Thanks for downloading this chapter from *Murach’s ASP.NET 4.6 Web Programming with C# 2015*. We hope it will show you how easy it is to learn from any Murach book, with its paired-pages presentation, its “how-to” headings, its practical coding examples, and its clear, concise style.

To view the full table of contents for this book, you can go to our [website](#). From there, you can read more about this book, you can find out about any additional downloads that are available, and you can review our other books on related topics.

Thanks for your interest in our books!
What developers have said about previous editions

“This book enabled me to write custom websites for two customers, which is exactly what I wanted it to do. No previous book did such an excellent job of demonstrating how to handle security and database classes.”
Charles Profitt, Developer, New York

“I purchased several ASP.NET books but didn’t really learn it until I purchased this one.”
J.R., Enterprise Developer, Florida

“The author does a fantastic job of walking readers through the fundamentals, making the process feel like you’re sitting down with a skilled instructor who’s moving you from one example to the next.”
Muhammad Riaz, Baton Rouge Oracle User Group

“Another awesome book from Murach. Their format makes learning new material easier, and their code examples WORK.”
Posted at an online bookseller

“I especially appreciated the workaround insights for common hang-ups, like dealing with back-button refreshes. These gems alone will be worth the price of the book for many first-time .NET web authors.”
5-star review by Mike Riley, DevProConnections.com

“With this book, you will not have to rely on Google and the internet for your tutorials any longer.”
James Payne, www.aspfree.com

“Great book. I’m hard to please, and as a programmer and engineer, I was surprised by how informative this book was.”
Posted at an online bookseller
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 use Visual Studio to develop the Future Value application that you reviewed in the last chapter. If you’ve used Visual Studio to develop Windows applications, you’ll see that you develop ASP.NET web applications in much the same way. As a result, you should be able to move quickly through this chapter.

<table>
<thead>
<tr>
<th>How to work with ASP.NET web application projects ......32</th>
</tr>
</thead>
<tbody>
<tr>
<td>How to start a new web application.................................32</td>
</tr>
<tr>
<td>How to work with the web application templates...................34</td>
</tr>
<tr>
<td>How to add a web form to a web application..........................36</td>
</tr>
<tr>
<td>How to work with the Visual Studio IDE...............................38</td>
</tr>
<tr>
<td>How to add folders and files to a web application..................40</td>
</tr>
<tr>
<td>How to open and close a web application..............................42</td>
</tr>
<tr>
<td>How to use Visual Studio to build a web form....................44</td>
</tr>
<tr>
<td>How to enter the HTML for a web form.................................44</td>
</tr>
<tr>
<td>How to add a table to a form............................................46</td>
</tr>
<tr>
<td>How to add text to the cells of a table.................................46</td>
</tr>
<tr>
<td>How to add web server controls to a form.............................48</td>
</tr>
<tr>
<td>How to set the properties of the controls..............................48</td>
</tr>
<tr>
<td>Common properties for web server controls............................50</td>
</tr>
<tr>
<td>How to add validation controls to a form.............................52</td>
</tr>
<tr>
<td>An introduction to the validation controls............................52</td>
</tr>
<tr>
<td>How to use the required field validator...............................54</td>
</tr>
<tr>
<td>How to use the range validator..........................................54</td>
</tr>
<tr>
<td>How to work with unobtrusive validation.............................56</td>
</tr>
<tr>
<td>The aspx code for the Future Value form..............................58</td>
</tr>
<tr>
<td>How to add C# code to a form.............................................62</td>
</tr>
<tr>
<td>How to use the Code Editor.................................................62</td>
</tr>
<tr>
<td>How to use page and control events....................................64</td>
</tr>
<tr>
<td>The C# code for the Future Value form................................66</td>
</tr>
<tr>
<td>How to test a web application.............................................68</td>
</tr>
<tr>
<td>How to run a web application.............................................68</td>
</tr>
<tr>
<td>How to view the HTML that’s sent to the browser....................70</td>
</tr>
<tr>
<td>Perspective............................................................................72</td>
</tr>
</tbody>
</table>
How to work with ASP.NET web application projects

This chapter starts by presenting some basic skills for working with ASP.NET web application projects. Once you're comfortable with those skills, you'll be ready to learn how to build your first ASP.NET web application.

How to start a new web application

To start a new web application, you use the New Project dialog box shown in figure 2-1. This dialog box lets you select the type of project you want to create by choosing a template. To create an ASP.NET web application, for example, you choose the ASP.NET Web Application template.

The New Project dialog box also lets you specify the name for the project, and it lets you identify the folder in which it will be stored. By default, projects are stored in the Visual Studio 2015\Projects folder under the My Documents folder, but you can change the default folder if you want to. To do that, display the Options dialog box (Tools\Options). Then, expand the Projects and Solutions group, select the General category, and enter the folder you want to use in the Projects Location text box.

If you want to change the location that's shown in the New Project dialog box, you can click the Browse button to select a different location; display the Location drop-down list to select a location you've used recently; or type a path directly. If you specify a path that doesn't exist, Visual Studio will create the necessary folders for you.

When you click the OK button, Visual Studio automatically creates a new folder for the project using the project name you specify. In the dialog box in this figure, for example, Ch02FutureValue is the project name and C:\aspnet46_cs is the location. By default, Visual Studio also creates a new folder for the solution using the same name as the project. As a result, Visual Studio will create one folder for the solution, and a subfolder for the project. It also creates an application on the IIS Express web server.

When you create a new web application, Visual Studio 2015 also lets you choose a target framework. The target framework determines the features that are available to an application. By default, the target framework for a web application is .NET Framework 4.5.2, even though the most recent version of the .NET Framework is 4.6. That's because version 4.5.2 has been widely deployed, while version 4.6 has not as of this writing. Fortunately, everything in this book will work with the 4.5.2 version of the framework.
The New Project dialog box

How to start a new web application

- Use the File  New  Project command or click on the “New Project…” link on the Start Page to open the New Project dialog box.
- Choose Web from the Installed  Templates  Visual C# category, and choose the ASP.NET Web Application template.
- Enter a name for the project, which will enter the same name for the solution. Then, enter the location (folder) for the project (and solution).
- Click the OK button to display the New ASP.NET Project dialog box shown in the next figure.

Description

- If the Create Directory For Solution box is checked, Visual Studio creates a folder for the solution and a subfolder for the project. Otherwise, these files are stored in the same folder.
- In addition to the solution folder and file that are created in the location you specify, Visual Studio creates a web application on the IIS Express web server.
- By default, new web applications use .NET Framework 4.5.2, even though the most recent version is 4.6. If you want to change the version, you can use the drop-down list at the top of the dialog box.
How to work with the web application templates

After you complete the New Project dialog box described in the last figure, the New ASP.NET Project dialog box shown in figure 2-2 is displayed. This dialog box lets you choose the specific template for your web application. The template you choose determines the files and folders that Visual Studio adds to the project when it creates your web application. The four templates you’ll work with in this book for ASP.NET 4.5.2 and ASP.NET 4.6 are described in the table in this figure.

The Empty template creates just two configuration files: the Web.config and packages.config files that you learned about in chapter 1. However, if you check the Web Forms checkbox in the Add Folders And Core References For section, App_Data and Models folders will also be added, along with a Global.asax file. Most of the applications in this book use the Empty template, often with the core references and folders for Web Forms added.

The Web Forms template, sometimes called the default template, adds numerous files and folders that provide various functionality. For example, this template includes an authentication system, a URL routing system, and a Bootstrap theme. If you ever want to use any of these features, this template is a good source of sample code. You’ll see some of the code from this template in later chapters of this book.

The MVC template sets up the directory and routing structure used by ASP.NET MVC. You’ll learn how to use this template in chapter 25 when you’re introduced to MVC. Note that this template uses MVC5, which is the current release of MVC.

The Web API template sets up the directory and routing structure used to create a REST based web service. You’ll learn how to create web services using this template in chapter 24.

Although you won’t learn how to use any of the other templates in this book, you might want to experiment with them. For example, the Single Page Application template uses HTML5, jQuery, KnockoutJS, and Bootstrap to create a single page that communicates with the server and updates portions of itself with JavaScript. That way, after the page loads, it never does a full round trip and page refresh, which makes it seem more like a desktop application. This kind of application, known as a SPA, is widely used for mobile applications.

The Azure templates use Microsoft’s Azure cloud platform and Web API to create backend services for web and mobile applications. And the ASP.NET 5 template lets you get the ASP.NET 5 Release Candidate (RC). This version of ASP.NET, also called ASP.NET Core 1, is still under development. You can use it to create MVC-based web applications, but it won’t include Web Forms.

Before you go on, you should notice the Host In The Cloud check box near the lower right of the New ASP.NET Project dialog box. Before you click the OK button to create a project, you should be sure this check box isn’t selected since you’ll be storing your applications on a local drive and not in the Cloud.
The New ASP.NET Project dialog box

![The New ASP.NET Project dialog box](image)

The ASP.NET 4.5.2 / ASP.NET 4.6 templates you’ll use in this book

<table>
<thead>
<tr>
<th>Template</th>
<th>Contains…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty</td>
<td>Two configuration files but no other files or folders.</td>
</tr>
<tr>
<td>Empty/Web Forms option</td>
<td>Two configuration files, empty App_Data and Models folders, and a Global.asax file with an empty Application_Start method.</td>
</tr>
<tr>
<td>Web Forms</td>
<td>Numerous files and folders for various functionality (see chapter 20).</td>
</tr>
<tr>
<td>MVC</td>
<td>Basic files and folders for an MVC application (see chapter 25).</td>
</tr>
<tr>
<td>Web API</td>
<td>Basic files and folders for a Web API web service (see chapter 24).</td>
</tr>
</tbody>
</table>

Description

- The first group of templates are labeled ASP.NET 4.5.2 or ASP.NET 4.6, depending on which version of the .NET Framework you chose in the New Project dialog box.
- ASP.NET provides several project templates. This book uses the Empty template, although often with the core references for Web Forms added.
- The Web Forms template, sometimes called the default template, includes functionality for authentication, URL routing, and Bootstrap. You’ll see the Web Forms, MVC, and Web API templates later in this book.
- The ASP.NET 5 template lets you get the ASP.NET 5 release candidate, which isn’t addressed in this book.
How to add a web form to a web application

If you start a web application from the ASP.NET Empty template, you’ll need to add a web form to the project. To do that, you can use the Add New Item dialog box shown in figure 2-3. From this dialog box, you select the Web Form template. Then, you enter the name you want to use for the new form and click the Add button to add it to your web application.

After you click Add, three files are added to your project. For instance, if you change the name in the dialog in this figure to Default, then files named Default.aspx, Default.aspx.cs, and Default.aspx.designer.cs will be added to the project. The Default.aspx file contains the HTML and ASP code that defines the form, and the Default.aspx.cs file contains the C# code that determines how the form works. The Default.aspx.designer.cs file contains generated code used by Visual Studio, and you shouldn’t need to edit this file. After these files are added to the project, Visual Studio displays the aspx file for the web form.

Another way to add a web form is to use the Add  Web Form command in the shortcut menu for the project. When you choose this command, the dialog box that’s displayed only lets you specify the name for the form.

To add an existing web form from another web application to your web application, you can use the third procedure in this figure. You might want to do that if you need to create a form that’s similar to a form in another web application. When you add the aspx file for a form, the code-behind and designer files are added too. Then, you can modify the aspx and C# code so the form works the way you want it to in your new web application.

When you add an existing web form in this way, you should know that the form will retain the name of the project it came from as the namespace name. Although the application will run correctly, the different namespace name might be confusing. Because of that, you should consider changing the name in the namespace declaration for the code-behind and in the Inherits attribute of the Page directive for the aspx file so they match the name of the current project.
The Add New Item dialog box for adding a new web form

![Add New Item dialog box](image)

Two ways to open the Add New Item dialog box
- Right-click the project in the Solution Explorer, and choose Add ➔ Add New Item from the shortcut menu.
- Click on the project in the Solution Explorer to select it, and then choose the Project ➔ Add New Item command.

How to add a new web form to a project
- In the Add New Item dialog box, select the Web Form template, enter a name for the form, and click the Add button.
- Choose Add ➔ Web Form from the shortcut menu for the project. Then, enter a name for the form in the dialog box that’s displayed and click the OK button.

How to add an existing web form to a project
- In the Solution Explorer, right-click the project and choose Add ➔ Add Existing Item. Then, locate the form you want to add, select its .aspx file, and click the Add button.

Description
- If there’s a web form in another application that’s like what you need, you can copy that form into your application. That copies the aspx, code-behind, and designer files. Then, you can modify the aspx code and C# code so the form works the way you want it to.
- The drop-down list above the list of templates lets you change their order. The two buttons let you choose whether the templates are displayed as small or medium icons.
How to work with the Visual Studio IDE

Figure 2-4 shows the Visual Studio IDE after a form named Default has been added to the Future Value web application. 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.

For instance, the Solution Explorer shows the folders and files of the web application. In the example in this figure, the Solution Explorer shows the Images folder with the logo banner image, the collapsed web form, and the two configuration files. To expand the web form and see the code-behind and designer files, you click on the arrowhead to the left of the web form.

To design a web form, you use the Web Forms Designer that’s in the center of Visual Studio. When you add a new web form to a web application, this Designer is displayed in Source view, which shows the starting HTML code for the form. However, you’ll do much of the design in Design view, which you can switch to by clicking on the Design button at the bottom of the Designer. You can also work in Split view, which includes both Source view and Design view.

As you work in the Designer, you’ll notice that different toolbars are enabled depending on what view you’re working in. In Source view, for example, the Standard and HTML Source Editing toolbars are enabled. In Design view, the Standard and Formatting toolbars are enabled. This is typical of the way Visual Studio works.

As you build a web application, you can 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 will replace the Designer in the center of the screen. Then, you can write the C# 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. You’ll learn more about that in a moment.

The Solution Explorer also contains Properties and References items. When you double-click on Properties, a tab displaying the properties of the application opens in the center of the screen. You can also see these properties in code by expanding the Properties item and then double-clicking on the AssemblyInfo.cs file. When you expand the References item, you’ll see a list of the assemblies for the namespaces that the application can use. These namespaces contain the classes that the project requires, and most of them are included when the project is created.

As you work with Visual Studio, you’ll see that it often 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.
How to work with views and windows

- To change the Web Forms Designer from one view to another, click on the Design, Split, or Source button at the bottom of the Designer window.

- To hide a window, click on its Auto Hide button, which is a pin icon. Then, the window is shown as a tab at the side of the screen. To display the window again, move the mouse pointer over the tab or click on it. To restore the window, display it and click on the Auto Hide button.

- To size a window, place the mouse pointer over one of its boundaries and drag it.

- To close a window, click on the close button in its upper right corner. To redisplay it, select it from the View menu.

- To add line numbers to a window, use the Tools→Options command to display the Options dialog box, then expand Text Editor and All Languages, click on General, check the Line Numbers checkbox, and click OK.

Description

- The primary window for designing web forms with Visual Studio is the *Web Forms Designer*, or just *Designer*, that’s in the middle of the IDE.

- The three supporting windows are the *Toolbox*, the *Solution Explorer*, and the *Properties* window.

- Visual Studio often provides several different ways to do the same task. In this book, we’ll try to show you the techniques that work the best.
How to add folders and files to a web application

Right after you start a new web application, it makes sense to add any other folders or 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 2-5. As you can see, this menu provides a New Folder command as well as an 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 Add and then the New Folder command, and entered the name for the folder. Then, I added an image file named MurachLogo.jpg to the Images folder. To do that, I right-clicked on the folder, chose Add and then Existing Item, and selected the file from the dialog box that was displayed.

Those are the only other folders and files that are needed for the Future Value application, but often you’ll need others. For instance, the application in chapter 4 requires three existing business classes, a database, and a number of image files. As you’ll see in that chapter, you store the database files in a special ASP.NET folder named App_Data, which you can add using the Add ASP.NET Folder menu shown in this figure.
The Future Value project as a new folder is being added

How to add a folder to a web application

- 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 Add → New Folder. Then, type a name for the folder in the dialog box that’s displayed and press Enter.
- To add a special ASP.NET folder, right-click on the project in the Solution Explorer and choose Add → Add ASP.NET Folder. Then, select the folder from the list that’s displayed.

How to add an existing item to a web application

- In the Solution Explorer, right-click on the project or folder that you want to add an existing item to. Then, select Add → Existing Item and respond to the 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 application.
How to open and close a web application

Visual Studio provides several ways to open an existing web application. One way is to display the Open Project dialog box shown in figure 2-6. You can use this dialog box to locate the folder that contains the web application you want to open. Then, you can double-click this folder to reveal the solution file, and you can double-click this file to open the solution for the application.

If you’ve opened the web application recently, you can open it again without using the Open Project dialog box. To do that, you can click on the project name in the Recent list of the Start page, also shown in this figure. Or, you can select a project from the list that’s displayed when you choose File → Recent Projects and Solutions.

By default, the Start Page is displayed each time you open Visual Studio, and it’s hidden when you open an application. If you want to display the Start Page while an application is open, you can do that using the command shown in this figure. You can also keep the Start Page open at all times by selecting the appropriate check box on this page. Finally, you can pin a solution so it’s always included in the Recent list as described in this figure.

To close a solution, you use the Close Solution command. After you close a solution for the first time, you’ll be able to find it in the Recent list on the Start Page and also in the list of projects that you see when you use the Recent Projects and Solutions command.
The Visual Studio Start Page and the Open Project dialog box

How to use the Open Project dialog box to open a web application

- Use the File ➔ Open ➔ Project/Solution command or click the “Open Project…” link in the Start section of the Start page to display this dialog box.
- Locate the folder that contains the web application and double-click it to display the solution file. Then, double-click the solution file or select it and click the Open button.

Other ways to open a web application

- Click on the project name in the Recent list of the Visual Studio Start Page.
- Click on the project name in the File ➔ Recent Projects and Solutions menu.

How to close a web application

- Use the File ➔ Close Solution command.

How to work with the Start Page

- To open the Start Page, select View ➔ Start Page.
- To keep the Start Page open after you open an application, select the appropriate check box in the lower left corner of the Start Page.
- To pin a project so it always appears in the Recent list, hover your mouse over the solution name and click on the horizontal pin icon that appears. To unpin a solution, click its pin icon again.

Description

- You use a web application’s solution file to open it. This file has an .sln file extension.
How to use Visual Studio to build a web form

Now that you know how to start, open, and close a web application, you’re ready to learn how to build a web page with HTML, web server controls, and validation controls. If any of this seems confusing as you read about it, the exercise at the end of this chapter will show you that all of the skills are quite manageable.

How to enter the HTML for a web form

Figure 2-7 presents the primary ways to add HTML to a web form. For many HTML elements, the easiest way to add them is to type the HTML for the elements directly into the source code, taking full advantage of IntelliSense. In this figure, for example, you can see how IntelliSense provides a snippet for an h1 element. Just remember to press the Tab key twice to insert both the opening and closing tags for an element.

For some elements, though, it’s better to insert a snippet using the second technique in this figure. To do that, you move the insertion point to where you want the snippet, right-click to display a menu, select Insert Snippet, select HTML, and select the HTML element that you want to insert. If, for example, you insert the snippet for an img element, the HTML includes the src and alt attributes.

This figure also shows how you can add an img element to the HTML by dragging the image from the Solution Explorer and dropping it wherever you want it. If you drop it in Design view, the Accessibility Properties dialog box is displayed. And that makes it easy to enter the Alternate Text property, which gets converted to an alt attribute in the HTML.

Whether or not you use the Accessibility Properties dialog box, the alt attribute should always be coded for an img element because it improves accessibility. Specifically, this attribute is used by screen readers to describe an image for the visually impaired. If an image is used for decorative purposes only, the value of this attribute should be an empty string (""").

In contrast, the Long Description property in the Accessibility Properties dialog box gets converted to the longdesc attribute. Since that attribute isn’t supported by HTML5 or any modern browser, though, you should leave it blank.

Usually, you’ll want to make a few adjustments and additions to the HTML right after the form is added to the application. For instance, you’ll want to enter a title for the form in the title element that’s in the head section. That’s the title that’s displayed in the title bar or tab of the browser when the form is run. You’ll also want to add an h1 element to the form that describes what the page does.

After making the HTML entries, you can use either Source view or Design view to add web server controls to the form. If you work in Design view, though, you’ll want to switch back to Source view from time to time. That way, you can review the source code that has been added, make sure the code is in the right location, and make adjustments to the source code.
The Future Value form in Split view after an img element has been added

How to add HTML elements to a form

- Enter the code for the element in Source view. As you work, Visual Studio’s IntelliSense will help you enter snippets, tags, attributes, etc. To add a snippet, press the Tab key twice.
- To insert a snippet for an HTML element in Source view without using IntelliSense, move the insertion point to where you want the snippet. Then, right-click, select Insert Snippet, select HTML, and select the element that you want inserted.

Two ways to add an img element to a form

- Insert a snippet for the element. That includes the src and alt attributes, but you have to add the values.
- Drag the image from the Solution Explorer to the Designer. This inserts an img element with a valid src attribute. In Design view, the Accessibility Properties dialog box is also displayed.

How to add and remove comments

- To add a comment at the insertion point, click the Comment button in the HTML Source Editing toolbar, or press Ctrl+K and then Ctrl+C. If you select lines of code before you do this, the lines will be commented out.
- To remove a comment, move the insertion point into it and click the Uncomment button, or press Ctrl+K and then Ctrl+U. If you select lines of code that have been commented out before you do this, they will be uncommented.

How to synchronize the views when you’re working in Split view

- Save the file or click on the message that’s displayed between the views.
How to add a table to a form

By default, forms use flow layout. This means that 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.

Usually, though, you will want more control than flow layout provides. One way to get that control is to use a table, which you’ll learn about now. Another way is to use CSS, which you’ll learn about in the next chapter.

Figure 2-8 shows how to add a table to a form in Design view. 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 Table 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.

Note that when you make some of these changes, Visual Studio adds classes to the HTML elements as well as a style element in the head section of the form that contains the rule sets for the classes. You’ll see this when you review the aspx code for the Future Value form.

How to add text to the cells of a table

In figure 2-8, 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 in the Formatting toolbar.
The Future Value form with a table that has been inserted into it

How to add a table to a form

• Use the Table→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 one of its handles.
• To select rows, columns, or cells, drag the mouse over them or hold the Ctrl key down as you click on the cells. To add, delete, size, or merge selected rows or columns, use the commands in the Table menu or the shortcut menu.

How to add text to a table and format it

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

Description

• To control the alignment of the text and controls on a web form, you can use tables.
• Some of the formatting that you apply to tables, rows, and columns is saved in CSS rule sets in a style element in the head section of the HTML. You’ll learn more about CSS in the next chapter.
How to add web server controls to a form

Figure 2-9 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 to 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, and whether you’re in Source view or Design view.

Here again, you can add a web server control to a form by inserting a snippet. But this time, after you select the Insert Snippet command, you select ASP.NET and then the server control that you want to add.

Once you’ve added the controls to the form, you can resize them in Design view 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. But 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.

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. To set those properties, you can work in the Properties window as shown in this figure. To display the properties for a control, just click on it in Design or Source view.

In the Properties window, you select a property by clicking it. Then, a brief description of that property is displayed 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 (…) will appear when you click on a property. Then, you can click the button to display a dialog box that sets the property.

Some properties are displayed in groups. In that case, a + symbol appears next to the group name. To expand the properties in the group, just click the + symbol, which then changes to a – symbol.

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 can see 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.

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. Because smart tag menus help you set common properties, they’re displayed automatically when you drag a control to a form in Design view. Later, you can display the smart tag menu of a control by hovering the mouse pointer over it until its smart tag appears and then clicking on that tag.

As you work with properties, you’ll find that many are set the way you want by default. In addition, some properties such as Height and Width are set 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 Future Value form after six server controls have been added to it

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 in either Source or Design view, and double-click on the control in the Toolbox to place it there.

- To insert a snippet for a server control in Source view, move the insertion point to where you want the snippet. Then, right-click, select Insert Snippet, select ASP.NET, and select the control that you want inserted.

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.

- To change the Height and Width properties, drag one of the handles on a control. This also changes the Height and Width in the Properties window.

- To sort the properties in the Properties window by category or alphabetically, click on one of the buttons at the top of the window. To expand or collapse the list of properties in a group, click on the + or – symbol for the group.

- To display a smart tag menu for a control in Design view, select the control and click the Smart Tag icon on the right of the control. In Source view, click in the aspx code for the control and hover over the line that appears under the <asp> tag to reveal the smart tag icon and then click on it.

Description

- Many web server controls have smart tag menus that provide options for performing common tasks and setting common properties.
Common properties for web server controls

The first table in figure 2-10 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 identify a control that you need to refer to in your C# code, and you can use the Text property to set what’s displayed in or on the control.

In contrast, the AutoPostBack, CausesValidation, EnableViewState, and Runat properties are unique to web server controls. As you should already know, the Runat property just indicates that the control must be processed by the web server. The other three properties are more interesting.

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 drop-down lists, check boxes, and radio buttons. Also note that this property isn’t available with button controls. That’s because button controls always either post a page back to the server or display another page.

The CausesValidation property is available for button controls and determines whether the validation controls are activated when the user clicks the button. This lets the browser check for valid data before the page is posted back to the server. You’ll learn more about validation controls in a moment.

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. For instance, you can use the Items collection to add, insert, and remove ListItem objects, and you can use the SelectedValue property to retrieve the value of the currently selected item. Although you can set these properties at design time, they are often set by the C# code in the code-behind file. You’ll learn more about these properties when you review the code-behind file for the Future Value form.
Common web server control properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoPostBack</td>
<td>Determines whether the page is posted back to the server when the value of the control changes. Available with controls like check boxes, text boxes, and lists. The default value is False.</td>
</tr>
<tr>
<td>CausesValidation</td>
<td>Determines whether the validation specified by the validation controls is done when a button control is clicked. The default value is True.</td>
</tr>
<tr>
<td>EnableViewState</td>
<td>Determines whether the control maintains its view state between HTTP requests. The default value is True.</td>
</tr>
<tr>
<td>Enabled</td>
<td>Determines whether the control is functional. The default value is True.</td>
</tr>
<tr>
<td>Height</td>
<td>The height of the control.</td>
</tr>
<tr>
<td>ID</td>
<td>The name that’s used to refer to the control.</td>
</tr>
<tr>
<td>Runat</td>
<td>Indicates that the control will be processed on the server by ASP.NET.</td>
</tr>
<tr>
<td>TabIndex</td>
<td>Determines the order in which the controls on the form receive the focus when the Tab key is pressed. A value of -1 means the control isn’t included in the tab order.</td>
</tr>
<tr>
<td>Text</td>
<td>The text that’s displayed in the control.</td>
</tr>
<tr>
<td>ToolTip</td>
<td>The text that’s displayed when the user hovers the mouse over the control.</td>
</tr>
<tr>
<td>Visible</td>
<td>Determines whether a control is displayed or hidden.</td>
</tr>
<tr>
<td>Width</td>
<td>The width of the control.</td>
</tr>
</tbody>
</table>

Common properties of drop-down list and list box controls

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>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 drop-down list or list box.</td>
</tr>
<tr>
<td>SelectedItem</td>
<td>The ListItem object for the currently selected item.</td>
</tr>
<tr>
<td>SelectedIndex</td>
<td>The index of the currently selected item starting from zero. If no item is selected in a list box, the value of this property is -1.</td>
</tr>
<tr>
<td>SelectedValue</td>
<td>The value of the currently selected item.</td>
</tr>
</tbody>
</table>

Note
- When buttons are clicked, they always post back to the server or display other pages. That’s why they don’t have AutoPostBack properties.
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 them. Then, in chapter 7, you can learn how to use all of these controls.

An introduction to the validation controls

Figure 2-11 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.

To add a validation control to a web form, you can use the same techniques that you use to add a server control. Before you can access the validation controls in the Toolbox, though, you need to open the Validation group by clicking on the arrowhead to its left. You can also add a validation control to a form by inserting a snippet.

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 don’t show when the form is displayed, the messages in their ErrorMessage properties are displayed if errors are 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 have JavaScript enabled. Because most browsers enable it, validation is usually done on the client. That way, a round trip to the server isn’t required to display error messages if invalid data is detected. 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 whether the test was successful. Then, after all the validators are tested, it sets the IsValid property of the page to indicate whether all the tests were valid. This is the property that’s usually tested by the C# code when the page is posted to the server. You’ll see how this works when you review the code-behind file for this form.
Chapter 2  How to develop a one-page web application

The validation controls on the Future Value form

![Image of a one-page web application interface with validation controls]

How to add a validator to a web form

- In either Design or Source view, move the insertion point to where you want the validator and double-click on the validator in the Validation group in the Toolbox. Or, drag the validator from the Toolbox to where you want it.
- In Source view, right-click, select Insert Snippet, select ASP.NET, and select the validation control that you want to insert.

How to set the properties for a validation control

- In either Design or Source view, use the Properties window.
- In Source view, enter the properties for the validator with help from IntelliSense.

Description

- You can use validation controls to test user entries and produce error messages. The validation is typically done when the focus leaves the control that’s being validated and also when the user clicks on a button that has its CausesValidation property set to True.
- Each validation control is associated with a specific server control, but you can associate more than one validation control with the same server control.
- If the user’s browser has JavaScript enabled, the validation controls work by running JavaScript in the browser. Then, if the validation fails, the page isn’t posted back to the server, which saves a round trip. If the browser doesn’t have JavaScript enabled, the validation is done on the server.
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 2-12. 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 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. The error message is also displayed if the user clears the value in the input control and then moves the focus to another control.

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. However, if you use a validation summary control as explained in chapter 7, 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 a required field validator that validates the text box with txtInterestRate as its ID. Here, the ForeColor property of the required field validator is set to “Red” so the error message will be displayed in that color. In the next chapter, you’ll learn how to use CSS to get the same result.

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 this figure, plus the properties in the second table. In particular, you set the minimum and maximum values for an input value.

The aspx code in this figure also shows how the properties are set for the range validator for the text box with txtInterestRate as its ID. For this to work correctly, you must set the Type property to the type of data that you’re testing. Because the interest rate entry can have decimal positions, for example, the Type property for its range validator is set to Double. In contrast, because a year entry should be a whole number, the Type property for its range validator should be set to Integer.
### Common validation control properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ControlToValidate</td>
<td>The ID of the control to be validated.</td>
</tr>
<tr>
<td>Display</td>
<td>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.</td>
</tr>
<tr>
<td>ErrorMessage</td>
<td>The message that’s displayed in the validation control when the validation fails.</td>
</tr>
</tbody>
</table>

#### Additional properties of a range validator

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaximumValue</td>
<td>The maximum value that the control can contain.</td>
</tr>
<tr>
<td>MinimumValue</td>
<td>The minimum value that the control can contain.</td>
</tr>
<tr>
<td>Type</td>
<td>The data type to use for range checking (String, Integer, Double, Date, or Currency).</td>
</tr>
</tbody>
</table>

#### The aspx code for a RequiredFieldValidator control

```xml
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
    ControlToValidate="txtInterestRate" Display="Dynamic"
    ErrorMessage="Interest rate is required." ForeColor="Red">
</asp:RequiredFieldValidator>
```

#### The aspx code for a RangeValidator control

```xml
<asp:RangeValidator ID="RangeValidator1" runat="server"
    ControlToValidate="txtInterestRate" Display="Dynamic"
    ErrorMessage="Interest rate must range from 1 to 20." ForeColor="Red"
    MaximumValue="20" MinimumValue="1" Type="Double" />
</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 a 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.
How to work with unobtrusive validation

ASP.NET 4.5 and later have a feature called *unobtrusive validation* that you need to be aware of. This feature controls how the client-side validation of the validation controls is done. Figure 2-13 shows the two settings for unobtrusive validation that a web application can have.

A setting of Webforms means that unobtrusive validation is enabled and ASP.NET will use jQuery for validation. *jQuery* is a JavaScript library that provides for cross-browser compatibility and reduces the amount of JavaScript that an ASP.NET application requires. A setting of None means that unobtrusive validation is disabled and ASP.NET will do the validation the way it was done in versions prior to 4.5, which is to use script elements within the HTML to supply the JavaScript for the validation.

The benefit of using unobtrusive validation is that it reduces the amount of JavaScript that has to be generated. That’s why unobtrusive validation is enabled by default.

The problem with this is that if you start a web application from the Empty template, unobtrusive validation is enabled but the jQuery library and configuration needed to use it aren’t there. This means that if you try to use a validation control, you’ll get an error. There are two ways to fix this.

One fix is to add the components that are required by unobtrusive validation. To do that, you can use NuGet, which is a Visual Studio feature that makes it easy to add third-party and open-source packages to an application. You’ll learn more about NuGet later, but this figure shows how to use NuGet to install the package that provides everything you need to make unobtrusive validation work in your web application.

The other fix is to disable unobtrusive validation. To do that for one page, you can set the UnobtrusiveValidationMode property to None in the Load event handler for the page, as shown in this figure. To do that for all pages in a web application, you can add an appSettings element like the one in this figure to the Web.config file.
Two values for the `UnobtrusiveValidationMode` setting

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Webforms</td>
<td>Uses the jQuery library for the validation that’s done by the validation controls.</td>
</tr>
<tr>
<td>None</td>
<td>Uses the older method of generating the JavaScript code for the validation controls and including it within script elements in the HTML for the page.</td>
</tr>
</tbody>
</table>

The NuGet Package Manager page

How to install the NuGet package for jQuery validation

- Right-click on the project in the Solution Explorer and select Manage NuGet Packages.
- In the page that appears, click “Browse” and type “aspnet.scriptmanager.jquery” in the search box.
- Select the package in the left pane and click on the Install button in the right pane.

A Load event handler that turns off unobtrusive validation for a page

```csharp
protected void Page_Load(object sender, EventArgs e) {
    UnobtrusiveValidationMode = UnobtrusiveValidationMode.None;
}
```

A Web.config setting that turns off unobtrusive validation for an application

```xml
<appSettings>
    <add key="ValidationSettings:UnobtrusiveValidationMode" value="None"/>
</appSettings>
```

Description

- ASP.NET provides a feature called *unobtrusive validation*. When it’s enabled, a JavaScript library named *jQuery* is used to do the validation that’s specified.
- Unobtrusive validation is on by default when you start a new web application from the Empty template. So if you’re using the validation controls, you either need to turn unobtrusive validation off or install the NuGet package for jQuery validation.
The aspx code for the Future Value form

Figure 2-14 presents the aspx code for the Future Value form. To help you see how the code relates to the form in the browser, this figure starts with the form displayed in Internet Explorer. Here, you can see that the title in the browser tab is the same as the title in the title element in the head section of the HTML.

After the title element, you can see a style element that includes two CSS rule sets. You can also see class attributes in the table and td elements that refer to these rule sets. In the next chapter, you’ll learn how this works, but for now realize that ASP.NET does this automatically when you use the Designer to format the elements on a page, even though this isn’t the best way to handle this formatting.

Within the body element, the first two elements are for the image and the h1 heading. You can see how these are rendered in the browser. This is followed by a form element that contains a div element. These form and div elements are generated by ASP.NET when you add a new form to a web application.

Within the div element is a table that contains six tr elements, one for each row. Within each of these elements are two td elements, one for each column. That’s the way the HTML for a table works. In the first td element for each of the first four rows, you can see the text that has been entered. In the second td element for each of these rows, a server control has been added. For instance, the control in the first row is a drop-down list, and the control in the second row is a text box. You can see how this table is rendered in the browser.

For each control, the ID property is used to give the control an identifier that’s easy to refer to. Here, ddl is used as a prefix for a drop-down list and txt is used as a prefix for a text box. That makes it easy to tell what type of control an identifier refers to. These identifiers are followed by names that clearly identify the controls. Within those names, the first letter of each word is capitalized, which makes the names easier to read. This is the naming convention that’s used throughout this book and the one that we recommend for your own use.
The design of the Future Value form

The aspx code for the Future Value form

```csharp
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="Ch02FutureValue.Default" %>

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Chapter 2: Future Value</title>
    <style type="text/css">
        .auto-style1 {
            width: 100%;
        }
        .auto-style2 {
            width: 172px;
        }
    </style>
</head>
<body>
<img src="Images/MurachLogo.jpg" alt="Murach Logo" />
<h1>401K Future Value Calculator</h1>
<form id="form1" runat="server">
    <div>
        <table class="auto-style1">
            <tr>
                <td class="auto-style2">Monthly investment</td>
                <td><asp:DropDownList ID="ddlMonthlyInvestment" runat="server" Height="22px" Width="147px">\n                    \n                </asp:DropDownList></td>
            </tr>
        </table>
    
    <label>Annual interest rate</label>
    <input type="number" value="0" />
    
    <label>Number of years</label>
    <input type="number" />
    
    <label>Future value</label>
    <input type="number" />
    
    <input type="button" value="Calculate" />
    <input type="button" value="Clear" />
    
    Interest rate must range from 1 to 20.
    Number of years is required.
</form>
</body>
</html>
```
In the code for the first text box control, you can see that the Text property has been set to a value of 3.0. That is the interest rate that will be displayed when the form is first displayed in the browser. In contrast, the value for the second text box control is coded between the opening and closing tags for the control. These are two different ways to set the starting value for a control.

In the code for the fifth row of the table, you can see that ASP.NET has put a non-breaking space (&nbsp;) in the cell for each column. It does that for all of the empty cells in a table.

In the sixth row, you can see the aspx code for the Calculate and Clear buttons. Here, each of the buttons has an OnClick property that points to the C# code in the code-behind file that will be run when the button is clicked. The Clear button also contains a CausesValidation property, which is set to False. This property tells the page not to do validation when the button is clicked. Because the default value of the CausesValidation property is True for buttons, this property doesn’t need to be set for the Calculate button.

This table is followed by the code for the validation controls. Because these controls are outside the table, their placement will be determined by flow layout. To have some control over this layout, a break element (<br />) is coded after the two interest rate validators. That means the error messages for the interest rate will be displayed on one line, and the messages for the years will be on another line. However, the Display property for these validators has been set to Dynamic, which means that space will be allocated for them only when it is needed.
The aspx code for the Future Value form (continued)

```html
<tr>
    <td class="auto-style2">Annual interest rate</td>
    <td><asp:TextBox ID="txtInterestRate" runat="server" Text="3.0"></asp:TextBox></td>
</tr>
<tr>
    <td class="auto-style2">Number of years</td>
    <td><asp:TextBox ID="txtYears" runat="server">10</asp:TextBox></td>
</tr>
<tr>
    <td>FUTURE VALUE</td>
    <td><asp:Label ID="lblFutureValue" runat="server" Font-Bold="True"></asp:Label></td>
</tr>
<tr>
    <td class="auto-style2"></td>
    <td>&nbsp;</td>
</tr>
<tr>
    <td class="auto-style2">
        <asp:Button ID="btnCalculate" runat="server" Text="Calculate" Width="122px" OnClick="btnCalculate_Click" /></td>
    <td><asp:Button ID="btnClear" runat="server" Text="Clear" Width="122px" CausesValidation="False" OnClick="btnClear_Click" /></td>
</tr>
</table>
</asp:RequiredFieldValidator>
</asp:RangeValidator>
<br />
<asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server" ErrorMessage="Number of years is required." ControlToValidate="txtYears" Display="Dynamic" ForeColor="Red">
</asp:RequiredFieldValidator>
<asp:RangeValidator ID="RangeValidator2" runat="server" ErrorMessage="Years must range from 1 to 45." ControlToValidate="txtYears" Type="Integer" Display="Dynamic" ForeColor="Red" MaximumValue="45" MinimumValue="1">
</asp:RangeValidator>
</div>
</form>
</body>
</html>

Figure 2-14 The aspx code for the Future Value form (part 2 of 2)
How to add C# code to a form

To add the functionality required by a web form, you add C# 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

Figure 2-15 shows how to use the Code Editor to enter and edit C# code, starting with two ways to start an event handler. If, for example, you double-click a control, an event handler for the default event of the control is started. If you double-click on a button control, for example, an event handler for the Click event of that control is created. Then, you can enter the code that you want to be executed within the braces of the event handler.

To create event handlers for other control events, you can use the Events button at the top of the Properties window. When you click this button, a list of all the events for the control that’s currently selected is displayed. Then, you can double-click on any event to generate an event handler for that event.

When Visual Studio generates an event handler for a control, it also adds the appropriate event property to the aspx code for that control. In the aspx code, for example, you saw the OnClick event properties that were generated for the Calculate and Clear buttons on the Future Value form. This is how events are wired to event handlers in ASP.NET.

When you add a web form to an application, Visual Studio automatically starts the Page_Load event handler. Then, you can add the code you need to this handler. An easy way to display this handler is to double-click outside the body of the form in Design view.

You can also code methods other than event handlers by entering the code for the method directly into the Code Editor window. Then, you can call those methods from the event handlers for the form.

As you enter C# code, be sure to take advantage of the snippets that the Code Editor offers. If, for example, you insert the snippet for a for loop, all of the code that you need for that structure is inserted into the code-behind file. Then, you can modify that code to suit your requirements. As you work, the Code Editor also provides IntelliSense that makes it easier to enter code.

Another useful feature of the Code Editor is the Quick Actions light bulb icon. When errors occur in your code, they’re underlined with a wavy line. If Visual Studio has suggestions for how to fix the error, this light bulb icon appears when you hover your mouse over the wavy line. Then, you can use the technique described in this figure to see and apply the suggested fixes.

Sometimes you’ll notice the light bulb icon at the beginning of a line of code even though there aren’t errors. That means that the Code Editor has identified something in that line of code that can be improved. Then, you can use the same technique you use to fix an error to get and apply the suggested improvements if you choose to.

When you test a web form, you may want to comment out portions of code by putting those portions of code within C# comments. Then, because comments
The Code Editor for a web form

Two ways to start an event handler
- Double-click on a control in the Designer to start an event handler for the default event of that control.
- Select a control in the Designer, click the Events button in the Properties window (the button with the lightning bolt), and double-click the event you want.

How to insert a code snippet
- Move the insertion point to where you want the snippet. Right-click, select Insert Snippet, select Visual C#, and select the snippet.

How to use the Quick Actions light bulb feature
- If errors are detected, they are underlined with a wavy line. Hover your mouse over the error to see an explanation. If a light bulb icon appears, click the down arrow next to it for a list of possible fixes for the error. Then, click on a fix to apply it.

How to comment out a portion of code
- Select the lines of code, then click on the Comment button in the Text Editor toolbar. To uncomment the lines, select them and click the Uncomment button.

Description
- An event handler is a C# method that is executed when an event occurs, and Visual Studio will generate the starting code for an event handler.
- The Code Editor includes editing features such as IntelliSense, automatic indentation, snippets, syntax checking, and Quick Actions light bulbs.
- When you add a web form to a project, Visual Studio automatically adds a Load event handler for the page. You can double-click outside the body of the form in Design view to display this event handler.
- To enter a method other than an event handler, you type the method from scratch.
are ignored, you can test the form to see whether those statements were the cause of a problem. Later, you can uncomment those lines of code and test again.

**How to use page and control events**

The first table in figure 2-16 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 section 2, you’ll see how this event can be useful when working with data in session state.

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, followed by the event handler for the Click event of the button that was clicked.

The TextChanged event occurs when the user changes the value in a text box. In contrast, the CheckedChanged event occurs when the user clicks a radio button or check box, and the SelectedIndexChanged event occurs when the user selects an item from a 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 Init and Load event handlers for the page. 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 Init and Load 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. This event handler resets the value in the drop-down list to the first value in the list by setting the SelectedIndex property of the control to 0. This handler also resets the text boxes and label to empty strings. Note that the name of this event handler is btnClear_Click, which is the ID of the button followed by an underscore and the name of the event. If you look back at the aspx code, you’ll see that this is the same name that’s in the OnClick event property that ASP.NET adds to the control when it generates the event handler.
Common ASP.NET page events

<table>
<thead>
<tr>
<th>Event</th>
<th>Method name</th>
<th>Occurs when...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>Page_Init</td>
<td>A page is requested from the server. This event is raised before the view state of the page controls has been restored.</td>
</tr>
<tr>
<td>Load</td>
<td>Page_Load</td>
<td>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.</td>
</tr>
<tr>
<td>PreRender</td>
<td>Page_PreRender</td>
<td>All the control events for the page have been processed but before the HTML that will be sent back to the browser is generated.</td>
</tr>
</tbody>
</table>

Common ASP.NET control events

<table>
<thead>
<tr>
<th>Event</th>
<th>Occurs when...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click</td>
<td>The user clicks a button, link button, or image button control.</td>
</tr>
<tr>
<td>TextChanged</td>
<td>The user changes the value in a text box.</td>
</tr>
<tr>
<td>CheckedChanged</td>
<td>The user selects a radio button in a group of radio buttons or selects or unselects a check box.</td>
</tr>
<tr>
<td>SelectedIndexChanged</td>
<td>The user selects an item from a drop-down list or a list box.</td>
</tr>
</tbody>
</table>

Code for the Click event of the btnClear button

```csharp
protected void btnClear_Click(object sender, EventArgs e)
{
    ddlMonthlyInvestment.SelectedIndex = 0;
    txtInterestRate.Text = "";
    txtYears.Text = "";
    lblFutureValue.Text = "";
}
```

Description

- All of the events handlers for an ASP.NET web page and its server controls are executed on the server. Because of that, a page must be posted back to the server before its events can be handled.
- 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. Then, the event handlers for any control events that were raised are executed in the order in which they were raised.
The C# code for the Future Value form

Figure 2-17 presents the C# code for the code-behind file of the Future Value form. It consists of multiple using directives (not all shown here), three event handlers, and a method named CalculateFutureValue. The event handlers handle the Load event for the page, the Click event of the Calculate button, and the Click event of the Clear button. The CalculateFutureValue method is called by the event handler for the Click event of the Calculate button.

In the code in this figure, the highlighted properties are the ones that are commonly tested in the code for web forms. The first one is the IsPostBack property that’s used in the Page_Load method. 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 method 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 by using the Add method of the Items collection for the 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. Instead, the values are restored from view state.

In contrast, the unobtrusive validation mode needs to be set every time the page loads. If it isn’t set, an error will occur. That’s why the code that sets this mode comes before the if statement in the Page_Load event handler.

The other page property that’s commonly tested is the IsValid property. It’s useful when the user’s browser doesn’t support the client-side scripts 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 method, you can see how the IsValid test is used. If it isn’t True, the processing isn’t done. But otherwise, this method uses the SelectedValue property of the drop-down list to get the value of the selected item, which represents the investment amount. Then, it uses the Text properties of the text boxes to get the years and interest rate values. After it gets these values, it converts them to their data types (integer and decimal). Last, it calls the CalculateFutureValue method to calculate the future value, uses the ToString method to convert the future value to a string with currency format, and puts the formatted value in the label of the form. When this method ends, the web form is sent back to the user’s browser.

Notice that this code doesn’t do anything if IsValid is false. That’s because it doesn’t need to. If the data isn’t valid, the validation controls will display the appropriate messages, just like they do when client-side scripts are enabled.

With the exception of the IsPostBack and IsValid properties and the unobtrusive validation setting, this is all standard C# code. Because of that, you shouldn’t have any trouble following it. But if you do, you can quickly upgrade your C# skills by getting our latest C# book.
The C# code for the Future Value form

using System;
using System.Web;
using System.Web.UI;
// more using directives generated by Visual Studio

namespace Ch02FutureValue
{
    public partial class Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            UnobtrusiveValidationMode = UnobtrusiveValidationMode.None;
            if (!IsPostBack)
            {
                for (int i = 50; i <= 500; i += 50)
                    ddlMonthlyInvestment.Items.Add(i.ToString());
            }

            protected void btnCalculate_Click(object sender, EventArgs e)
            {
                if (IsValid)
                {
                    int monthlyInvestment = Convert.ToInt32(ddlMonthlyInvestment.SelectedValue);
                    decimal yearlyInterestRate = Convert.ToDecimal(txtInterestRate.Text);
                    int years = Convert.ToInt32(txtYears.Text);
                    decimal futureValue = this.CalculateFutureValue(monthlyInvestment, yearlyInterestRate, years);
                    lblFutureValue.Text = futureValue.ToString("c");
                }
            }

            protected decimal CalculateFutureValue(int monthlyInvestment, decimal yearlyInterestRate, int years)
            {
                int months = years * 12;
                decimal monthlyInterestRate = yearlyInterestRate / 12 / 100;
                decimal futureValue = 0;
                for (int i = 0; i < months; i++)
                {
                    futureValue = (futureValue + monthlyInvestment) * (1 + monthlyInterestRate);
                }
                return futureValue;
            }

            protected void btnClear_Click(object sender, EventArgs e)
            {
                ddlMonthlyInvestment.SelectedIndex = 0;
                txtInterestRate.Text = "";
                txtYears.Text = "";
                lblFutureValue.Text = "";
            }
        }
    }
}

Figure 2-17 The C# code for the Future Value form
How to test a web application

After you design the forms and develop the C# code for a web application, you need to test it to be sure it works properly. Then, if you discover any errors, you need to find the errors, correct them, and test again. For now, you’ll just learn some basic skills for testing and application. But in chapter 5, you’ll learn more about testing and debugging.

How to run a web application

To run a web application, you can use one of the techniques in figure 2-18. Before Visual Studio runs the application, though, it compiles the aspx and C# code for the web forms. Then, if the web forms compile without errors, Visual Studio runs the application using IIS Express and displays the starting page of the application in your default browser. At that point, 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 syntax errors that have to be corrected as well as warning messages. In this figure, just one error message and no warning messages are displayed.

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.

Keep in mind, though, that the error may not be in the statement in the line of code that causes the problem. For instance, the message in this example says that futurevalue doesn’t exist, but the problem is that this variable was spelled differently when it was declared. To fix that, you need to fix the declaration.

After you fix all of the compilation errors and run the application in the browser, you should be aware that an exception may occur. That happens when ASP.NET can’t execute one of the statements in the C# code, even though it compiled without error. Then, if the exception isn’t handled by the application, ASP.NET switches to the Code Editor window and highlights the statement that caused the exception. At that point, you can stop 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 addition to testing whether the web application runs without error, you should also test to see that it displays correctly in different browsers. Visual Studio makes it easy to change the default browsers for this purpose by providing a drop-down browser list. After you use that list to change the default browser, you can click on the browser name or press F5 to run the web application in that browser.
How to run an application

- To run an application in the default browser, press F5 or click on the browser name in the Standard toolbar.
- To change the default browser, select a browser from the drop-down browser list.

How to stop an application

- Click the Close button in the upper right corner of the browser. For some browsers like Internet Explorer, that will stop the application in Visual Studio.
- In Visual Studio, click the Stop Debugging button in the Debug toolbar (the one with the red square on it) or press Shift+F5. This also stops an application when an exception occurs.

How to fix syntax errors and exceptions

- To go to the statement that caused a syntax error, double-click on the error in the Error List window. That will give you a clue to the cause of the error.
- When an exception occurs, the application is interrupted and the statement that caused the error is displayed in Visual Studio. Then, you can stop the application and debug it.

Description

- If any errors are detected when you run an application, a dialog box asks whether you want to continue by running the last successful build. If you click No, the application isn’t run and an Error List is displayed.
- If a statement can’t be executed when the application is run, even though it compiles successfully, an exception will occur. Then, you need to debug the problem.
How to view the HTML that’s sent to the browser

To view the HTML for a page that’s displayed in a browser, you can use one of the techniques in figure 2-19. Although you won’t need to view this code often, it gives you a better idea of what’s going on behind the scenes. It can also be helpful when you need to see exactly how ASP.NET has rendered the aspx code to HTML.

You should know, though, that the HTML code in the example in this figure has been copied from the web browser into Notepad++ and then formatted to make it easier to follow. In contrast, the code that’s displayed in the web browser isn’t formatted so it’s harder to review.

In this example, you can see some of the HTML that has been rendered for the Future Value form after the user has selected a value from the drop-down list, entered values into the text boxes, and clicked the Calculate button. This code is instructive in several ways.

First, note that this code doesn’t include any aspx code. That’s because the aspx has been converted to HTML. For instance, the aspx for the drop-down list in the first row of the table has been converted to an HTML select element that contains one option element for each value in the list.

Second, note that view state data is stored in a hidden input field named _VIEWSTATE. However, the value of this field is encoded 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, note that the values that the user entered are included in the HTML. For instance, the value in the drop-down list is 50, and the value in the first text box is 3.0. 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.

Fourth, note the script element that comes right after the view state data. Although the JavaScript code that it contained has been replaced with a comment, this is one of several script elements that were generated for this form. They provide the JavaScript code for validating the data in the browser.

Fifth, note that the HTML for a label server control is a span element, not a label element as you might expect. If you were able to scroll down, you would also see that the error messages for the validation controls are displayed in span elements. As you will see in the next chapter, you sometimes need to know how a server control is rendered in HTML if you want to apply CSS formatting to it, and reviewing the source code is one way to find out.

Keep in mind that this HTML is generated automatically by ASP.NET, so you usually don’t have to worry about it. You just develop the application by using Visual Studio, and the rest of the work is done for you. Sometimes, though, reviewing the source code can help you solve a debugging problem.
Some of the HTML for the Future Value form after a post back

Two ways to view the HTML for a page in a browser

- Select the View → Source command from the browser’s menu.
- Right-click on the web page and select the View Source command from the shortcut menu.

Description

- When an ASP.NET page is requested by a browser, ASP.NET generates the HTML for the page and returns that HTML to the browser.
- View state data is stored in a hidden input field within the HTML. This data is encoded so you can’t read it.
- If the page contains validation controls, the HTML for the page contains script elements that include the JavaScript that does the validation.
- Values that the user enters into a page are returned to the browser as part of the HTML.
Perspective

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

As you will see in the next chapter, though, you should also have a solid set of HTML and CSS skills. That way, you can separate the content for a web page (the HTML) from its formatting (the CSS), and that makes it easier to develop and maintain the pages of a web application. You’ll also learn the basics of using Bootstrap in the next chapter, which makes it easier to develop pages that can be displayed on devices of varying sizes.

Terms

<table>
<thead>
<tr>
<th>template</th>
<th>snippet</th>
</tr>
</thead>
<tbody>
<tr>
<td>target framework</td>
<td>web server control</td>
</tr>
<tr>
<td>default template</td>
<td>smart tag menu</td>
</tr>
<tr>
<td>web form</td>
<td>property</td>
</tr>
<tr>
<td>Web Forms Designer</td>
<td>validation control</td>
</tr>
<tr>
<td>Designer</td>
<td>validator</td>
</tr>
<tr>
<td>Toolbox</td>
<td>required field validator</td>
</tr>
<tr>
<td>Solution Explorer</td>
<td>range validator</td>
</tr>
<tr>
<td>Properties window</td>
<td>unobtrusive validation</td>
</tr>
<tr>
<td>Source view</td>
<td>jQuery</td>
</tr>
<tr>
<td>Design view</td>
<td>event handler</td>
</tr>
<tr>
<td>Split view</td>
<td>Code Editor</td>
</tr>
<tr>
<td>IIS Express</td>
<td>syntax error</td>
</tr>
<tr>
<td>flow layout</td>
<td>exception</td>
</tr>
</tbody>
</table>

Summary

- You create a web application from a template that determines the folders and files that Visual Studio adds to the project.
- The target framework determines the features that are available to an application. By default, the target framework for a web application is .NET Framework 4.5.2.
- When you use Visual Studio to design web forms, the primary window is the Web Forms Designer (or just Designer). It is supported by the Toolbox, the Solution Explorer, and the Properties window.
- When you use the Designer to build a web form, you can work in Source view, Design view, or Split view.
• In Source view, *IntelliSense* and *snippets* help you enter the tags and attributes for HTML elements and web server controls. In Design view, the Toolbox makes it easy to add web server controls to a form and the Properties window helps you set the *properties* for the controls.

• ASP.NET provides *validation controls* that provide for both client-side and server-side data validation. For client-side validation, JavaScript must be enabled in the user’s browser, but most browsers have it enabled.

• When *unobtrusive validation* is used for a web application, ASP.NET will use a JavaScript library called *jQuery* for the validation. That option is on by default when you start a new web application.

• Visual Studio provides a *Code Editor* with IntelliSense and snippets that makes it easier to enter the C# statements for the *event handlers* and other methods that a web application requires.

• Three of the page events that can trigger an event handler are the Init, Load, and PreRender events. The first two are raised each time a page is requested from the server. The last one is raised right before the HTML is generated for a page.

• Four of the server control events that can trigger an event handler are the Click event for a button, the TextChanged event for a text box, the CheckedChanged event for a check box or radio button, and the SelectedIndexChanged event for a list.

• The IsPostBack property of a page can be used to tell whether a page is being posted back from a browser or loaded for the first time. The IsValid property of a page can be used to tell whether the validation controls have found that all of the entries are valid.

• If you try to run a web form that has *syntax errors* in the C# code, Visual Studio stops compiling the assembly and displays the errors in an Error List window.

• An *exception* occurs when ASP.NET can’t execute one of the statements in the C# code, even though it compiles without error. Then, you need to stop the application, find the cause of the exception, and fix it.

• If you view the source code while a page is displayed in a browser, you can see the hidden fields that are used for view state, the scripts that are used for data validation, and the HTML that’s generated for the server controls.
Before you do the exercises for this book...

If you haven’t already done so, you need to install the software that’s required for this book as well as the downloadable applications. Appendix A shows how to do that.

Exercise 2-1  Build the Future Value application

This exercise guides you through the development of the Future Value application that’s presented in this chapter. This will give you a chance to experiment with the many features that Visual Studio offers.

Start, close, and open the web application

1. Start a web application with the Empty template as shown in figures 2-1 and 2-2. Name the application Ex02FutureValue and store it in the C:\aspnet46_cs directory.

2. Add a web form as shown in figure 2-3 and name it Default.aspx.

3. Add a folder named Images to your project and add the MurachLogo.jpg file to it using the techniques in figure 2-5. The jpg file is in the C:\aspnet46_cs directory.

4. Close the application as shown in figure 2-6, saving changes if you’re prompted to. Then, open the application again as shown in this figure.

Use the Web Forms Designer to build the form

5. Open the Default.aspx web form and switch to Source view. Type “Chapter 2: Future Value” in the title element in the head section of the HTML.

6. Move the cursor to the end of the opening body tag and press the Enter key to create a new line. Next, drag the Murach logo file from the Images folder in the Solution Explorer to the new line. That should create an img element with a properly coded src attribute. Now, add an alt attribute to this element with “Murach Logo” as its value, and switch to Design view to see the changes.

7. Switch to Source view, place the insertion point after the img element, and add an h1 element that has “401K Future Value Calculator” as its content.

8. Switch to Design view to see this change. Then, run the form in the default browser by pressing F5. That automatically saves the changes to the Default.aspx file. After your form is displayed in the default browser, close the browser. If your default browser is Internet Explorer, this should stop the application.

9. Return to Visual Studio. If the Toolbox isn’t available, that means the web form is still running. So, click the Stop Debugging button in the Debug toolbar to stop the application.

10. In Source view, place the insertion point inside the div tags. Then, in Design view, use the techniques in figure 2-8 to add a table that provides for six rows and two columns. Next, add the text shown in the first four rows to the first column of the table. Then, drag the right boundary of the first column to reduce its width as shown in this figure.
11. Switch to Source view to see the HTML for the table. Note that a style element has been added to the head section and class attributes have been added to the table element and some of the td elements. This was generated when you reduced the width of the first column by dragging its boundary. Note too that non-breaking space characters (&nbsp;) have been generated for the empty td elements.

12. Switch to Design view and use the techniques in figure 2-9 to add the drop-down list, text boxes, label, and buttons shown in that figure to the table. Then, adjust the size of the list, text box, and buttons, but not the label, so the table looks the way you want it to.

13. Use the techniques of figure 2-9 and the summary in figure 2-10 to set the ID and Text properties of the controls. For the Clear button, also set the CausesValidation property to False. For the label, delete the value of the Text property.

14. Press F5 to run the application, and check the web form to make sure it looks the way it’s supposed to. Then, switch to Visual Studio and click the Stop Debugging button in the Debug toolbar.

**Add the validation controls**

15. In Source view or Design view, add the validation controls for the text boxes as shown in figures 2-11 and 2-12.

16. In Design view, double-click outside the body of the web form to go to the event handler for the Load event of the page in the code-behind file for the form. Then, turn off unobtrusive validation by adding the statement in the Load event handler shown in figure 2-13. IntelliSense makes this easy.

17. 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 the Calculate button.

18. Stop the application. Then, if necessary, fix any problems and test again. If, for example, validation is done when you click the Clear button, make sure its CausesValidation property is set to False. Or, if a range validator is behaving strangely, make sure its Type property is set correctly.

**Add the C# code and test as you go**

19. Double-click outside of the body of the form in Design view to switch to the Code Editor. That will take you to the Load event handler that you started coding earlier.

20. At the top of the Code Editor, note the using directives that Visual Studio generates. Then, add the rest of the code for the Load event handler shown in figure 2-17, taking full advantage of the IntelliSense that’s provided. Then, press F5 to compile and test this event handler. If any syntax errors are detected, use the techniques in figure 2-18 to fix them.

21. In the aspx file, in Design view, double-click on the Clear button to start an event handler for the Click event of that button. Then, enter the code for this event handler as shown in figure 2-17, and test again.
22. Enter the code for the CalculateFutureValue method that’s shown in figure 2-17. When you’re ready to add the for loop, right-click, select Visual C# from the shortcut menu, and select the for snippet. Then, finish the coding for this method.

23. In the aspx file, in Design view, double-click on the Calculate button to start an event handler for the Click event of that button. Next, type “if” and then press the Tab key twice to insert the snippet for the if statement. Then, enter the code for this event handler. This method should call the CalculateFutureValue method as in figure 2-17. Then, test this code.

24. If necessary, fix any design or coding problems that remain. When you’re through, the application should work the way you want it to.

**Do more testing and experimenting**

25. Set the EnableViewState property of the drop-down list to False, and test the application to see what happens. When an exception occurs, stop the application and reset the property.

26. Set the EnableClientScript property for all four validators to False so the validation will only be done on the server. Then, test the application to make sure that the validation still works. When you’re through testing, end the application and reset these properties.

27. Run the application again, and use the technique in figure 2-19 to review the HTML that’s sent to the browser. There, you can see the HTML that’s generated for the web form, the input element with the “hidden” type that’s used for view state, and the script elements that contain the JavaScript that’s used for client-side validation.

28. When you’re through experimenting, close the project. Then, close Visual Studio.
How to build your .NET web programming skills

The easiest way is to let Murach’s ASP.NET 4.6 Web Programming with C# 2015 be your guide! So if you’ve enjoyed this chapter, I hope you’ll get your own copy of the book today. You can use it to:

- Teach yourself how to develop professional, bulletproof ASP.NET Web Forms applications that manage state, validate data, handle everyday business needs, do database processing, and are inviting and easy-to-navigate for users

- Pick up new skills whenever you want to or need to by focusing on material that’s new to you

- Look up coding details or refresh your memory on forgotten details when you’re in the middle of developing a Web Forms application

- Loan to your colleagues who are always asking you questions about ASP.NET Web Forms programming

To get your copy, you can order online at www.murach.com or call us at 1-800-221-5528 (toll-free in the U.S. and Canada). And remember, when you order directly from us, this book comes with my personal guarantee:

100% Guarantee

You must be satisfied. Each book you buy directly from us must outperform any competing book or course you’ve ever tried, or send it back within 60 days for a full refund…no questions asked.

Thanks for your interest in Murach books!

Mike Murach, Publisher