also uses an HTML table to align the text and web server controls below the image and heading.
- When the user clicks on the Calculate button, the future value is calculated based on the three user entries and the results are displayed in the label control.
- When the user clicks on the Clear button, the two text boxes and the label are cleared and the drop-down list is reset to a value of 50.
- To end the application, the user can click the Close button in the upper right corner of the browser window.

Figure 2-5 The design of the Future Value form

How to use flow layout

By default, you develop web forms in *flow layout*. When you use flow layout, the text and controls you add to a form are positioned from left to right and from top to bottom. Because of that, the position of the controls can change when the form is displayed depending on the size of the browser window and the resolution of the display.

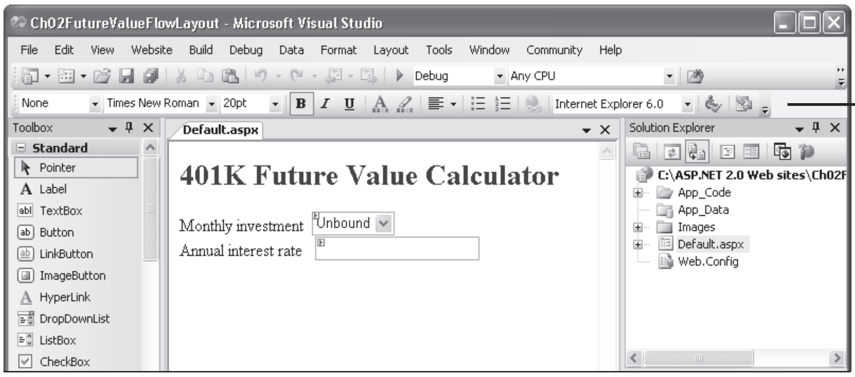
To understand how flow layout works, figure 2-6 shows the beginning of a version of the Future Value form that doesn't use a table to align its text and controls. To create this form, I started by typing the text for the heading directly into the form. Then, I pressed the Enter key twice to add space between the heading and the text and controls that follow it. Next, I typed the text that identifies the first control, I pressed the space bar twice to add some space after the text, and I added a drop-down list. When I added the drop-down list, it was placed immediately to the right of the text and spaces. I used similar techniques to enter the remaining text and text box.

Finally, I formatted the heading at the top of the form. To do that, I selected the text and then used the controls in the Formatting toolbar to change the font size to 20 points, to make the heading bold, and to change its color to blue.

You can see the result in the aspx code in this figure. Notice that the special code &nbsp; was inserted for each space between the text and the controls that follow. In addition, a Br element is inserted for each line break. To apply the formatting to the heading, a Strong element is used, along with a Span element with a Style attribute that specifies the font size and color.

Because you're limited to what you can do with spaces and line breaks, you'll frequently use tables to format a form in flow layout. For example, you can see in this figure that the drop-down list and the text box aren't perfectly aligned. In addition, there's not much space between the line that contains the drop-down list and the line that contains the text box. In the next figure, then, you'll learn how to add a table to a form so you can align the text and controls just the way you want.

The beginning of the Future Value form created using flow layout



The aspx code for the Future Value form

```
<form id="form1" runat="server">
<div>
    <strong><span style="font-size: 20pt; color: blue">
    401K Future Value Calculator</span></strong><br /><br />
    Monthly investment&nbsp;&nbsp;
    <asp:DropDownList ID="DropDownList1" runat="server">
    </asp:DropDownList><br />
    Annual interest rate&nbsp;&nbsp;
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
</div>
</form>
```

How to use flow layout

- When you add controls to a form in *flow layout*, they will appear one after the other, from left to right and from top to bottom. Flow layout is the default for web forms in Visual Studio 2005.
- To insert a space after a control, use the space bar. The special code &nbsp; is inserted into the aspx file.
- To insert a line break after a control, press Enter. A <br /> tag is inserted into the aspx file.
- To insert literal text, type it directly into the designer window. Then, you can use the controls in the Formatting toolbar and the commands in the Format menu to change the font or font size; apply bold, italics, or underlining; or apply foreground or background color.
- To align text and controls when you use flow layout, you normally use tables as described in the next figure.

Figure 2-6 How to use flow layout

How to add a table to a form

Figure 2-7 shows how to add a table to a form. In this case, a table of six rows and two columns has already been added to the form, but the Insert Table dialog box is displayed to show what the settings are for that table. Usually, you can keep the dialog box entries that simple, because you can easily adjust the table once it's on the form.

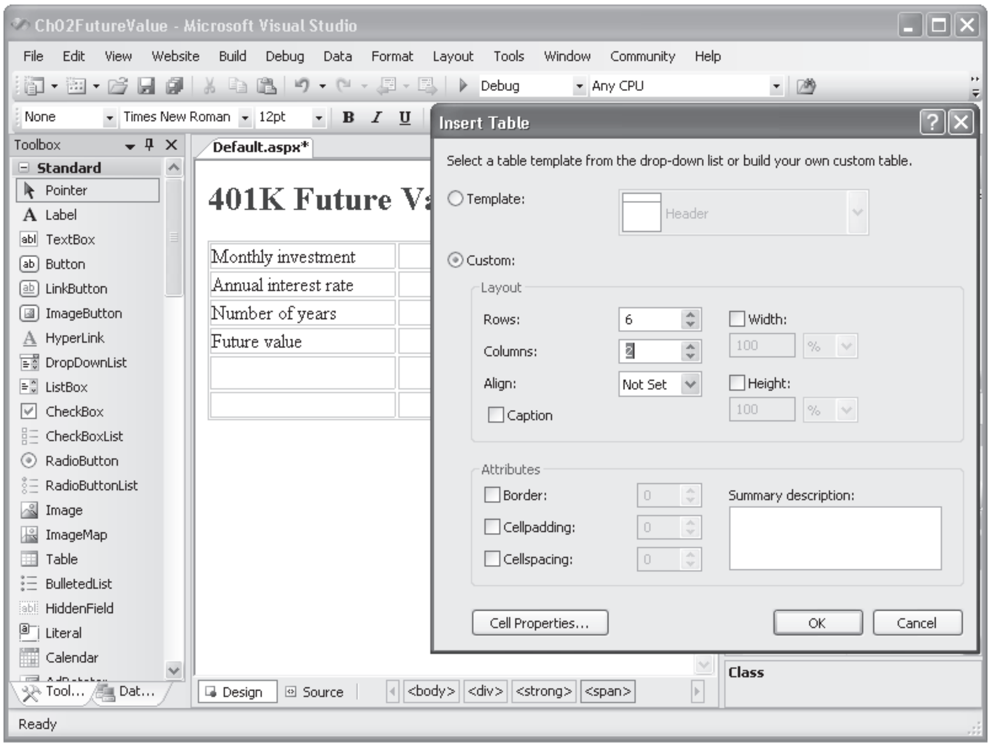
The easiest way to resize a row or column is to drag it by its border. To change the width of a column, drag it by its right border. To change the height of a row, drag it by its bottom border. You can also change the height and width of the entire table by selecting the table and then dragging it by its handles.

You can also format a table in Design view by selecting one or more rows or columns and then using the commands in the Layout menu or the shortcut menu that's displayed when you right-click the selection. These commands let you add, delete, or resize rows or columns. They also let you merge the cells in a row or column. If, for example, you want a control in one row to span two columns, you can merge the cells in that row.

How to add text to the cells of a table

In figure 2-7, you can see that text has been entered into the cells in the first four rows of the first column of the table. To do that, you just type the text into the cells. Then, you can format the text by selecting it and using the controls in the Formatting toolbar or the commands in the Format menu. If, for example, you want to bold the four text entries, you can select the four cells that contain the text and click on the Bold button.

The Future Value form with a table that has been inserted into it



401K Future V

Monthly investment
Annual interest rate
Number of years
Future value

Select a table template from the drop-down list or build your own custom table.

Template: Header

Custom:

Layout

Rows: 6

Columns: 2

Align: Not Set

Caption

Width: 100 %

Height: 100 %

Attributes

Border: 0

Cellpadding: 0

Cellspacing: 0

Summary description:

How to add a table to a form

- Use the Layout➔Insert Table command to display the Insert Table dialog box. Then, set the number of rows and columns that you want in the table, set any other options that you want, and click OK.

How to format a table after it has been added to a form

- To resize a row, drag it by its bottom border. To resize a column, drag it by its right border. To resize the entire table, select the table and then drag its handles.
- Select rows or columns and then use the commands in the Layout menu or the shortcut menu to add, delete, resize, or merge the rows or columns.

How to add and format text

- To add text to a table, type the text into the cells of the table.
- To format the text in a table, select the text, and then use the controls in the Formatting toolbar or the commands in the Format menu to apply the formatting.

Description

- To control the alignment of the text and controls on a web form in flow layout, you can use tables.

Figure 2-7 How to add a table to a form and text to a table

How to add server controls to a form

Figure 2-8 shows how to add web server controls to a form. To do that, you can just drag a control from the Standard group of the Toolbox and drop it on the form. Or, you can move the cursor where you want a control inserted and then double-click on the control in the Toolbox. This works whether you're placing a control within a cell of a table or outside of a table.

Once you've added the controls to the form, you can resize them by dragging the handles on their sides. If the controls are in a table, you may also want to resize the columns or rows of the table at this time. Keep in mind that you can resize a cell as well as the control within a cell, and sometimes you have to do both to get the formatting the way you want it.

Although you'll typically use web server controls on your web forms, you can also use HTML server controls. These controls appear in the HTML group of the Toolbox, and you can add them to a form the same way that you add web server controls. In addition, you can add an HTML image control to a form by dragging an image from the Solution Explorer. That's how I added the image at the top of the Future Value form.

How to set the properties of the controls

After you have placed the controls on a form, you need to set each control's properties so the control looks and works the way you want it to when the form is displayed. To set those properties, you work in the Properties window as shown in figure 2-8. To display the properties for a specific control, just click on it in Design view.

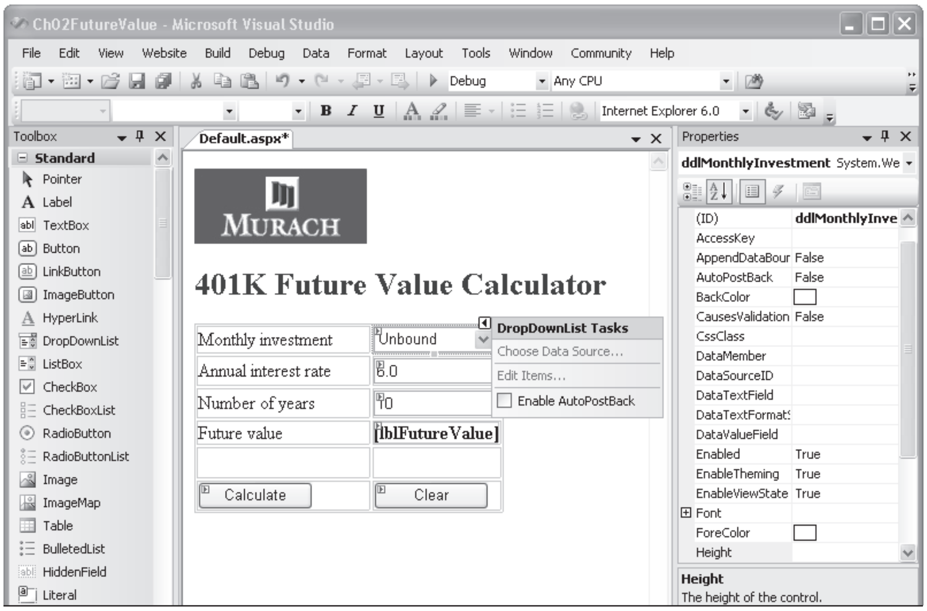
In the Properties window, you select a property by clicking it. Then, a brief description of that property is displayed in the pane at the bottom of the window. To change a property setting, you change the entry to the right of the property name by typing a new value or choosing a new value from a drop-down list. In some cases, a button with an ellipsis (…) on it will appear when you click on a property. In that case, you can click this button to display a dialog box that helps you set the property.

Some properties are displayed in groups. In that case, a plus sign appears next to the group name. This is illustrated by the Font property in this figure. To display the properties in a group, just click the plus sign next to the group name.

To display properties alphabetically or by category, you can click the appropriate button at the top of the Properties window. At first, you may want to display the properties by category so you have an idea of what the different properties do. Once you become more familiar with the properties, though, you may be able to find the ones you're looking for faster if you display them alphabetically.

As you work with properties, you'll find that most are set the way you want them by default. In addition, some properties such as Height and Width are set interactively as you size and position the controls in Design view. As a result, you usually only need to change a few properties for each control. The only

The Future Value form after six web server controls have been added to it



401K Future Value Calculator

Monthly investment	Unbound
Annual interest rate	6.0
Number of years	10
Future value	[lblFutureValue]

ddlMonthlyInvestment System.We

(ID)	ddlMonthlyInve
AccessKey	
AppendDataBour	False
AutoPostBack	False
BackColor	
CausesValidation	False
CssClass	
DataMember	
DataSourceID	
DataTextField	
DataTextFormat!	
DataValueField	
Enabled	True
EnableTheming	True
EnableViewState	True
Font	
ForeColor	
Height	

Height
The height of the control.

How to add a web server control to a web form

- Drag the control from the Standard group in the Toolbox to the form or to a cell in a table on the form. Or, move the cursor to where you want the control, and then double-click on the control in the Toolbox.

How to set the properties for a control

- Select a control by clicking on it, and all of its properties are displayed in the Properties window. Then, you can select a property in this window and set its value. When you select a property, a brief description is displayed in the pane at the bottom of the window.
- To change the Height and Width properties, you can drag one of the handles on a control. This also changes the Height and Width in the Properties window.
- To change the properties for two or more controls at the same time, select the controls. Then, the common properties of the controls are displayed in the Properties window.
- You can use the first two buttons at the top of the Properties window to sort the properties by category or alphabetically.
- You can use the plus and minus signs that are displayed in the Properties window to expand and collapse the list of properties.
- Many web server controls have a *smart tag menu* that provides options for performing common tasks and setting common properties. To display a smart tag menu, click the Smart Tag icon in the upper right of the control.

Note

- The image on this form was created by dragging the MurachLogo.jpg file from the Solution Explorer to the form. This creates an HTML image control.

Figure 2-8 How to add web server controls to a form and set their properties

property that I set for the drop-down list, for example, is the ID property. This property contains the name that you'll use to refer to the control when you write the Visual Basic code for the code-behind file.

Another way to set properties for some controls is to use the control's *smart tag menu*. In this figure, for example, you can see the smart tag menu for the drop-down list. You can use this menu to choose the data source for the control, which sets the DataSourceID, DataTextField, and DataValueField properties; edit the items in the list, which modifies the collection of items that's accessed through the Items property; and enable or disable the automatic posting of the page when a value is selected from the list, which sets the AutoPostBack property. Because smart tag menus help you set common properties, they're displayed automatically when you drag a control to a form. You can also display a smart tag menu by clicking the Smart Tag icon in the upper right corner of the control.

Common properties for web server controls

The first table in figure 2-9 presents the properties for web server controls that you're most likely to use as you develop web forms. If you've worked with Windows controls, you'll notice that many of the properties of the web server controls provide similar functionality. For example, you use the ID property to name a control that you need to refer to in code, and you can use the Text property to determine what's displayed in or on the control. However, the AutoPostBack, CausesValidation, EnableViewState, and Runat properties are unique to web server controls. Since you already know the purpose of the Runat property, I'll focus on the other three properties here.

The AutoPostBack property determines whether the page is posted back to the server when the user changes the value of the control. Note that this property is only available with certain controls, such as check boxes, drop-down lists, and radio buttons. Also note that this property isn't available with button controls. That's because button controls always post a page back to the server.

The CausesValidation property is available for button controls and determines whether the validation controls are activated when the user clicks the button. This lets you check for valid data before the form is posted back to the server. You'll learn more about validation controls a few figures from now.

The EnableViewState property determines whether a server control retains its property settings from one posting to the next. For that to happen, the EnableViewState property for both the form and the control must be set to True. Since that's normally the way you want this property set, True is the default.

The second table in this figure lists four more properties that are commonly used with drop-down lists and list boxes. However, you don't need to set these at design time. Instead, you use them when you write the Visual Basic code for the code-behind file. For instance, you use the Items collection to add, insert, and remove ListItem objects. And you use the SelectedValue property to retrieve the value of the currently selected item. You'll learn more about these properties when you review the code-behind file for the Future Value form.

Common web server control properties

Property	Description
AutoPostBack	Determines whether the page is posted back to the server when the value of the control changes. Available with controls such as a check box, drop-down list, radio button, or text box. The default value is False.
CausesValidation	Determines whether the validation that's done by the validation controls occurs when you click on the button, link button, or image button. The default value is True. (You'll learn how to use two common validation controls later in this chapter.)
EnableViewState	Determines whether the control maintains its view state between HTTP requests. The default value is True.
Enabled	Determines whether the control is functional. The default value is True.
Height	The height of the control.
ID	The name that's used to refer to the control.
Runat	Indicates that the control will be processed on the server by ASP.NET.
TabIndex	Determines the order in which the controls on the form receive the focus when the Tab key is pressed.
Text	The text that's displayed in the control.
ToolTip	The text that's displayed when the user hovers the mouse over the control.
Visible	Determines whether a control is displayed or hidden.
Width	The width of the control.

Common properties of drop-down list and list box controls

Property	Description
Items	The collection of ListItem objects that represents the items in the control. Although you can set the values for these list items at design time, you normally use code to add, insert, and remove the items in a list or list box.
SelectedItem	The ListItem object for the currently selected item.
SelectedIndex	The index of the currently selected item. If no item is selected in a list box, the value of this property is -1.
SelectedValue	The value of the currently selected item.

Note

- When buttons are clicked, they always post back to the server. That's why they don't have AutoPostBack properties.

Figure 2-9 Common properties for web server controls

How to work in Source view

As you design a form in Design view, HTML and asp tags are being generated in Source view. This is the code that's used to render the web page that's sent to the user's browser. What you see in Design view is just a visual representation of that code. In figure 2-10, you can see some of the tags for the Future Value form after the Designer has been switched to Source view.

How to use Source view to modify the design

As you saw in the last chapter, HTML consists of tags. For instance, the <form> and </form> tags mark the start and end of the HTML code for a web form. And the <table> and </table> tags mark the start and end of the HTML code for a table.

In addition to the HTML tags, ASP.NET adds asp tags for the web server controls that are added to the form. For instance, the <asp:DropDownList> and </asp:DropDownList> tags mark the start and end of the code for a drop-down list. Within these tags, you'll find the code for the property settings of the controls. Note, however, that all of this asp code is converted to HTML before the page can be sent to a browser, because a browser can only interpret HTML.

Because the file that contains the source code for a web form has an aspx extension, we refer to the source code for a form as *aspx code*. This also indicates that the code contains both HTML and asp tags.

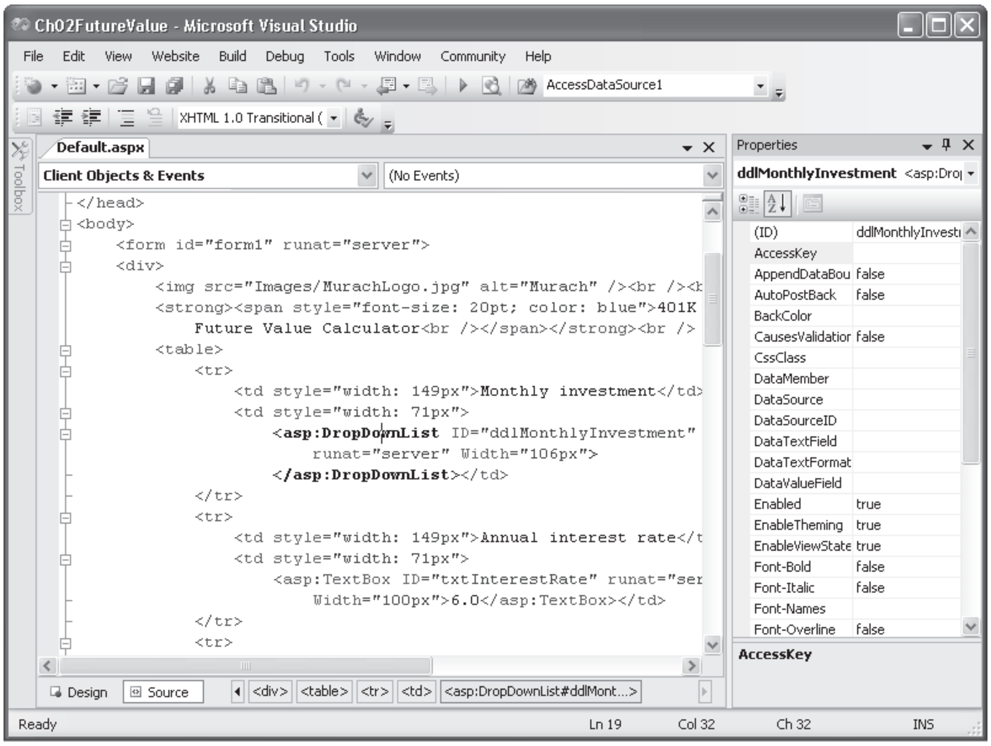
In case you need it, chapter 5 presents a crash course in HTML. In the meantime, though, you may be surprised to discover how easy it is to modify the design of a form by adjusting the aspx code using the *HTML Editor*.

To start, you can modify the title of the form that you'll find between the Head tags near the top of the source code. This is the title that's displayed in the title bar of the browser when the form is run (see figure 2-5). In this example, the title has been changed from "Untitled Page" to "Chapter 02: Future Value." As you will see, all of the applications in this book have titles that indicate both the chapter number and the type of application.

You can also use this technique to change the text that has been entered into a form or to change some the settings for HTML elements. If, for example, you want to change the text in the first row of the table from "Monthly investment" to "Investment amount," you can just edit the text in Source view. If you want to change the width of a cell, you can edit that entry. And if you want to modify the color for the heading, you can do that too. As you edit, just follow the syntax of the other entries, which will be easier to do after you read chapter 5.

To change the properties of a server control, you can click in the starting asp tag to select the control. Then, you can use the Properties window just as if you were in Design view. When you change a property, the *attribute* that represents the property in the asp tag for the control is changed. You can also change the attributes directly in the source code whenever the syntax is obvious. That's often the fastest way to make an adjustment.

The design of the Future Value form in Source view



Ch02FutureValue - Microsoft Visual Studio
File Edit View Website Build Debug Tools Window Community Help
AccessDataSource1
XHTML 1.0 Transitional (
Toolbox
Default.aspx
Client Objects & Events
(No Events)
Design
Source
<div> <table> <tr> <td> <asp:DropDownList#ddlMont...>
Ready
Ln 19
Col 32
Ch 32
INS
Properties
ddlMonthlyInvestment <asp:Dro
AccessKey

```
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <img src="Images/MurachLogo.jpg" alt="Murach" /><br /><b
        <strong><span style="font-size: 20pt; color: blue">401K
            Future Value Calculator<br /></span></strong><br />
        <table>
            <tr>
                <td style="width: 149px">Monthly investment</td>
                <td style="width: 71px">
                    <asp:DropDownList ID="ddlMonthlyInvestment"
                        runat="server" Width="106px">
                    </asp:DropDownList></td>
            </tr>
            <tr>
                <td style="width: 149px">Annual interest rate</t
                <td style="width: 71px">
                    <asp:TextBox ID="txtInterestRate" runat="ser
                        Width="100px">6.0</asp:TextBox></td>
            </tr>
            <tr>
```

(ID)	ddlMonthlyInvest
AccessKey	
AppendDataBou	false
AutoPostBack	false
BackColor	
CausesValidatior	false
CssClass	
DataMember	
DataSource	
DataSourceID	
DataTextField	
DataTextFormat	
DataValueField	
Enabled	true
EnableTheming	true
EnableViewState	true
Font-Bold	false
Font-Italic	false
Font-Names	
Font-Overline	false

How to change the title of the form

- Change the text between the <title> and </title> tags.

How to change the HTML and text for the form

- Change the source code itself.

How to change the property settings for a control

- To select a control, move the insertion point into the asp tag for the control. Then, use the Properties window to change the property settings for the control. Or, you can modify the property settings in the source code itself.

Description

- Design view presents a visual representation of the code that you see in Source view.
- The source code includes HTML tags and asp tags. Before the form is sent to a browser, the asp tags are converted to HTML because browsers can only run HTML.
- The properties you set for a control appear as *attributes* in the asp tag for the control.
- We refer to the source code as *aspx code*, because the source files have aspx extensions.

Figure 2-10 How to use Source view to modify the design of a form

The aspx code for the Future Value form

Figure 2-11 presents the aspx code for the Future Value form that has been developed thus far (except for the Page and Doctype directives, which you'll learn more about in chapter 5). For now, though, please note that the code for the title that's displayed in the web browser is between the <head> tags, and the code for the form design is between the <div> tags.

Within the <div> tags, I've highlighted the code for the HTML image control and the code for the six web server controls. If you study this code, you can see how the properties are set for each of these controls. For instance, you can see that I set the Width properties of the button controls to 100 pixels so they are both the same width.

You can also see that I set the width of the drop-down list to 106 pixels, even though it appears to be the same width as the text boxes, which are 100 pixels wide. And you can see that I had to set the height of the cells in the fifth row of the table to give that row an adequate height. My point is that the sizing properties in the aspx code aren't always consistent, so you often have to fiddle with these properties to get the design the way you want it.

Before I go on, I want to point out that when you create an HTML image control by dragging the image from the Solution Explorer, it's generated with just a Src property that identifies the name and location of the image. However, the version of HTML that Visual Studio 2005 uses also requires the Alt attribute, which specifies the text that's displayed if for some reason the image can't be displayed. Because of that, I had to add this property to the HTML code for the control.

The aspx code for the Future Value form

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Chapter 02: Future Value</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <img src="Images/MurachLogo.jpg" alt="Murach"/><br /><br />
        <strong><span style="font-size: 20pt; color: blue">401K
            Future Value Calculator<br /></span></strong><br />
        <table>
            <tr>
                <td style="width: 149px">Monthly investment</td>
                <td style="width: 71px">
                    <asp:DropDownList ID="ddlMonthlyInvestment"
                        runat="server" Width="106px">
                    </asp:DropDownList></td>
            </tr>
            <tr>
                <td style="width: 149px">Annual interest rate</td>
                <td style="width: 71px">
                    <asp:TextBox ID="txtInterestRate" runat="server"
                    Width="100px">6.0</asp:TextBox></td>
            </tr>
            <tr>
                <td style="width: 149px">Number of years</td>
                <td style="width: 71px">
                    <asp:TextBox ID="txtYears" runat="server"
                    Width="100px">10</asp:TextBox></td>
            </tr>
            <tr>
                <td style="width: 149px">Future value</td>
                <td style="width: 71px">
                    <asp:Label ID="lblFutureValue" runat="server"
                    Font-Bold="True"></asp:Label></td>
            </tr>
            <tr>
                <td style="width: 149px; height: 25px"></td>
                <td style="width: 71px; height: 25px"></td>
            </tr>
            <tr>
                <td style="width: 149px">
                    <asp:Button ID="btnCalculate" runat="server"
                     BackColor="LightGray" Text="Calculate"
                     Width="100px" /></td>
                <td style="width: 71px">
                    <asp:Button ID="btnClear" runat="server"
                     BackColor="LightGray" Text="Clear"
                     Width="100px" /></td>
            </tr>
        </table>
    </div>
    </form>
</body>
</html>
```

Figure 2-11 The aspx code for the Future Value form

How to add validation controls to a form

A *validation control* is a type of ASP.NET control that's used to validate input data. The topics that follow introduce you to the validation controls and show you how to use two of the commonly used controls. Then, in chapter 7, you can learn all the skills that you need to master the use of these controls.

An introduction to the validation controls

Figure 2-12 shows the Validation group in the Toolbox. It offers five controls that can be called *validators*. These are the controls that you use to check that the user has entered valid data. You can use the last control in this group, the validation summary control, to display all the errors that have been detected by the validators on the form.

The easiest way to add a validation control to a web form is to drag it from the Toolbox. In this example, four validators have been added to the form: two required field validators and two range validators. In this case, the controls have been added below the table so ASP.NET will use flow layout to position the controls. However, these controls could have been added to a third column of the table. Although these controls aren't displayed when the form is displayed, the messages in their ErrorMessage properties are displayed if errors are detected.

In this case, the first required field validator checks to make sure that a value has been added to the text box for the interest rate, and the first range validator checks to make sure that this value ranges from 1 to 20. Similarly, the second required field validator checks to make sure that a value has been entered in the text box for years, and the second range validator checks to make sure that this value ranges from 1 to 45.

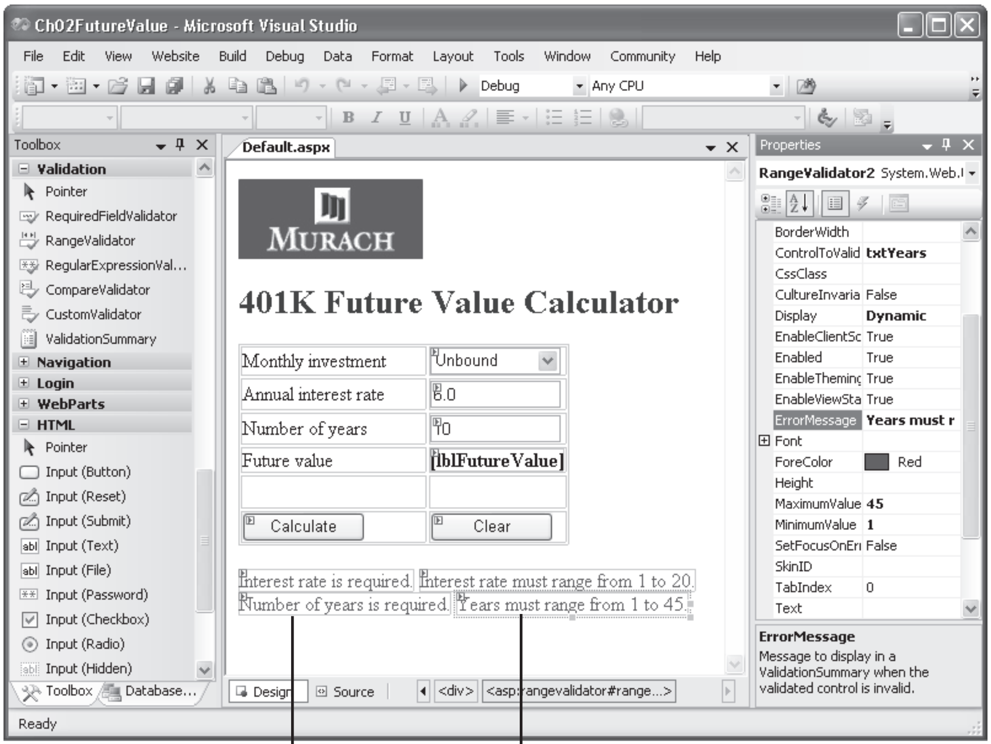
Validation tests are typically done on the client before the page is posted to the server. That way, a round trip to the server isn't required to display error messages if any invalid data is detected.

In most cases, client-side validation is done when the focus leaves an input control that has validators associated with it. That can happen when the user presses the Tab key to move to the next control or clicks another control to move the focus to that control. Validation is also done when the user clicks on a button that has its CausesValidation property set to True.

To perform client-side validation, a browser must support *Dynamic HTML*, or *DHTML*. Because most browsers in use today support DHTML, validation can usually be done on the client. However, validation is always done on the server too when a page is submitted. ASP.NET does this validation after it initializes the page.

When ASP.NET performs the validation tests on the server, it sets the IsValid property of each validator to indicate if the test was successful. In addition, after all the validators are tested, it sets the IsValid property of the page to indicate if all the input data is valid. You can test this property in the event handler for the event that causes the page to be posted to the server. You'll see how this works when you review the code-behind file for this form.

The validation controls on the Future Value form



MURACH

401K Future Value Calculator

Monthly investment	Unbound
Annual interest rate	6.0
Number of years	10
Future value	[lblFutureValue]
Calculate	Clear

Interest rate is required. Interest rate must range from 1 to 20.
Number of years is required. Years must range from 1 to 45.

Properties

RangeValidator2 System.Web.I

BorderWidth	
ControlToValid	txtYears
CssClass	
CultureInvaria	False
Display	Dynamic
EnableClientSc	True
Enabled	True
EnableThemin	True
EnableViewSta	True
ErrorMessage	Years must r
Font	
ForeColor	Red
Height	
MaximumValue	45
MinimumValue	1
SetFocusOnEr	False
SkinID	
TabIndex	0
Text	

ErrorMessage
Message to display in a ValidationSummary when the validated control is invalid.

Description

- You can use *validation controls* to test user input and produce error messages. The validation is performed when the focus leaves the control that's being validated and also when the user clicks on a button control whose CausesValidation property is set to True.
- Each validation control is associated with a specific server control, but you can associate one or more validation controls with a single server control.
- The validation controls work by running client-side script. Then, if the validation fails, the page isn't posted back to the server. However, the validation is also performed on the server in case the client doesn't support scripts.
- If the client doesn't support scripts, you can test whether validation has been successful on the server by testing whether the IsValid property of the page is True.

Figure 2-12 An introduction to the validation controls

How to use the required field validator

To use the *required field validator*, you set the properties shown in the table at the top of figure 2-13. These are the properties that are used by all the validators.

To start, you associate the validation control with a specific input control on the form through its ControlToValidate property. Then, when the focus leaves the input control or the user clicks on a button whose CausesValidation property is set to True, the validator checks whether a value has been entered into the input control. If not, the message in the ErrorMessage property is displayed.

When an error occurs, the Display property of the validation control determines how the message in the ErrorMessage property is displayed. When you use flow layout, Dynamic usually works the best for this property. If you use a validation summary control as explained in chapter 7, though, you can change this property to None.

If you look at the aspx code in this figure, you can see how the properties are set for the two required field validators that are shown in the previous figure. The first one validates the text box named txtInterestRate. The second one validates the text box named txtYears. This aspx code will be added after the end tag for the table in the code in figure 2-11.

How to use the range validator

The *range validator* lets you set the valid range for an input value. To use this control, you set the properties in the first table in figure 2-13, plus the properties in the second table. In particular, you set the minimum and maximum values for an input value.

For this control to work correctly, you must set the Type property to the type of data you're testing for. Because the interest rate entry can have decimal positions, for example, the Type property for the first range validator is set to Double. In contrast, because the year entry should be a whole number, the Type property for the second range validator is set to Integer. You can see how all of the properties for the two range validators are set by reviewing the aspx code.

Common validation control properties

Property	Description
ControlToValidate	The ID of the control to be validated.
Display	Determines how an error message is displayed. Specify Static to allocate space for the message in the page layout, Dynamic to have the space allocated when an error occurs, or None to display the errors in a validation summary control.
ErrorMessage	The message that's displayed in the validation control when the validation fails.

Additional properties of a range validator

Property	Description
Maximum	The maximum value that the control can contain.
Minimum	The minimum value that the control can contain.
Type	The data type to use for range checking (String, Integer, Double, Date, or Currency).

The aspx code for the validation controls on the Future Value form

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
    ControlToValidate="txtInterestRate" Display="Dynamic"
    ErrorMessage="Interest rate is required.">
</asp:RequiredFieldValidator>
<asp:RangeValidator ID="RangeValidator1" runat="server"
    ControlToValidate="txtInterestRate" Display="Dynamic"
    ErrorMessage="Interest rate must range from 1 to 20."
    MaximumValue="20" MinimumValue="1" Type="Double">
</asp:RangeValidator><br />
<asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server"
    ControlToValidate="txtYears" Display="Dynamic"
    ErrorMessage="Number of years is required.">
</asp:RequiredFieldValidator>
<asp:RangeValidator ID="RangeValidator2" runat="server"
    ControlToValidate="txtYears" Display="Dynamic"
    ErrorMessage="Years must range from 1 to 45."
    MaximumValue="45" MinimumValue="1" Type="Integer">
</asp:RangeValidator>
```

Description

- The *required field validator* is typically used with text box controls, but can also be used with list controls.
- The *range validator* tests whether a user entry falls within a valid range.
- If the user doesn't enter a value into the input control that a range validator is associated with, the range validation test passes. Because of that, you should also provide a required field validator if a value is required.

Figure 2-13 How to use the required field and range validators

How to add code to a form

To add the functionality required by a web form, you add Visual Basic code to its code-behind file. This code responds to the events that the user initiates on the form. This code also responds to events that occur as a form is processed.

How to use the Code Editor

To create and edit Visual Basic code, you use the *Code Editor* shown in figure 2-14. The easiest way to display the Code Editor window is to double-click the form or a control in the Web Forms Designer window. That displays the code-behind file for the form.

If you double-click the form in Design view, Sub and End Sub statements for the Load event of the page are generated. If you double-click a control, Sub and End Sub statements for the default event of the control are generated. If you double-click on a button control, for example, an *event procedure* (or *event handler*) for the Click event of that control is created. Then, you can enter the code for that procedure between the generated Sub and End Sub statements.

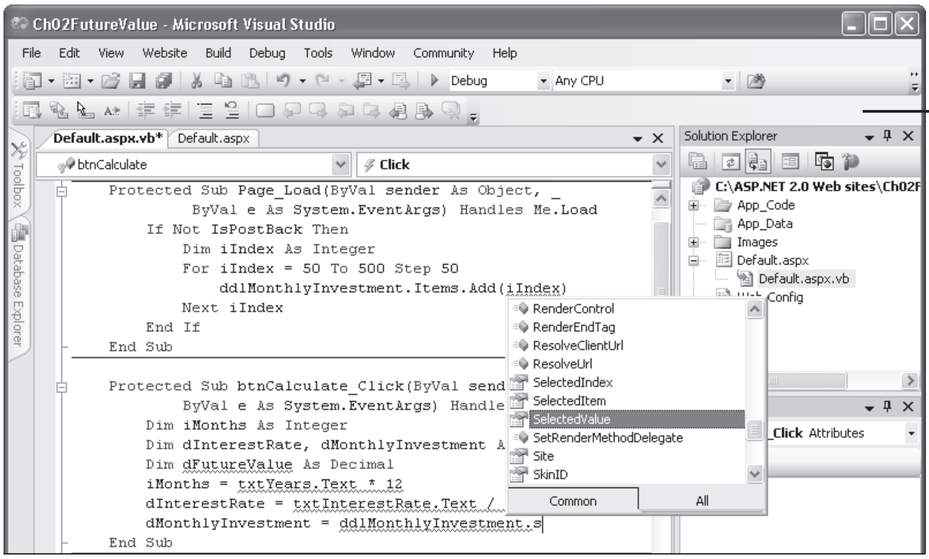
To create procedures for other events, you can use the drop-down lists at the top of the Code Editor window. The list at the left side of the window includes all of the available objects. When you select one of these objects, the list at the right side of the window lists all the events for that object. When you select an event, Visual Studio generates Sub and End Sub statements for the event handler.

You can also code *general procedures* by entering code directly into the Code Editor window. To create a *Sub procedure*, for example, you enter a Sub statement. And to create a *Function procedure*, or just *function*, you enter a Function statement. When you press the Enter key after entering one of these statements, the End Sub or End Function statement is generated for you. Then, you can enter the code required to implement the procedure between these statements, and you can call the procedure from another procedure.

As you work with the Code Editor, you'll notice that it provides some powerful features that can help you code more quickly and accurately. One of the most useful of these features is the Auto List Members feature provided by IntelliSense. This feature displays a list of members that are available for an object when you type the object name and a period. Then, you can highlight the member you want by clicking on it, typing the first few letters of its name, or using the arrow keys to scroll through the list. In this figure, you can see the members that are listed for a drop-down list after the first character of the member name is entered. When you press the Tab key, the member you select is inserted into your code.

You can also use the Text Editor toolbar to work with code in the Code Editor. You can use it to perform functions such as commenting or uncommenting several lines of code at once, increasing or decreasing the indentation of several lines of code, and working with bookmarks. If you experiment with this toolbar, you should quickly see how it works.

A project with the Code Editor window displayed



```
    Protected Sub Page_Load(ByVal sender As Object, _
            ByVal e As System.EventArgs) Handles Me.Load
        If Not IsPostBack Then
            Dim iIndex As Integer
            For iIndex = 50 To 500 Step 50
                ddlMonthlyInvestment.Items.Add(iIndex)
            Next iIndex
        End If
    End Sub

    Protected Sub btnCalculate_Click(ByVal send
            ByVal e As System.EventArgs) Handle
        Dim iMonths As Integer
        Dim dInterestRate, dMonthlyInvestment A
        Dim dFutureValue As Decimal
        iMonths = txtYears.Text * 12
        dInterestRate = txtInterestRate.Text /
        dMonthlyInvestment = ddlMonthlyInvestment.s
    End Sub
```

Three ways to open or switch to a file in the Code Editor window

- Select a web form in the Solution Explorer and click the View Code button at the top of the Solution Explorer. Double-click on a Visual Basic file (.aspx.vb or .vb) in the Solution Explorer. Or, click on a tab at the top of the Web Forms Designer (if the file is already open).

Four ways to start an event procedure

- Double-click on a blank portion of a web form to start an event procedure for the Load event of the page.
- Double-click on a control in the Web Forms Designer to start an event procedure for the default event of that control.
- Select a control in the Web Forms Designer, click the Events button in the Properties window (the button with the lightening bolt), and double-click the event you want to create an event procedure for.
- Select a control from the drop-down list at the top left of the Code Editor window, and select an event from the drop-down list at the top right. To create an event procedure for a page event, select (Page Events) from the first drop-down list.

Description

- The *Code Editor* includes powerful text editing features such as automatic indentation, syntax checking, and statement completion (as shown above).
- To enter a Sub procedure or function, you type the procedure or function from scratch, but Visual Studio will insert the End Sub or End Function statement.

Figure 2-14 How to use the Code Editor

How to use page and control events

The first table in figure 2-15 presents some of the common events for working with web pages. The Init and Load events of a page occur whenever a page is requested from the server. The Init event occurs first, and it's used by ASP.NET to restore the view state of the page and its controls. Because of that, you don't usually create an event handler for this event. Instead, you add any initialization code to the event handler for the Load event. You'll see how this works in the next figure.

In contrast, the PreRender event is raised after all the control events for the page have been processed. It's the last event to occur before a page is rendered to HTML. In later chapters, you'll see a couple of cases in which this event is useful.

The second table in this figure lists some of the common events for web server controls. When the user clicks a button, for example, the Click event of that control is raised. Then, the page is posted back to the server, the event handlers for the Init and Load events of the page are executed, if present, followed by the event handler for the Click event of the control that was clicked.

The TextChanged event occurs when the user changes the value in a text box. In most cases, you won't code an event handler for the TextChanged event. However, you might code an event handler for the CheckedChanged event that occurs when the user clicks a radio button or checks a check box. You might also code an event handler for the SelectedIndexChanged event that occurs when the user selects an item from a drop-down list.

If you want the event handler for one of these events to be executed immediately when the event occurs, you can set the AutoPostBack property of the control to True. Then, the event handler will be executed after the Load and Init event handlers for the page. Note that if you don't set the AutoPostBack property to True, the event is still raised, but the event handler isn't executed until another user action causes the page to be posted to the server. Then, the event handlers for the Load and Init events of the page are executed, followed by the event handlers for the control events in the order they were raised.

In this figure, you can see the event handler for the Click event of the Clear button on the Future Value form. Note that the name for this event handler is btnClear_Click, which is the ID of the button followed by the name of the event. Remember, though, that the Handles clause actually determines what event or events the procedure responds to. In this procedure, the value in the drop-down list is reset to 50, and the text boxes and label are reset to empty strings.

Incidentally, using the Handles clause is the default method for wiring events to their event handlers. However, you can also wire an event to an event handler by naming the event handler on the event attribute of a control. Although you'll learn how this works in chapter 6, there's usually no reason to change from using the Handles clause.

Common ASP.NET page events

Event	Procedure name	Occurs when…
Init	Page_Init	A page is requested from the server. This event is raised before the view state of the page controls has been restored.
Load	Page_Load	A page is requested from the server, after all controls have been initialized and view state has been restored. This is the event you typically use to perform initialization operations such as retrieving data and initializing form controls.
PreRender	Page_PreRender	All the control events for the page have been processed but before the HTML that will be sent back to the browser is generated.

Common ASP.NET control events

Event	Occurs when…
Click	The user clicks a button, link button, or image button control.
TextChanged	The user changes the value in a text box.
CheckedChanged	The user selects a radio button in a group of radio buttons or selects or unselects a check box.
SelectedIndexChanged	The user selects an item from a list box, a drop-down list, a check box list, or a radio button list.

Code for the Click event of the btnClear button

```
Protected Sub btnClear_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles btnClear.Click
    ddlMonthlyInvestment.Text = 50
    txtInterestRate.Text = ""
    txtYears.Text = ""
    lblFutureValue.Text = ""
End Sub
```

Description

- All of the events associated with an ASP.NET web page and its server controls are executed on the server. Because of that, the page must be posted back to the server to process any event for which you've coded an event handler.
- When a page is posted back to the server, the Init and Load events are always raised so any event handlers for those events are run first. Next, the event handlers for any control events that were raised are executed in the ordered they were raised. When these event handlers finish, the PreRender event is raised and any event handler for that event is run.

Figure 2-15 How to use page and control events

The Visual Basic code for the Future Value form

Figure 2-16 presents the complete Visual Basic code for the code-behind file of the Future Value form. It consists of three event handlers that handle the Load event for the page and the Click events of the Calculate and Clear buttons. This code also includes a function procedure named FutureValue that is called by the event handler for the Click event of the Calculate button.

In this code, I've highlighted the two page properties that are commonly tested in the code for web forms. The first one is the IsPostBack property that's used in the Page_Load procedure. If it is True, it means that the page is being posted back from the user. If it is False, it means that the page is being requested by the user for the first time.

As a result, the statements within the If statement in the Page_Load procedure are only executed if the page is being requested for the first time. In that case, the values 50 through 500 are added to the drop-down list. For all subsequent requests by that user, the IsPostBack property will be True so the values aren't added to the drop-down list.

The other page property that's commonly tested is the IsValid property. It's useful when the user's browser doesn't support the script for the validation controls. In that case, the application has to rely on the validation that's always done on the server. Then, if IsValid is True, it means that all of the input data is valid. But if IsValid is False, it means that one or more controls contain invalid input data so the processing shouldn't be done.

In the btnCalculate_Click procedure, you can see how the IsValid test is used. If it isn't True, the processing isn't done. But otherwise, this procedure gets the years and interest rate values from the text boxes and converts them to monthly units. Then, it uses the SelectedValue property of the drop-down list to get the value of the selected item, which represents the investment amount. Last, it calls the FutureValue function to calculate the future value, uses the FormatCurrency method to format the future value, and puts the formatted value in the label of the form. When this procedure ends, the web form is sent back to the user's browser.

With the exception of the IsPostBack and IsValid properties, this is all standard Visual Basic code so you shouldn't have any trouble following it. But if you do, you can quickly upgrade your Visual Basic skills by getting our latest Visual Basic book.

The Visual Basic code for the Future Value form

```
Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, _
             ByVal e As System.EventArgs) Handles Me.Load
        If Not IsPostBack Then
            Dim iIndex As Integer
            For iIndex = 50 To 500 Step 50
                ddlMonthlyInvestment.Items.Add(iIndex)
            Next iIndex
        End If
    End Sub

    Protected Sub btnCalculate_Click(ByVal sender As Object, _
            ByVal e As System.EventArgs) Handles btnCalculate.Click
        Dim iMonths As Integer
        Dim dInterestRate, dMonthlyInvestment As Decimal
        Dim dFutureValue As Decimal
        If IsValid Then
            iMonths = txtYears.Text * 12
            dInterestRate = txtInterestRate.Text / 12 / 100
            dMonthlyInvestment = ddlMonthlyInvestment.SelectedValue
            dFutureValue = FutureValue(iMonths, dInterestRate, dMonthlyInvestment)
            lblFutureValue.Text = FormatCurrency(dFutureValue)
        End If
    End Sub

    Private Function FutureValue(ByVal Months As Integer, _
            ByVal InterestRate As Decimal, _
            ByVal MonthlyInvestment As Decimal) As Decimal
        Dim iIndex As Integer
        Dim dFutureValue As Decimal
        For iIndex = 1 To Months
            dFutureValue = (dFutureValue + MonthlyInvestment) _
                         * (1 + InterestRate)
        Next iIndex
        Return dFutureValue
    End Function

    Protected Sub btnClear_Click(ByVal sender As Object, _
            ByVal e As System.EventArgs) Handles btnClear.Click
        ddlMonthlyInvestment.Text = 50
        txtInterestRate.Text = ""
        txtYears.Text = ""
        lblFutureValue.Text = ""
    End Sub

End Class
```

Figure 2-16 The Visual Basic code for the Future Value form

How to test a web application

After you design the forms and develop the code for a web application, you need to test it to be sure it works properly. Then, if you discover any errors in the application, you can debug it, correct the errors, and test it again.

In chapter 4, you'll learn all the skills you need to test and debug a web application. For now, I just want to show you how to run a web site with the built-in development server so you can test any applications that you develop for this chapter. Then, I'll show you the HTML code that's sent to the browser so you can see how that works.

How to run a web site with the built-in development server

When you run a file-system web site by using one of the techniques in figure 2-17, Visual Studio 2005 compiles the application. If the application compiles without errors, Visual Studio automatically launches the built-in ASP.NET 2.0 Development Server and displays the starting page of the web site in your default browser. Then, you can test the application to make sure that it works the way you want it to.

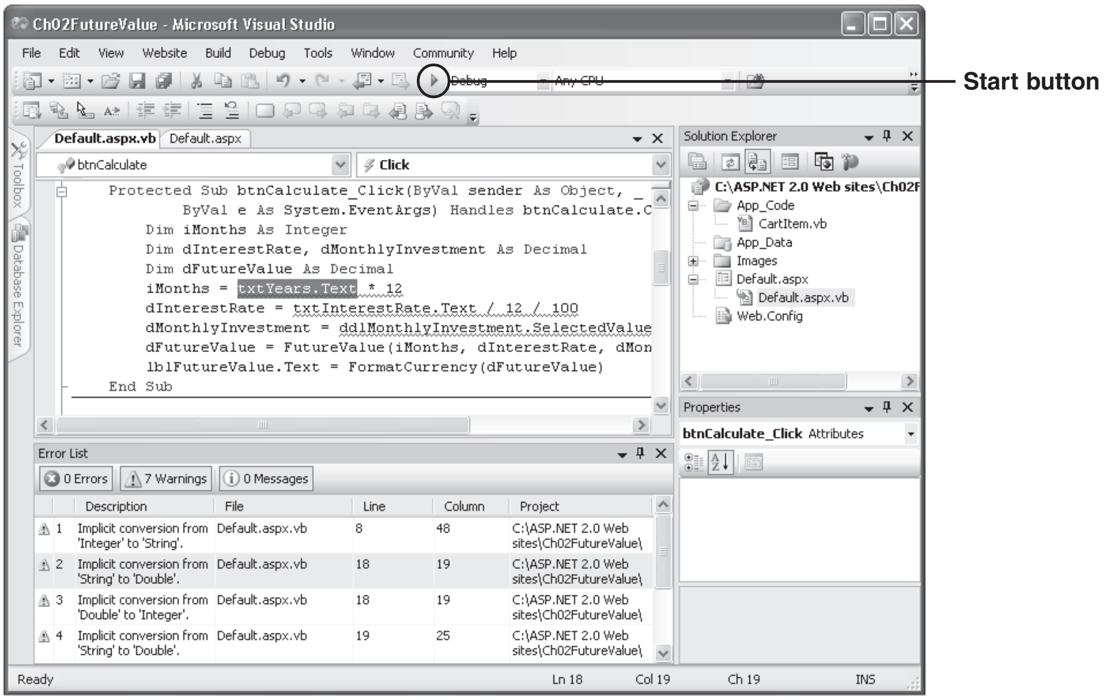
However, if any errors are detected as part of the compilation, Visual Studio opens the Error List window and displays the errors. These can consist of errors that have to be corrected as well as warning messages. In this figure, all of the errors have been corrected, but 7 warning messages are displayed in the Error List window.

To fix an error, you can double-click on it in the Error List window. This moves the cursor to the line of code that caused the error in the Code Editor. By moving from the Error List window to the Code Editor for all of the messages, you should be able to find the coding problems and fix them.

As you're testing an application with the development server, exceptions may occur. If an exception isn't handled by the application, ASP.NET switches to the Code Editor window and highlights the statement that caused the exception. In this case, you can end the application by clicking on the Stop Debugging button in the Debug toolbar or using the Debug➔Stop Debugging command. Then, you can fix the problem and test again.

In chapter 4, you'll learn all of the debugging skills that you'll need for more complex applications. For simple applications, though, you should be able to get by with just the skills you have right now.

An ASP.NET project with the shortcut menu for a web form displayed



```
    Protected Sub btnCalculate_Click(ByVal sender As Object, _
            ByVal e As System.EventArgs) Handles btnCalculate.C
        Dim iMonths As Integer
        Dim dInterestRate, dMonthlyInvestment As Decimal
        Dim dFutureValue As Decimal
        iMonths = txtYears.Text * 12
        dInterestRate = txtInterestRate.Text / 12 / 100
        dMonthlyInvestment = ddlMonthlyInvestment.SelectedValue
        dFutureValue = FutureValue(iMonths, dInterestRate, dMon
        lblFutureValue.Text = FormatCurrency(dFutureValue)
    End Sub
```

	Description	File	Line	Column	Project
1	Implicit conversion from 'Integer' to 'String'.	Default.aspx.vb	8	48	C:\ASP.NET 2.0 Web sites\Ch02FutureValue\
2	Implicit conversion from 'String' to 'Double'.	Default.aspx.vb	18	19	C:\ASP.NET 2.0 Web sites\Ch02FutureValue\
3	Implicit conversion from 'Double' to 'Integer'.	Default.aspx.vb	18	19	C:\ASP.NET 2.0 Web sites\Ch02FutureValue\
4	Implicit conversion from 'String' to 'Double'.	Default.aspx.vb	19	25	C:\ASP.NET 2.0 Web sites\Ch02FutureValue\

How to run an application

- Click on the Start button in the Standard toolbar or press F5. Then, the project is compiled and the starting page is displayed in your default browser.
- The first time you run an ASP.NET application, a dialog box will appear asking whether you want to modify the web.config file to enable debugging. Click the OK button to proceed.

How to stop an application

- Click the Close button in the upper right corner of the browser. Or, if an exception occurs, click the Stop Debugging button in the Debug toolbar or press Shift+F5.

How to fix build errors

- If any errors are detected when the project is compiled, an Error List window is opened and a list of errors is displayed along with information about each error. To display the source code that caused an error, double-click on the error in the Error List window.
- After you've fixed all of the errors, run the application again, and repeat this process if necessary. Note, however, that you don't have to fix the warnings.

Figure 2-17 How to run a web site with the built-in development server

How to review the HTML that's sent to the browser

To view the HTML for a page that's displayed in a browser, you can use the Source command in your browser's View menu. To illustrate, figure 2-18 presents the HTML that's sent back to the browser after I selected a new value from the drop-down list, entered new values into the text boxes, and clicked the Calculate button. Although you'll rarely need to view this code, it does give you a better idea of what's going on behind the scenes.

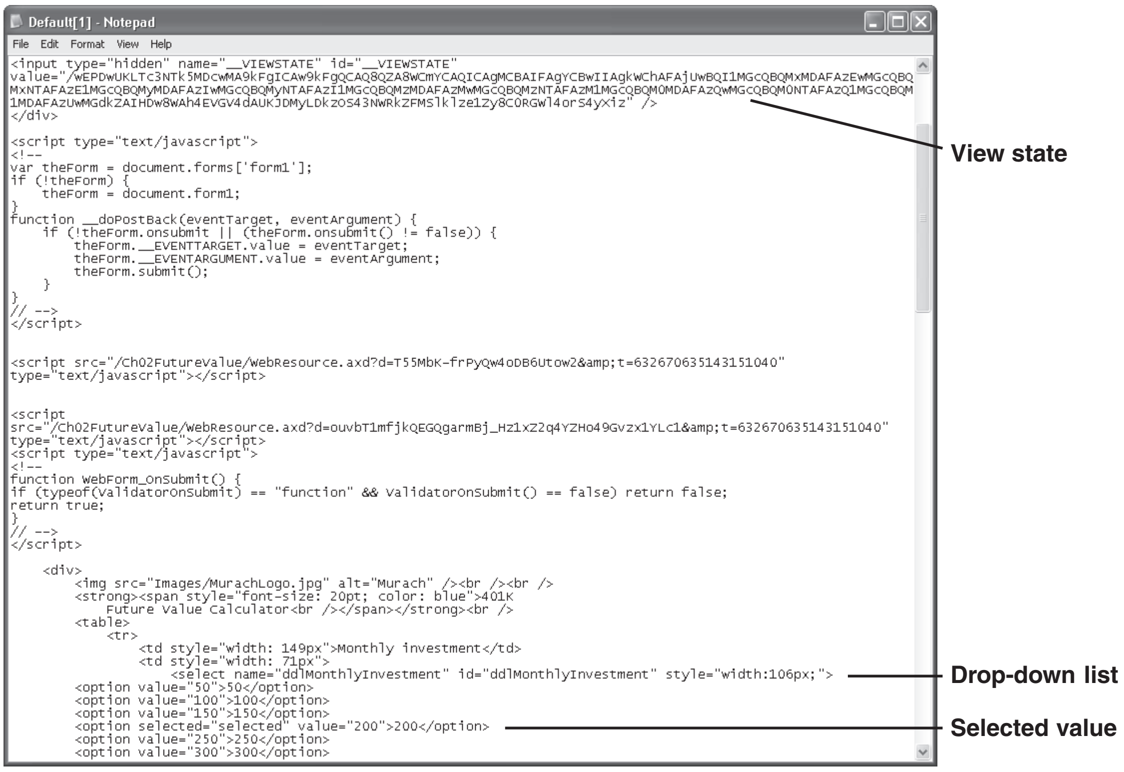
First, you'll see that this code doesn't include any asp tags. That's because these tags are rendered to standard HTML so the controls they represent can be displayed in the browser. For instance, the asp tag for the drop-down list in the first row of the table has been converted to an HTML select tag.

Second, you can see that the view state data is stored in a hidden input field named _ViewState. Here, the value of this field is encrypted so you can't read it. Because the data in view state is passed to and from the browser automatically, you don't have to handle the passing of this data in your code.

Third, you can see that the data that I selected from the drop-down list is included in the HTML. Although you can't see it, the data that was entered into the text boxes is included as well. This illustrates that you don't need view state to save the information that's entered by the user. Instead, view state is used to maintain the state of properties that have been set by code. For example, it's used to maintain the values that are loaded into the drop-down list the first time the user requests the form.

Keep in mind that this HTML is generated automatically by ASP.NET, so you don't have to worry about it. You just develop the application by using Visual Studio in the way I've just described, and the rest of the work is done for you.

The HTML for the Future Value form after a post back



```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUKLTc3NTk5MDcwMA9kFgICAw9kFgQCAQ8QZA8WCmYCAQICAgMCBAIFAgYCBwIIAgkWChAFAjUwBQI1MGcQBQMxMDAFAzEwMGcQBQ
MxNTAFAzE1MGcQBQMyMDAFAzIwMGcQBQMyNTAFAzI1MGcQBQMzMDAFAzMwMGcQBQMzNTAFAzM1MGcQBQM0MDAFAzQwMGcQBQM0NTAFAzQ1MGcQBQM
1MDAFAzUwMGdkZAIHDw8WAh4EVGV4dAUKJDMyLDkzOS43NWRkZFMSlklze1Zy8C0RGWl4orS4yXiz" />
</div>

<script type="text/javascript">
<!--
var theForm = document.forms['form1'];
if (!theForm) {
    theForm = document.form1;
}
function __doPostBack(eventTarget, eventArgument) {
    if (!theForm.onsubmit || (theForm.onsubmit() != false)) {
        theForm.__EVENTTARGET.value = eventTarget;
        theForm.__EVENTARGUMENT.value = eventArgument;
        theForm.submit();
    }
}
// -->
</script>


<script src="/Ch02FutureValue/WebResource.axd?d=T55MbK-frPyQw4oDB6Utow2&amp;t=632670635143151040"
type="text/javascript"></script>


<script
src="/Ch02FutureValue/WebResource.axd?d=ouvbT1mfjkQEGQgarmBj_Hz1xZ2q4YZHo49Gvzx1YLc1&amp;t=632670635143151040"
type="text/javascript"></script>
<script type="text/javascript">
<!--
function WebForm_OnSubmit() {
if (typeof(ValidatorOnSubmit) == "function" && ValidatorOnSubmit() == false) return false;
return true;
}
// -->
</script>

    <div>
        <img src="Images/MurachLogo.jpg" alt="Murach" /><br /><br />
        <strong><span style="font-size: 20pt; color: blue">401K
            Future Value Calculator<br /></span></strong><br />
        <table>
            <tr>
                <td style="width: 149px">Monthly investment</td>
                <td style="width: 71px">
                    <select name="ddlMonthlyInvestment" id="ddlMonthlyInvestment" style="width:106px;">
        <option value="50">50</option>
        <option value="100">100</option>
        <option value="150">150</option>
        <option selected="selected" value="200">200</option>
        <option value="250">250</option>
        <option value="300">300</option>
```

Description

- To view the HTML for a page, use the View➔Source command in the browser's menu.
- The HTML that the browser receives consists entirely of standard HTML tags because all of the ASP.NET tags are converted to standard HTML when the page is rendered.
- View state data is stored in a hidden input field within the HTML. This data is encrypted so you can't read it.
- If the page contains validation controls and client scripting is enabled for those controls, the HTML for the page contains script to perform the validation on the client if the client supports DHTML.
- Values that the user enters into a page are returned to the browser as part of the HTML for the page.

Perspective

The purpose of this chapter has been to teach you the basic skills for creating one-page ASP.NET applications with Visual Studio. If you've already used Visual Studio and Visual Basic to develop Windows applications, you shouldn't have any trouble mastering these skills. You just need to get used to HTML and the properties and events for web server controls and validation controls.

Terms

web site
file-system web site
web project
project
Web Forms Designer
Toolbox
Solution Explorer
Properties window
web form
Source view
Design view
web server control
drop-down list
text box
label
button
HTML server control
server control
validation control
flow layout
smart tag menu
HTML Editor
attribute
aspx code
validator
required field validator
range validator
Dynamic HTML (DHTML)
Code Editor
event procedure
event handler
general procedure
Sub procedure
Function procedure
function

About the book's applications

You can download all of the applications that are presented in this book from our web site (www.murach.com). Then, you can run the applications, review all of their code, and experiment with them on your own system. For more information about downloading and running these applications, please read appendix A.

If you're new to ASP.NET web programming...

If you're new to ASP.NET web programming, we recommend that you practice what you've learned after you finish each chapter in the first section of this book. For instance, you can now use the techniques of chapter 2 to build a Future Value application of your own. To do that, you can page through the figures, use the techniques that are illustrated, and compare your application with the one that you've downloaded. To show you what we mean, exercise 2-1 guides you through the process of building the Future Value application. By the time you complete section 1, though, you should be ready to start building applications of your own.

Exercise 2-1 Build the Future Value application

This exercise demonstrates how you can practice what you've learned after you complete each of the chapters in the first section of this book.

Start, close, and open the application

1. Start a new file-system web site as shown in figure 2-1 named FutureValue in a folder on your own system like C:\Practice Web Sites.
2. Add a folder named Images to your project and add the Murach logo to it, as shown in figure 2-3. You can find the logo in the Images folder of the downloaded FutureValue application.
3. Close the web site using the technique in figure 2-4, and reopen it using one of the three techniques in that figure. Then, switch to Design view.

Use Design view to build the form

4 Drag the logo from the Images folder in the Solution Explorer to the top of the web form. Then, use the techniques in figure 2-6 to enter and format the text for the heading in figure 2-5.

5. Use the techniques in figure 2-7 to add and format a table that provides for the six rows shown in figure 2-5. Then, add the text shown in figure 2-5 to the first four rows in the first column of the table.
6. Use the techniques in figure 2-8 to add the drop-down list, text boxes, label, and buttons shown in figure 2-5 to the table. Then, adjust the size of the columns, rows, and controls, so the table looks the way it does in figure 2-5.
7. Use the techniques of figure 2-8 and the summary in figure 2-9 to set the properties of the controls so they look like the ones in figure 2-5.

Use Source view to modify the aspx code

8. Switch to Source view, and change the title of the form to Future Value using the technique of figure 2-10. In addition, add an Alt attribute with a value of "Murach" to the HTML image control.
9. Press F5 to run the application. When the dialog box asks whether you want to modify the web.config file to enable debugging, click the OK button. Now, test to see what works by clicking on the controls. Also, check to make sure that the web form looks the way it's supposed to when it's displayed in the default browser. (Note that the fifth row of the table is blank.)
10. To end the application, click the Close button in the upper right corner of the browser. Then, adjust the design of the form as necessary by using either Design view or Source view, and test it again.

Add the validation controls

11. Add the validation controls for the interest rate and years text boxes as shown in figure 2-12 and 2-13.
12. Press F5 to run the application. Then, test the field validators by leaving fields blank or entering invalid data. The validation will be done when the focus leaves a text box or when you click on a button.

13. To end the application, click the browser's Close button. Then, fix any problems and test again. If, for example, validation is done when you click the Clear button, you can fix that by setting its CausesValidation property to False.

Add the Visual Basic code and test as you go

14. Use one of the techniques in figure 2-14 to open the Code Editor for the form. Then, use the tab at the top of the window to switch to Design view.
15. Double-click on a blank portion of the form to switch to the Code Editor and start a procedure for the Load event. Next, write the code for this procedure, which should be like the code in figure 2-16. Then, press F5 to compile and test this procedure. If any errors are detected when the application is compiled, use the techniques in figure 2-17 to fix them.
16. Switch back to Design view, and double-click on the Clear button to start a procedure for the Click event of that button. Next, write the code for this procedure, which should be like the code in figure 2-16. Then, compile and test this procedure, and fix any errors.
17. Write the FutureValue function from scratch as shown in figure 2-16. After you enter the signature for the function, Visual Studio should add the End Function line that ends the function.
18. Switch back to Design view, and double-click on the Calculate button to start a procedure for the Click event of that button. Next, write the code for this procedure, which should use the FutureValue function and be like the code in figure 2-16. Then, compile and test, and fix any errors.

Do more testing and experimenting

19. Set the EnableViewState property of the drop-down list to False, and test the application to see what happens. When an exception occurs, click the Stop Debugging button in the Debugging toolbar. Then, reset the property.
20. Set the EnableClientScript property for the validators of the first text box to False so this validation will only be done on the server. Then, test the application to make sure that the validation still works.
21. Run the application again, and use the technique in figure 2-18 to review the HTML that's sent to the browser. When you're through, close the project