Web App Development with ASP.NET

23

If any man will draw up his case, and put his name at the foot of the first page, I will give him an immediate reply. Where he compels me to turn over the sheet, he must wait my leisure.
—Lord Sandwich

Objectives

In this chapter you'll learn:

- Web app development using ASP.NET.
- To handle the events from a Web Form's controls.
- To use validation controls to ensure that data is in the correct format before it's sent from a client to the server.
- To maintain user-specific information.
- To create a data-driven web app using ASP.NET and the ADO.NET Entity Framework.

Outline

23.1 Introduction

In this chapter, we introduce **web-app development** with Microsoft's **ASP.NET** technology. Web-based apps create web content for web-browser clients.

We present several examples that demonstrate web-app development using **Web Forms**, **web controls** (also called **ASP.NET server controls**) and Visual C# programming. Web Form files have the filename extension **`.aspx`** and contain the web page's GUI. You customize Web Forms by adding web controls including labels, textboxes, images, buttons and other GUI components. The Web Form file represents the web page that's sent to the client browser. We often refer to Web Form files as **ASPX files**.

An ASPX file created in Visual Studio has a corresponding class written in a .NET language—we use Visual C# in this book. This class contains event handlers, initialization code, utility methods and other supporting code. The file that contains this class is called the **code-behind file** and provides the ASPX file's programmatic implementation.

Software Used in This Chapter

To develop the code and GUIs in this chapter, we used Microsoft's **Visual Studio Express 2012 for Web**—a free IDE designed for developing ASP.NET web apps. The full version of Visual Studio 2012 includes the functionality of Visual Studio Express 2012 for Web, so the instructions we present in this chapter also apply to Visual Studio 2012. See the Before You Begin section that follows the Preface for additional information on downloading and installing the software for this chapter.

Online Chapter—Web App Development with ASP.NET: A Deeper Look

In the online Chapter 29, Web App Development with ASP.NET: A Deeper Look, we present several additional web-app development topics, including:

- master pages to maintain a uniform look-and-feel across the pages in a web app
- creating password-protected websites with registration and login capabilities

- using the **Web Site Administration Tool** to specify which parts of a website are password protected
- using ASP.NET AJAX to quickly and easily improve the user experience for your web apps, giving them responsiveness comparable to that of desktop apps.

23.2 Web Basics

In this section, we discuss what occurs when a user requests a web page in a browser. In its simplest form, a *web page* is nothing more than an *HTML (HyperText Markup Language) document* (with the extension `.html` or `.htm`) that describes to a web browser the document's content and how to format it.

HTML documents normally contain *hyperlinks* that link to different pages or to other parts of the same page. When the user clicks a hyperlink, a **web server** locates the requested web page and sends it to the user's web browser. Similarly, the user can type the *address of a web page* into the browser's *address field* and press *Enter* to view the specified page.

In this chapter, we develop web apps using visual development techniques that are similar to those you used with Windows `Forms` in Chapters 14–15. To take full advantage of web app development, you'll also want to learn HTML5, CSSS and JavaScript—topics that we cover in our textbook *Internet & World Wide Web How to Program, 5/e*. You can learn more about this book at `deitel.com/books/iw3htp5`.

URIs and URLs

URIs (Uniform Resource Identifiers) identify resources on the Internet. URIs that start with `http://` are called *URLs (Uniform Resource Locators)*. Common URLs refer to files, directories or server-side code that performs tasks such as database lookups, Internet searches and business application processing. If you know the URL of a publicly available resource anywhere on the web, you can enter that URL into a web browser's address field and the browser can access that resource.

Parts of a URL

A URL contains information that directs a browser to the resource that the user wishes to access. Web servers make such resources available to web clients. Popular web servers include Microsoft's Internet Information Services (IIS), Apache's HTTP Server and Nginx.[1]

Let's examine the components of the URL

```
    http://www.deitel.com/books/downloads.html
```

The `http://` indicates that the HyperText Transfer Protocol (HTTP) should be used to obtain the resource. HTTP is the web protocol that enables clients and servers to communicate. Next in the URL is the server's fully qualified **hostname** (`www.deitel.com`)—the name of the web server computer on which the resource resides. This computer is referred to as the **host**, because it houses and maintains resources. The hostname `www.deitel.com` is translated into an **IP (Internet Protocol) address**—a numerical value that uniquely identifies the server on the Internet. A **Domain Name System (DNS) server** maintains a database of hostnames and their corresponding IP addresses, and performs the translations automatically.

1. `w3techs.com/`.

The remainder of the URL (`/books/downloads.html`) specifies the resource's location (`/books`) and name (`downloads.html`) on the web server. The location could represent an actual directory on the web server's file system. For *security* reasons, however, the location is typically a *virtual directory*. The web server translates the virtual directory into a real location on the server, thus hiding the resource's true location.

Making a Request and Receiving a Response

When given a URL, a web browser uses HTTP to retrieve the web page found at that address. Figure 23.1 shows a web browser sending a request to a web server. Figure 23.2 shows the web server responding to that request.

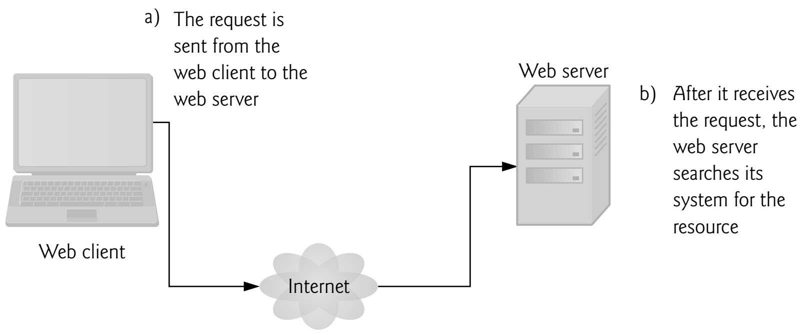

a) The request is sent from the web client to the web server
Web server
b) After it receives the request, the web server searches its system for the resource
Web client
Internet

Fig. 23.1 | Client requesting a resource from a web server.

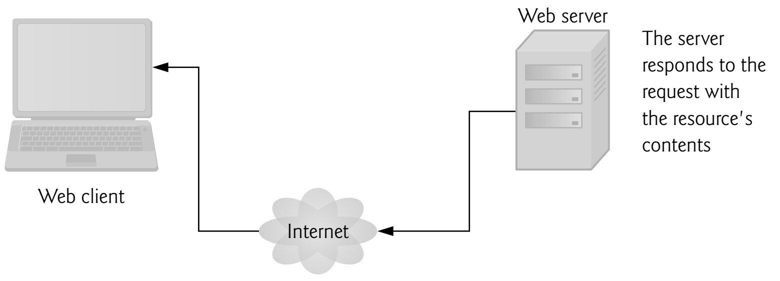

Web server
The server responds to the request with the resource's contents
Web client
Internet

Fig. 23.2 | Client receiving a response from the web server.

23.3 Multitier App Architecture

Web-based apps are **multitier apps** (sometimes referred to as ***n*-tier apps**). Multitier apps divide functionality into separate **tiers** (that is, logical groupings of functionality). Although tiers can be located on the *same* computer, the tiers of web-based apps commonly reside on *separate* computers for *security* and *scalability*. Figure 23.3 presents the basic architecture of a three-tier web-based app.

Information Tier

The **bottom tier** (also called the **information tier**) maintains the app's data. This tier typically stores data in a relational database management system. For example, a retail store might have a database for storing product information, such as descriptions, prices and quantities in stock. The same database also might contain customer information, such as user names, billing addresses and credit card numbers. This tier can contain multiple databases, which together comprise the data needed for an app.

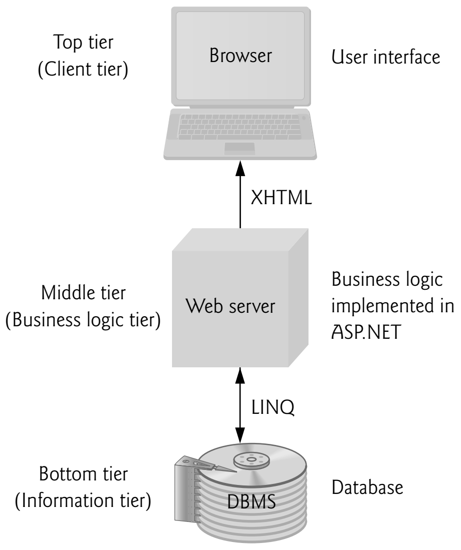


Fig. 23.3 | Three-tier architecture.

Business Logic

The **middle tier** implements **business logic**, **controller logic** and **presentation logic** to control interactions between the app's clients and its data. The middle tier acts as an intermediary between data in the information tier and the app's clients. The middle-tier controller logic processes client requests (such as requests to view a product catalog) and retrieves data from the database. The middle-tier presentation logic then processes data from the information tier and presents the content to the client. Web apps present data to clients as web pages.

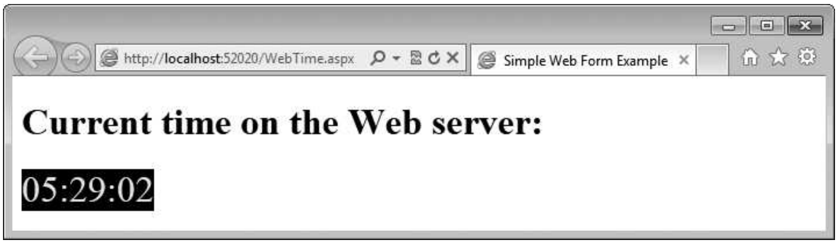
Business logic in the middle tier enforces *business rules* and ensures that data is reliable before the server app updates the database or presents the data to users. Business rules dictate how clients can and cannot access app data, and how apps process data. For example, a business rule in the middle tier of a retail store's web-based app might ensure that all product quantities remain positive. A client request to set a negative quantity in the bottom tier's product information database would be rejected by the middle tier's business logic.

Client Tier

The **top tier**, or **client tier**, is the app's user interface, which gathers input and displays output. Users interact directly with the app through the user interface (typically viewed in a web browser), keyboard and mouse. In response to user actions (for example, clicking a hyperlink), the client tier interacts with the middle tier to make requests and to retrieve data from the information tier. The client tier then displays to the user the data retrieved from the middle tier. The client tier *never* directly interacts with the information tier.

23.4 Your First Web App

Our first example displays the web server's time of day in a browser window (Fig. 23.4). When this app executes—that is, a web browser requests the app's web page—the web server executes the app's code, which gets the current time and displays it in a `Label`. The web server then returns the result to the web browser that made the request, and the web browser *renders* the web page containing the time. We executed this app in both the *Internet Explorer* and *Firefox* web browsers to show you that the web page renders identically in different browsers—the page should look the same in most Web browsers.

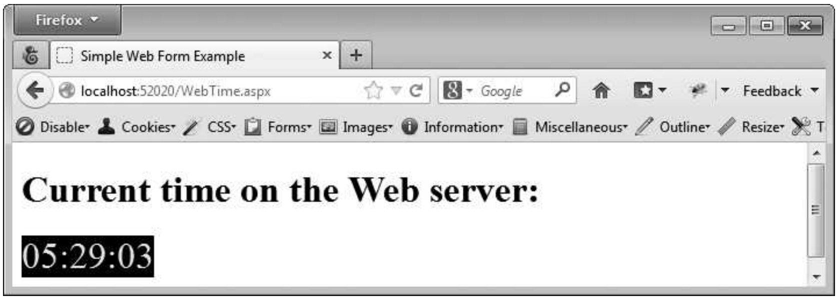


http://localhost:52020/WebTime.aspx

Simple Web Form Example

Current time on the Web server:

05:29:02



Simple Web Form Example

localhost:52020/WebTime.aspx

Current time on the Web server:

05:29:03

Fig. 23.4 | `WebTime` web app running in both Internet Explorer and Firefox.

Testing the App in Your Default Web Browser

To test this app in your default web browser, perform the following steps:

1. Open Visual Studio Express For Web.
2. Select **Open Web Site...** from the **FILE** menu.
3. In the **Open Web Site** dialog (Fig. 23.5), ensure that **File System** is selected, then navigate to this chapter's examples, select the `WebTime` folder and click the **Open** `Button`.
4. Select `WebTime.aspx` in the **Solution Explorer**, then type *Ctrl* + *F5* to execute the web app.

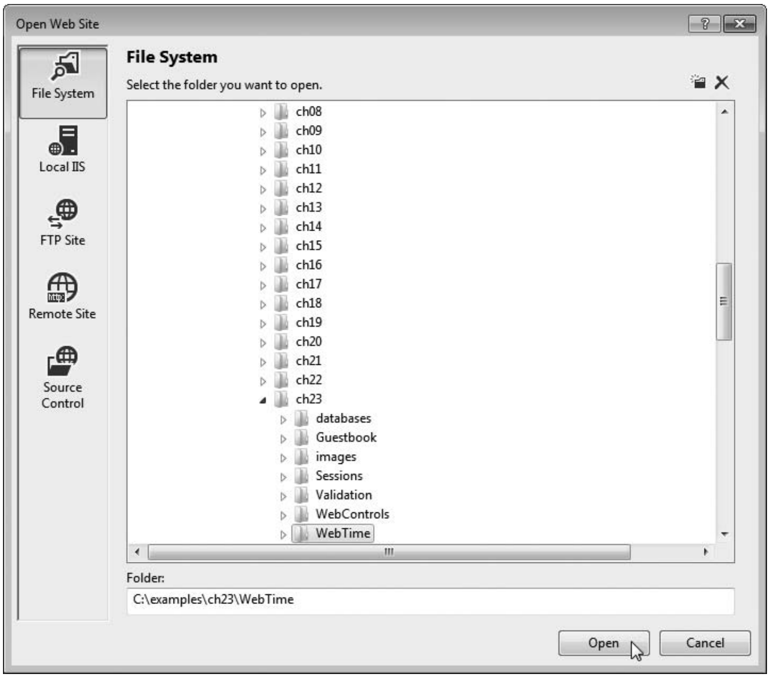


Fig. 23.5 | **Open Web Site** dialog.

Testing the App in a Selected Web Browser

If you wish to execute the app in another web browser, you can copy the web page's address from your default browser's address field and paste it into another browser's address field, or you can perform the following steps:

1. In the **Solution Explorer**, right click `WebTime.aspx` and select **Browse With...** to display the **Browse With** dialog (Fig. 23.6).

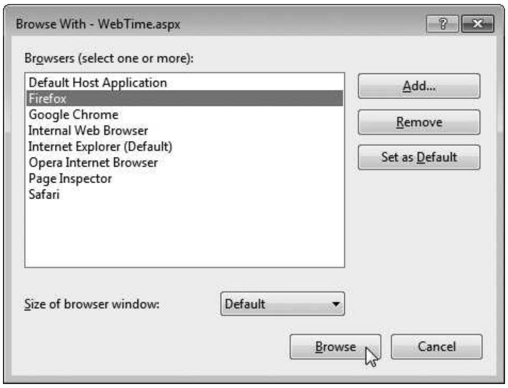


Fig. 23.6 | Selecting another web browser to execute the web app.

2. From the **Browsers** list, select the browser in which you'd like to test the web app and click the **Browse** `Button`.

If the browser you wish to use is not listed, you can use the **Browse With** dialog to add items to or remove items from the list of web browsers.

23.4.1 Building the `WebTime` App

Now that you've tested the app, let's create it in Visual Studio Express For Web.

Step 1: Creating the Web Site Project

Select **FILE > New Web Site...** to display the **New Web Site** dialog (Fig. 23.7). In the left column of this dialog, ensure that **Visual C#** is selected, then select **ASP.NET Empty Web Site** in the middle column. At the bottom of the dialog you can specify the location and name of the web app.

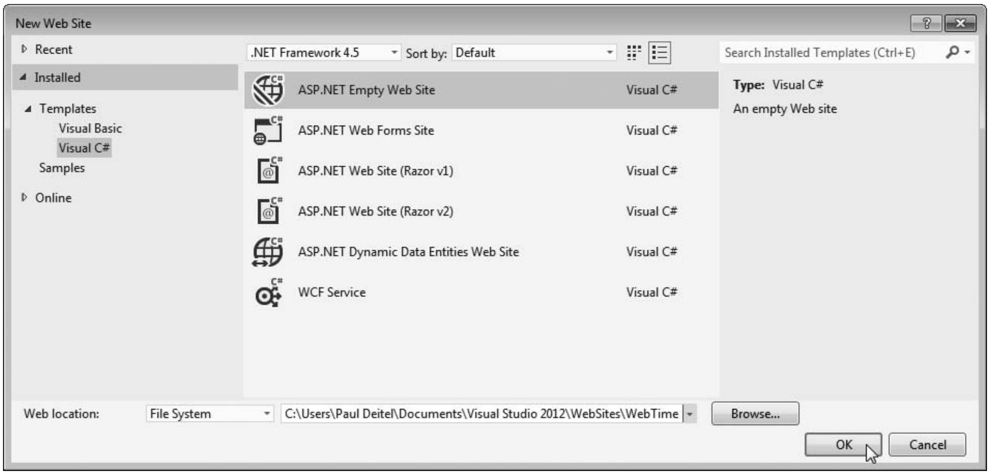

New Web Site
Recent
Installed
Templates
Visual Basic
Visual C#
Samples
Online
.NET Framework 4.5
Sort by: Default
Search Installed Templates (Ctrl+E)
ASP.NET Empty Web Site Visual C#
ASP.NET Web Forms Site Visual C#
ASP.NET Web Site (Razor v1) Visual C#
ASP.NET Web Site (Razor v2) Visual C#
ASP.NET Dynamic Data Entities Web Site Visual C#
WCF Service Visual C#
Type: Visual C#
An empty Web site
Web location:
File System
C:\Users\Paul Deitel\Documents\Visual Studio 2012\WebSites\WebTime
Browse...
OK
Cancel

Fig. 23.7 | Creating an **ASP.NET Web Site** in Visual Studio Express For Web.

The **Web location:** `ComboBox` provides the following options:

- **File System**: Creates a new website for testing on your local computer. Such websites execute on your local machine in **IIS Express** and can be accessed only by web browsers running on the same computer. IIS Express is a version of Microsoft's Internet Information Services (IIS) web server that allows you to test your web apps locally. You can later publish your website to a production IIS web server for access via a local network or the Internet. Each example in this chapter uses the **File System** option, so select it now.
- **HTTP**: Creates a new website on an IIS web server and uses HTTP to allow you to put your website's files on the server. IIS is Microsoft's software that's used to run production websites. If you own a website and have your own web server, you might use this to build a new website directly on that server computer. You must be an Administrator on the computer running IIS to use this option.

- **FTP**: Uses File Transfer Protocol (FTP) to allow you to put your website's files on the server. The server administrator must first create the website on the server for you. FTP is commonly used by so-called "hosting providers" to allow website owners to share a server computer that runs many websites.

Change the name of the web app from `WebSite1` to `WebTime`, then click **OK** to create the website.

Step 2: Adding a Web Form to the Website and Examining the **Solution Explorer**

A **Web Form** represents one page in a web app—we'll often use the terms "page" and "Web Form" interchangeably. A Web Form contains a web app's GUI. To create the `WebTime.aspx` Web Form:

1. Right click the project name in the **Solution Explorer** and select **Add > Add New Item...** to display the **Add New Item** dialog (Fig. 23.8).

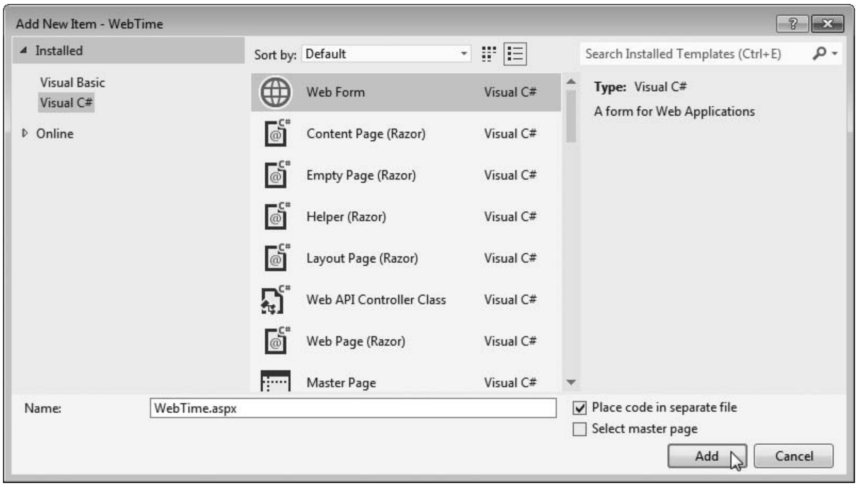
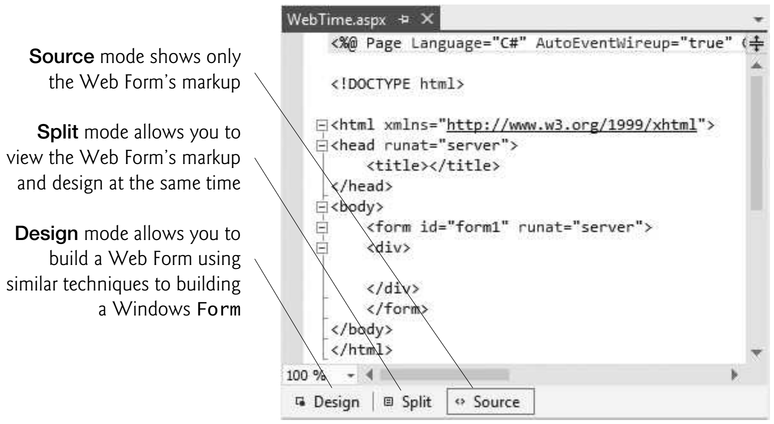

Add New Item - WebTime
Installed
Visual Basic
Visual C#
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
Web Form Visual C#
Content Page (Razor) Visual C#
Empty Page (Razor) Visual C#
Helper (Razor) Visual C#
Layout Page (Razor) Visual C#
Web API Controller Class Visual C#
Web Page (Razor) Visual C#
Master Page Visual C#
Type: Visual C#
A form for Web Applications
Name: WebTime.aspx
Place code in separate file
Select master page
Add
Cancel

Fig. 23.8 | Adding a new **Web Form** to the website with the **Add New Item** dialog.

2. In the left column, ensure that **Visual C#** is selected, then select **Web Form** in the middle column.
3. In the **Name:** `TextBox`, change the file name to `WebTime.aspx`, then click the **Add** `Button`.

After you add the Web Form, the IDE opens it in **Source** view by default (Fig. 23.9). This view displays the markup for the Web Form. As you become more familiar with ASP.NET and building web sites in general, you might use **Source** view to perform high precision adjustments to your design (in HTML and/or CSS) or to program in the JavaScript language that executes in web browsers. For the purposes of this chapter, we'll keep things simple by working exclusively in **Design** mode. To switch to **Design** mode, you can click the **Design** `Button` at the bottom of the code editor window.


Source mode shows only the Web Form's markup
Split mode allows you to view the Web Form's markup and design at the same time
Design mode allows you to build a Web Form using similar techniques to building a Windows `Form`
WebTime.aspx
100 %
Design
Split
Source

```
<%@ Page Language="C#" AutoEventWireup="true" (

<!DOCTYPE html>

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

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

Fig. 23.9 | Web Form in **Source** view.

The Solution Explorer

The **Solution Explorer** (Fig. 23.10) shows the contents of the website. We expanded the node for `WebTime.aspx` to show you its *code-behind file* `WebTime.aspx.cs`. Visual Studio Express For Web's **Solution Explorer** contains a **Nest Related Files** button that organizes each Web Form and its code-behind file.

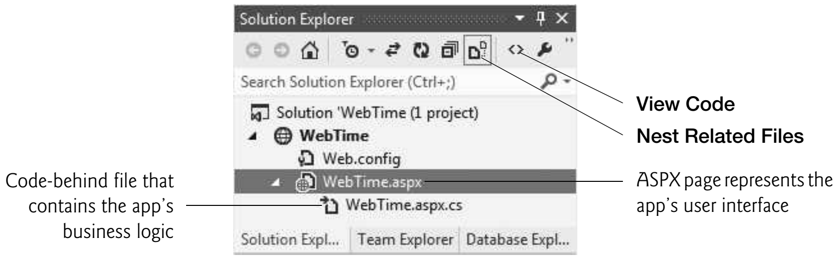

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'WebTime (1 project)
WebTime
Web.config
WebTime.aspx
WebTime.aspx.cs
Solution Expl...
Team Explorer
Database Expl...
View Code
Nest Related Files
ASPX page represents the app's user interface
Code-behind file that contains the app's business logic

Fig. 23.10 | **Solution Explorer** window for an **Empty Web Site** project after adding the Web Form `WebTime.aspx`.

If the ASPX file is not open in the IDE, you can open it in **Design** mode by double clicking it in the **Solution Explorer** then selecting the **Design** tab, or by right clicking it in the **Solution Explorer** and selecting **View Designer**. To open the code-behind file in the code editor, you can double click it in the **Solution Explorer** or

- select the ASPX file in the **Solution Explorer**, then click the **View Code** (<>) `Button`
- right click the ASPX file in the **Solution Explorer**, then select **View Code**
- right click the code-behind file in the **Solution Explorer** and select **Open**

The Toolbox

Figure 23.11 shows the **Toolbox** displayed in the IDE when the project loads. Part (a) displays the beginning of the **Standard** list of web controls, and part (b) displays the remaining web controls and the list of other control groups. We discuss specific controls listed in Fig. 23.11 as they're used throughout the chapter. Many of the controls have similar or identical names to Windows Forms controls presented earlier in the book.

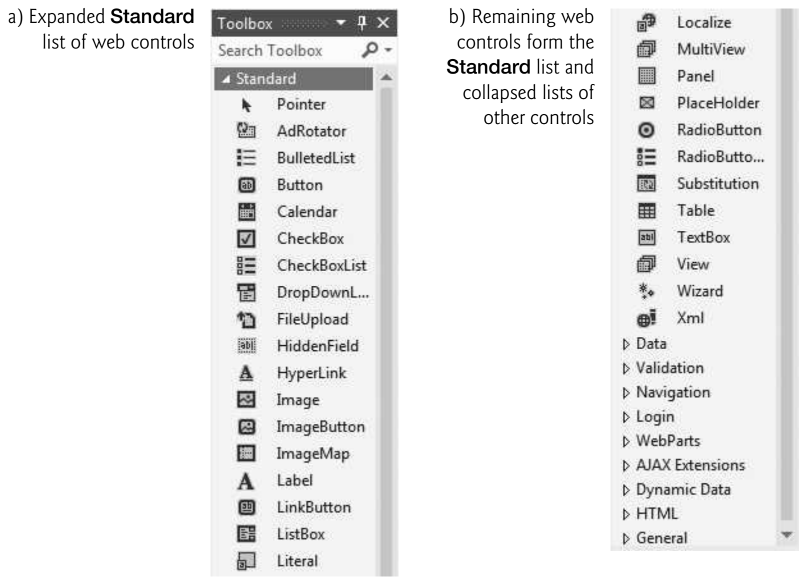


Fig. 23.11 | **Toolbox** in Visual Studio Express For Web.

The Web Forms Designer

Figure 23.12 shows the initial Web Form in **Design** mode. You can drag and drop controls from the **Toolbox** onto the Web Form. You can also type at the current cursor location to add so-called *static text* to the web page. In response to such actions, the IDE generates the appropriate markup in the ASPX file.

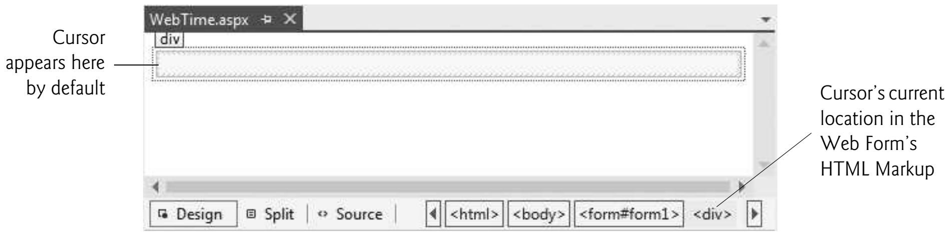

Cursor appears here by default
WebTime.aspx
div
Design
Split
Source
<html>
<body>
<form#form1>
<div>
Cursor's current location in the Web Form's HTML Markup

Fig. 23.12 | **Design** mode of the Web Forms Designer.

Step 3: Changing the Title of the Page

Before designing the Web Form's content, you'll change its title to `Simple Web Form Example`. This title is typically displayed in the web browser's title bar or on the browser tab that is displaying the page (see Fig. 23.4). It's also used by search engines like Google and

Bing when they index real websites for searching. Every page should have a title. To change the title:

1. Ensure that the ASPX file is open in **Design** view.
2. In the **Properties** window's drop-down list, view the Web Form's properties by selecting `DOCUMENT`, which represents the Web Form. A web page is often called a document.
3. Modify the page's `Title` property by setting it to `Simple Web Form Example`.

Designing a Page

Designing a Web Form is similar to designing a Windows `Form`. To add controls to the page, drag-and-drop them from the **Toolbox** onto the Web Form in **Design** view. The Web Form itself and the control's you add to the Web Form are objects that have properties, methods and events. You can set these properties visually using the **Properties** window, programmatically in the code-behind file or by editing the markup directly in the `.aspx` file. You can also type text directly on a Web Form at the cursor location.

Controls and other elements are placed sequentially on a Web Form one after another in the order in which you drag-and-drop them onto the Web Form. The cursor indicates the insertion point in the page. If you want to position a control between existing text or controls, you can drop the control at a specific position between existing page elements. You can also rearrange controls with drag-and-drop actions in **Design** view. The positions of controls and other elements are relative to the Web Form's upper-left corner. This type of layout is known as *relative positioning* and it allows the browser to move elements and resize them based on the size of the browser window. Relative positioning is the *default*, and we'll use it throughout this chapter.

For precise control over the location and size of elements, you can use absolute positioning in which controls are located exactly where you drop them on the Web Form. If you wish to use absolute positioning:

1. Select **TOOLS > Options...**, to display the **Options** dialog.
2. Expand the **HTML Designer** node, select the **CSS Styling** node and ensure that the checkbox labeled **Change positioning to absolute for controls added using Toolbox, paste or drag and drop** is selected.

Step 4: Adding Text and a `Label`

You'll now add some text and a `Label` to the Web Form. Perform the following steps to add the text:

1. Ensure that the Web Form is open in **Design** mode.
2. Type the following text at the current cursor location:

```
    Current time on the Web server:
```

3. Select the text you just typed, then select **Heading 1** from the **Block Format** `ComboBox` (Fig. 23.13) in the IDE's **Formatting** toolbar. This formats the text as a first-level heading that typically appears in a larger bold font. In more complex pages, headings help you specify the relative importance of parts of the content—like chapters in a book and sections in a chapter.

Block Format ComboBox on the **Formatting** toolbar

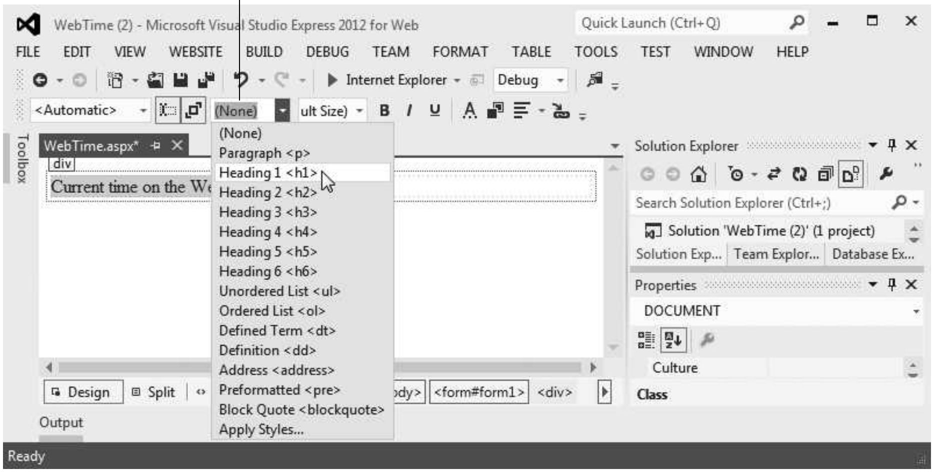


Fig. 23.13 | Changing the text to **Heading 1** heading.

4. Click to the right of the text you just typed and press the *Enter* key to start a new paragraph in the page. The Web Form should now appear as in Fig. 23.14.

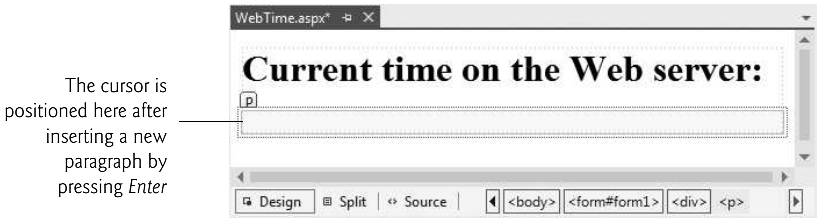

The cursor is positioned here after inserting a new paragraph by pressing *Enter*
WebTime.aspx*
Current time on the Web server:
p
Design Split Source
<body> <form#form1> <div> <p>

Fig. 23.14 | `WebTime.aspx` after inserting text and a new paragraph.

5. Next, drag a `Label` control from the **Toolbox** into the new paragraph or double click the `Label` control in the **Toolbox** to insert the `Label` at the current cursor position.
6. Using the **Properties** window, set the `Label`'s `(ID)` property to `timeLabel`. This specifies the variable name that will be used to *programmatically* change the `Label`'s `Text`.
7. Because, the `Label`'s `Text` will be set programmatically, delete the current value of the `Label`'s `Text` property. When a `Label` does not contain text, its name is displayed in square brackets in **Design** view (Fig. 23.15) as a placeholder for design and layout purposes. This text is not displayed at execution time.

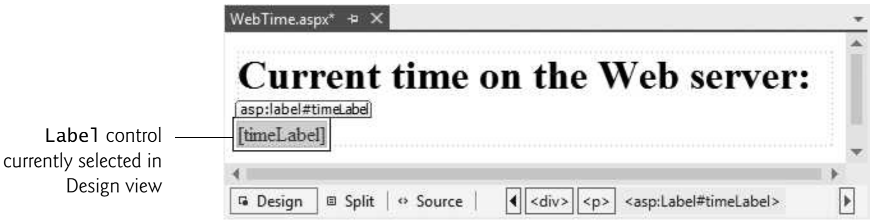


Fig. 23.15 | `WebTime.aspx` after adding a `Label`.

Step 5: Formatting the `Label`

Formatting in a web page is performed with *CSS (Cascading Style Sheets)*. The details of CSS are beyond the scope of this book. However, it's easy to use CSS to format text and elements in a Web Form via the tools built into Visual Studio Express For Web. In this example, we'd like to change the `Label`'s *background color* to black, its *foreground color* to yellow and make its *text size* larger. To format the `Label`, perform the following steps:

1. Click the `Label` in **Design** view to ensure that it's selected.
2. Select **VIEW > CSS Properties** to display the **CSS Properties** window at the left side of the IDE (Fig. 23.16).

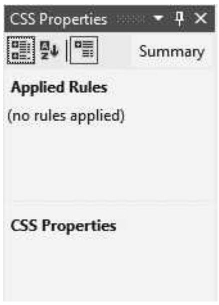


Fig. 23.16 | **CSS Properties** window.

3. Right click below **Applied Rules** and select **New Style...** to display the **New Style** dialog (Fig. 23.17).
4. Type the new style's name in the **Selector:** `ComboBox`—we chose `.timeStyle` since this will be the style used to format the time that's displayed in the page. Styles that apply to specific elements must be named with a dot (`.`) preceding the name. Such a style is called a **CSS class**.
5. Each item you can set in the **New Style** dialog is known as a *CSS attribute*. To change `timeLabel`'s *foreground color*, select the **Font** category from the **Category** list, then select the yellow color swatch for the **color** attribute.
6. Next, change the **font-size** attribute to `xx-large`.

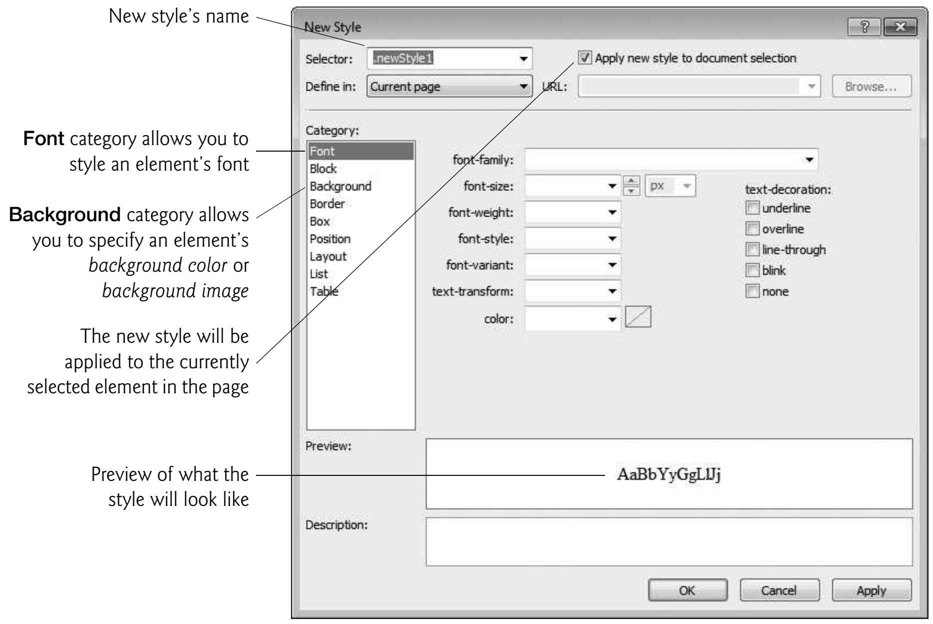


Fig. 23.17 | **New Style** dialog.

7. To change `timeLabel`'s *background color*, select the **Background** category, then select the black color swatch for the **background-color** attribute.

The **New Style** dialog should now appear as shown in Fig. 23.18. Click the **OK** `Button` to apply the style to the `timeLabel` so that it appears as shown in Fig. 23.19. Also, notice that the `Label`'s *`CssClass` property* is now set to `timeStyle` in the **Properties** window.

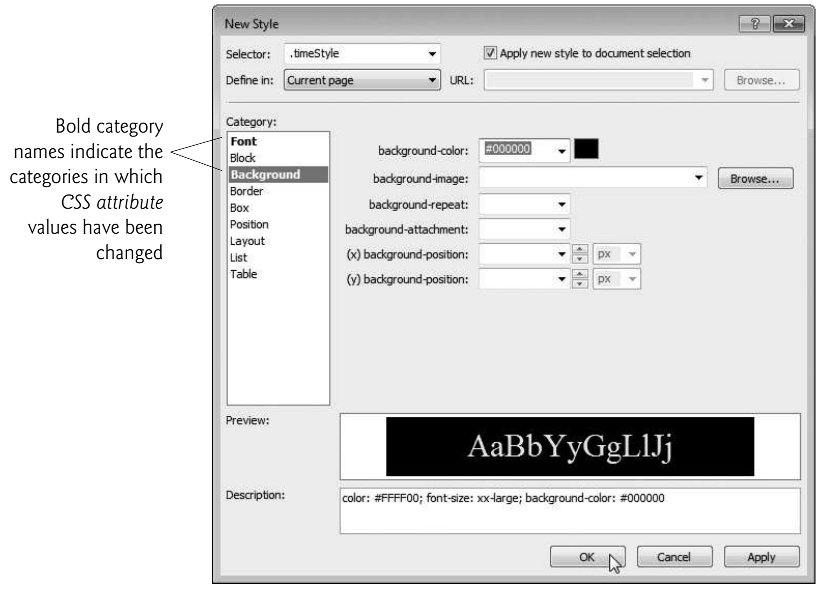
Step 6: Adding Page Logic

Now that you've designed the GUI, you'll write code in the *code-behind file* to obtain the server's time and display it on the `Label`. Open `WebTime.aspx.cs` by double clicking it in the **Solution Explorer**. In this example, you'll add an event handler to the code-behind file to handle the Web Form's **`Init` event**, which occurs when the page is requested by a web browser. The event handler for this event—named **`Page_Init`**—initializes the page. The only initialization required for this example is to set the `timeLabel`'s `Text` property to the time on the web server computer. The code-behind file initially contains a `Page_Load` event handler. To create the `Page_Init` event handler, simply rename `Page_Load` as `Page_Init`. Then complete the event handler by inserting the following code in its body:

```
    // display the server's current time in timeLabel
    timeLabel.Text = DateTime.Now.ToString("hh:mm:ss");
```

Step 7: Setting the **Start Page** *and Running the Program*

To ensure that `WebTime.aspx` loads when you execute this app, right click it in the **Solution Explorer** and select **Set As Start Page**. You can now run the program in one of several ways. At the beginning of Fig. 23.4, you learned how to view the Web Form by typing *Ctrl* + *F5*.



color: #FFFF00; font-size: xx-large; background-color: #000000

Fig. 23.18 | **New Style** dialog after changing the Label's style.

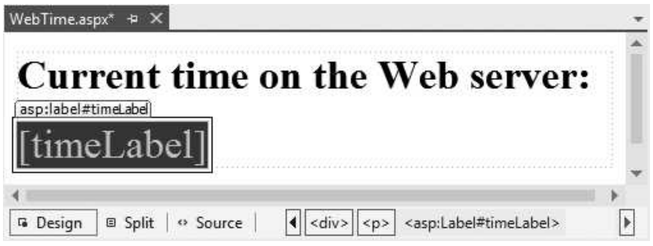


Fig. 23.19 | **Design** view after changing the `Label`'s style.

You can also right click an ASPX file in the **Solution Explorer** and select **View in Browser**. Both of these techniques execute *IIS Express*, open your *default web browser* and load the page into the browser, thus running the web app. IIS Express stops when you exit Visual Studio Express For Web.

If problems occur when running your app, you can run it in *debug mode* by selecting **DEBUG > Start Debugging**, by clicking the **Start Debugging** `Button` (▶) or by typing *F5* to view the web page in a web browser with *debugging enabled*. You cannot debug a web app unless debugging is *explicitly* enabled in the app's **`Web.config`** file—a file that's generated when you create an ASP.NET web app. This file stores the app's configuration settings. You'll rarely need to manually modify `Web.config`. The first time you select **DEBUG > Start Debugging** in a project, a dialog appears and asks whether you want the IDE to modify the `Web.config` file to enable debugging. After you click **OK**, the IDE executes the app. You can stop debugging by selecting **DEBUG > Stop Debugging**.

Regardless of how you execute the web app, the IDE will compile the project before it executes. In fact, ASP.NET compiles your web page whenever it changes between HTTP requests. For example, suppose you browse the page, then modify the ASPX file or add code to the code-behind file. When you reload the page, ASP.NET recompiles the page on the server before returning the response to the browser. This important behavior ensures that clients always see the latest version of the page. You can manually compile an entire website by selecting **Build Web Site** from the **DEBUG** menu in Visual Studio Express For Web.

23.4.2 Examining `WebTime.aspx`'s Code-Behind File

Figure 23.20 presents the code-behind file `WebTime.aspx.cs`. Line 5 begins the declaration of class `WebTime`. A class declaration can span multiple source-code files—the separate portions of the class declaration in each file are known as **partial classes**. The **`partial` modifier** indicates that the code-behind file is part of a larger class. Like Windows `Forms` apps, the rest of the class's code is generated for you based on your visual interactions to create the app's GUI in **Design** mode. That code is stored in other source code files as partial classes with the same name. The compiler assembles all the partial classes that have the same into a single class declaration.

```
 1  // Fig. 23.20: WebTime.aspx.cs
 2  // Code-behind file for a page that displays the web server's time.
 3  using System;
 4
 5  public partial class WebTime : System.Web.UI.Page
 6  {
 7     // initializes the contents of the page
 8     protected void Page_Init( object sender, EventArgs e )
 9     {
10        // display the server's current time in timeLabel
11        timeLabel.Text = DateTime.Now.ToString( "hh:mm:ss" );
12     } // end method Page_Init
13  } // end class WebTime
```

Fig. 23.20 | Code-behind file for a page that displays the web server's time.

Line 5 indicates that `WebTime` inherits from class **`Page`** in namespace **`System.Web.UI`**. This namespace contains classes and controls for building web-based apps. Class `Page` represents the default capabilities of each page in a web app—all pages inherit directly or indirectly from this class.

Lines 8–12 define the `Page_Init` event handler, which initializes the page in response to the page's `Init` event. The only initialization required for this page is to set the `timeLabel`'s `Text` property to the time on the web server computer. The statement in line 11 retrieves the current time (`DateTime.Now`) and formats it as *hh*`:`*mm*`:`*ss*. For example, 9 AM is formatted as 09:00:00, and 2:30 PM is formatted as 02:30:00. As you'll see, variable `timeLabel` represents an *ASP.NET `Label` control*. The ASP.NET controls are defined in namespace **`System.Web.UI.WebControls`**.

23.5 Standard Web Controls: Designing a Form

This section introduces some of the web controls located in the **Standard** section of the **Toolbox** (Fig. 23.11). Figure 23.21 summarizes the controls used in the next example.

Web control	Description
`TextBox`	Gathers user input and displays text.
`Button`	Triggers an event when clicked.
`HyperLink`	Displays a hyperlink.
`DropDownList`	Displays a drop-down list of choices from which a user can select an item.
`RadioButtonList`	A group of radio buttons.
`Image`	Displays images (for example, PNG, GIF and JPG).

Fig. 23.21 | Commonly used web controls.

A Form Gathering User Input

Figure 23.22 depicts a form for gathering user input. This example does not perform any tasks—that is, no action occurs when the user clicks **Register**. As an exercise, we ask you to provide the functionality. Here we focus on the steps for adding these controls to a Web Form and for setting their properties. Subsequent examples demonstrate how to handle the events of many of these controls. To execute this app:

1. Select **Open Web Site...** from the **FILE** menu.
2. In the **Open Web Site** dialog, ensure that **File System** is selected, then navigate to this chapter's examples, select the `WebControls` folder and click the **Open** `Button`.
3. Select `WebControls.aspx` in the **Solution Explorer**, then type *Ctrl* + *F5* to execute the web app in your default web browser.

Step 1: Creating the Web Site

To begin, follow the steps in Section 23.4.1 to create an **Empty Web Site** named `WebControls`, then add a Web Form named `WebControls.aspx` to the project. Set the document's `Title` property to `"Web Controls Demonstration"`. Right click `WebControls.aspx` in the **Solution Explorer** and select **Set As Start Page** to ensure that this page loads when you execute the app.

Step 2: Adding the Images to the Project

The images used in this example are located in the `images` folder with this chapter's examples. Before you can display this app's images in the Web Form, they must be added to your project. To add the `images` folder to your project:

1. Open Windows Explorer.
2. Locate and open this chapter's examples folder (`ch23`).
3. Drag the `images` folder from Windows Explorer into Visual Studio Express For Web's **Solution Explorer** window and drop the folder on the name of your project.

The IDE will automatically copy the folder and its contents into your project.

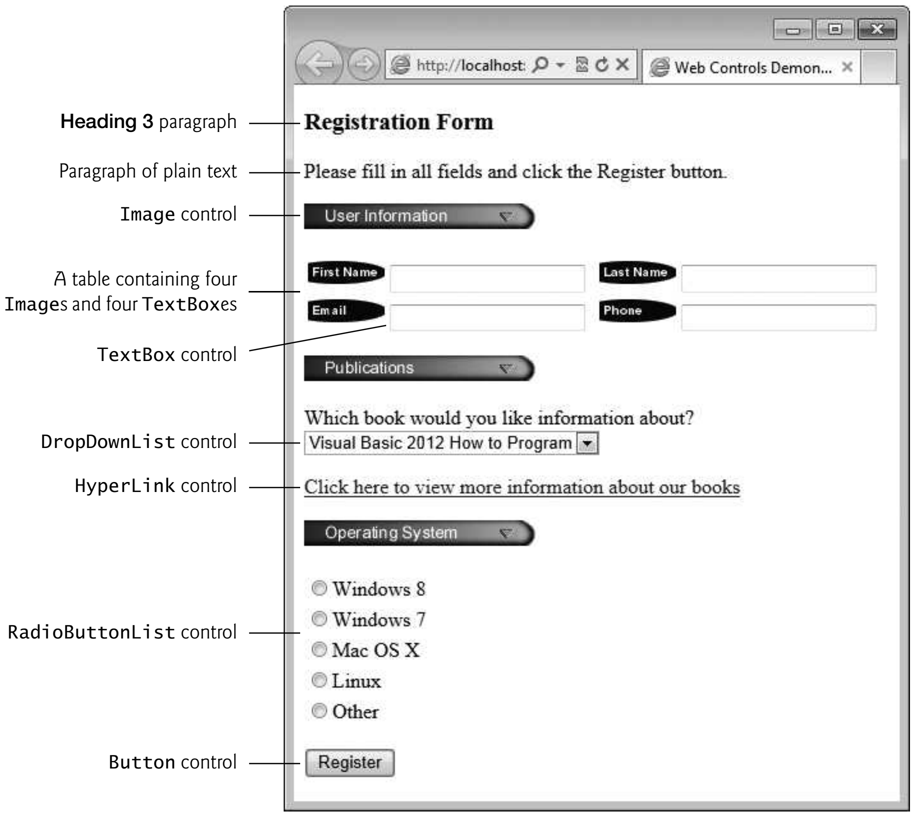


Fig. 23.22 | Web Form that demonstrates web controls.

Step 3: Adding Text and an Image to the Form

Next, you'll begin creating the page. Perform the following steps:

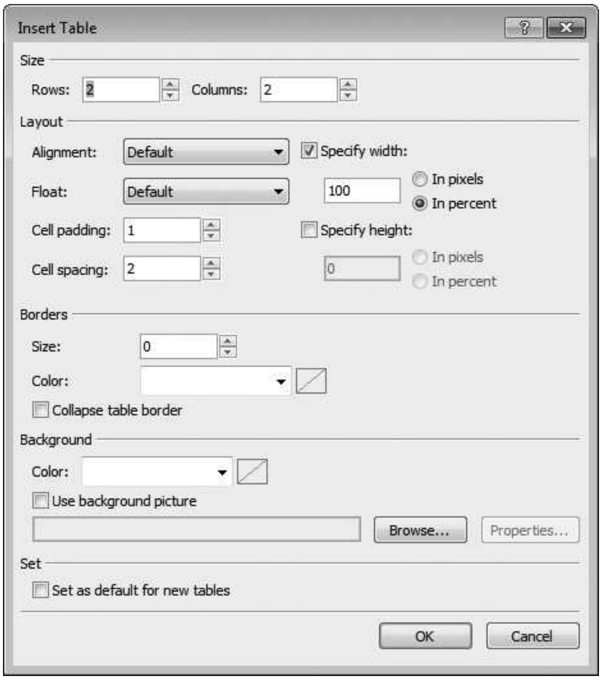
1. First create the page's heading. At the current cursor position on the page, type the text `"Registration Form"`, then use the **Block Format** `ComboBox` in the IDE's toolbar to change the text to **Heading 3** format.
2. Press *Enter* to start a new paragraph, then type the text `"Please fill in all fields and click the Register button."`.
3. Press *Enter* to start a new paragraph, then double click the **`Image`** control in the `Toolbox`. This control inserts an image into a web page, at the current cursor position. Set the `Image`'s `(ID)` property to `userInformationImage`. The **`ImageUrl`** property specifies the location of the image to display. In the **Properties** window, click the ellipsis for the `ImageUrl` property to display the **Select Image** dialog. Select the `images` folder under **Project folders:** to display the list of images. Then select the image `user.png`.
4. Click **OK** to display the image in **Design** view, then click to the right of the `Image` and press *Enter* to start a new paragraph.

Step 4: Adding a Table to the Form

Form elements are sometimes placed in tables for layout purposes—like the elements that represent the first name, last name, e-mail and phone information in Fig. 23.22. Next, you'll create a table with two rows and two columns in **Design** mode.

1. Select **Table > Insert Table** to display the **Insert Table** dialog (Fig. 23.23). This dialog allows you to configure the table's options.
2. Under **Size**, ensure that the values of **Rows** and **Columns** are both `2`—these are the default values.
3. Click **OK** to close the **Insert Table** dialog and create the table.

By default, the contents of a table cell are aligned *vertically* in the *middle* of the cell.



Rows: 2 Columns: 2
Alignment: Default
Float: Default
Cell padding: 1
Cell spacing: 2
[✓] Specify width: 100 ( ) In pixels [● In percent]
[ ] Specify height: 0 ( ) In pixels ( ) In percent
Size: 0
Color:
[ ] Collapse table border
Color:
[ ] Use background picture
[ ] Set as default for new tables

Fig. 23.23 | **Insert Table** dialog.

After creating the table, controls and text can be added to particular cells to create a neatly organized layout. Next, add `Image` and `TextBox` controls to each the four table cells as follows:

1. Click the table cell in the first row and first column of the table, then double click the `Image` control in the **Toolbox**. Set its `(ID)` property to `firstNameImage` and set its `ImageUrl` property to the image `fname.png`.
2. Next, double click the `TextBox` control in the **Toolbox**. Set its `(ID)` property to `firstNameTextBox`. As in Windows Forms, a **`TextBox`** control allows you to obtain text from the user and display text to the user.

3. Repeat this process in the first row and second column, but set the `Image`'s `(ID)` property to `lastNameImage` and its `ImageUrl` property to the image `lname.png`, and set the `TextBox`'s `(ID)` property to `lastNameTextBox`.
4. Repeat *Steps 1* and *2* in the second row and first column, but set the `Image`'s `(ID)` property to `emailImage` and its `ImageUrl` property to the image `email.png`, and set the `TextBox`'s `(ID)` property to `emailTextBox`.
5. Repeat *Steps 1* and *2* in the second row and second column, but set the `Image`'s `(ID)` property to `phoneImage` and its `ImageUrl` property to the image `phone.png`, and set the `TextBox`'s `(ID)` property to `phoneTextBox`.

Step 5: Creating the **Publications** *Section of the Page*

This section contains an `Image`, some text, a `DropDownList` control and a `HyperLink` control. Perform the following steps to create this section:

1. Click below the table, then use the techniques you've already learned in this section to add an `Image` named `publicationsImage` that displays the `publications.png` image.
2. Click to the right of the `Image`, then press *Enter* and type the text `"Which book would you like information about?"` in the new paragraph.
3. Hold the *Shift* key and press *Enter* to create a new line in the current paragraph, then double click the **`DropDownList`** control in the **Toolbox**. Set its `(ID)` property to `booksDropDownList`. This control is similar to the Windows `Forms ComboBox` control, but doesn't allow users to type text. When a user clicks the drop-down list, it expands and displays a list from which the user can make a selection.
4. You can add items to the `DropDownList` using the **ListItem Collection Editor** by clicking the ellipsis next to the `DropDownList`'s `Items` property in the **Properties** window, or by using the **DropDownList Tasks** smart-tag menu. To open this menu, click the small arrowhead that appears in the upper-right corner of the control in **Design** mode (Fig. 23.24). Visual Studio Express 2012 for Web displays smart-tag menus for many ASP.NET controls to facilitate common tasks. Clicking **Edit Items...** in the **DropDownList Tasks** menu opens the **ListItem Collection Editor**, which allows you to add `ListItem` elements to the `DropDownList`. Add items for `"Visual Basic 2012 How to Program"`, `"Visual C# 2012 How to Program"`, `"Java`

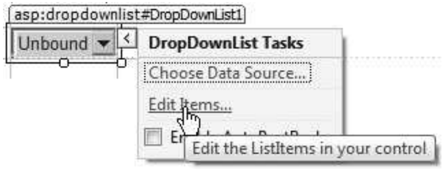

asp:dropdownlist#DropDownList1
Unbound
DropDownList Tasks
Choose Data Source...
Edit Items...
Edit the ListItems in your control

Fig. 23.24 | **DropDownList Tasks** smart-tag menu.

How to Program" and "C++ How to Program" by clicking the **Add** Button four times. For each item, select it, then set its Text property to one of the four book titles.

5. Click to the right of the DropDownList and press *Enter* to start a new paragraph, then double click the **HyperLink** control in the **Toolbox** to add a hyperlink to the web page. Set its (ID) property to booksHyperLink and its Text property to "Click here to view more information about our books". Set the **NavigateUrl** property to http://www.deitel.com. This specifies the resource or web page that will be requested when the user clicks the HyperLink. Setting the **Target** property to _blank specifies that the requested web page should open in a new tab or browser window. By default, HyperLink controls cause pages to open in the *same* browser window.

Step 6: Completing the Page
Next you'll create the **Operating System** section of the page and the **Register** Button. This section contains a **RadioButtonList** control, which provides a series of radio buttons from which the user can select only one. The **RadioButtonList Tasks** smart-tag menu provides an **Edit Items...** link to open the **ListItem Collection Editor** so that you can create the items in the list. Perform the following steps:

1. Click to the right of the HyperLink control and press *Enter* to create a new paragraph, then add an Image named osImage that displays the os.png image.
2. Click to the right of the Image and press *Enter* to create a new paragraph, then add a RadioButtonList. Set its (ID) property to osRadioButtonList. Use the **ListItem Collection Editor** to add the items shown in Fig. 23.22.
3. Finally, click to the right of the RadioButtonList and press *Enter* to create a new paragraph, then add a **Button**. A Button web control represents a button that triggers an action when clicked. Set its (ID) property to registerButton and its Text property to Register. As stated earlier, clicking the **Register** button in this example does not do anything.

You can now execute the app (*Ctrl* + *F5*) to see the Web Form in your browser.

23.6 Validation Controls

This section introduces a different type of web control, called a **validation control** or **validator**, which determines whether the *data* in another web control is in the proper *format*. For example, validators can determine whether a user has provided information in a required field or whether a zip-code field contains exactly five digits. Validators provide a mechanism for *validating user input* on the client and the server. When the page is sent to the client, the validator is converted into JavaScript that performs the validation in the client web browser. JavaScript is a scripting language that enhances the functionality of web pages and is typically executed on the client. Unfortunately, some client browsers might not support scripting or the user might disable it. For this reason, you should *always perform validation on the server*. ASP.NET validation controls can function on the *client*, on the *server* or *both*.

Validating Input in a Web Form
The Web Form in Fig. 23.25 prompts the user to enter a name, e-mail address and phone number. A website could use a form like this to collect contact information from visitors.

After the user enters any data, but before the data is sent to the web server, validators ensure that the user *entered a value in each field* and that the e-mail address and phone-number values are in an acceptable format. In this example, (555) 123-4567, 555-123-4567 and 123-4567 are all considered valid phone numbers. Once the data is submitted, the web server responds by displaying a message that repeats the submitted information. A real business app would typically store the submitted data in a database or in a file on the server. We simply send the data back to the client to demonstrate that the server received the data. To execute this app:

1. Select **Open Web Site...** from the **FILE** menu.
2. In the **Open Web Site** dialog, ensure that **File System** is selected, then navigate to this chapter's examples, select the `Validation` folder and click the **Open** `Button`.
3. Select `Validation.aspx` in the **Solution Explorer**, then type *Ctrl* + *F5* to execute the web app in your default web browser.

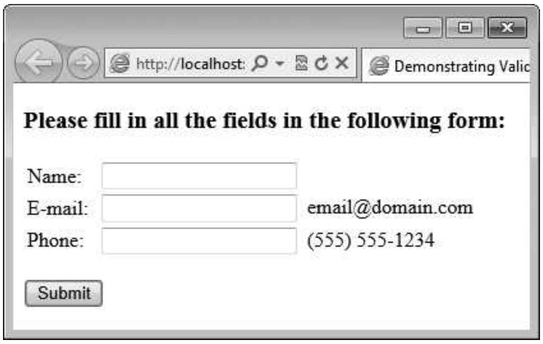
In the sample output:

- Fig. 23.25(a) shows the initial Web Form.
- Fig. 23.25(b) shows the result of submitting the form before typing any data in the `TextBoxes`.
- Fig. 23.25(c) shows the results after entering data in each `TextBox`, but specifying an invalid e-mail address and invalid phone number.
- Fig. 23.25(d) shows the results after entering valid values for all three `TextBoxes` and submitting the form.

Step 1: Creating the Web Site

To begin, follow the steps in Section 23.4.1 to create an **Empty Web Site** named `Validation`, then add a Web Form named `Validation.aspx` to the project. Set the document's `Title` property to `"Demonstrating Validation Controls"`. To ensure that `Validation.aspx` loads when you execute this app, right click it in the **Solution Explorer** and select **Set As Start Page**.

a) Initial Web Form



Please fill in all the fields in the following form:

Name:

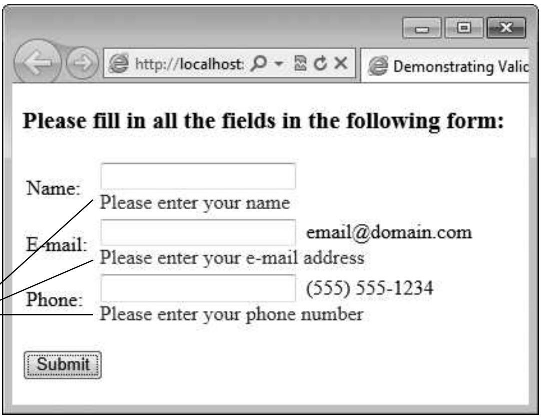
E-mail: email@domain.com

Phone: (555) 555-1234

Fig. 23.25 | Validators in a Web Form that retrieves user contact information. (Part 1 of 2.)

b) Web Form after the user presses the **Submit** `Button` without having entered any data in the `TextBox`es; each `TextBox` is followed by an error message that was displayed by a validation control

`RequiredFieldValidator` controls

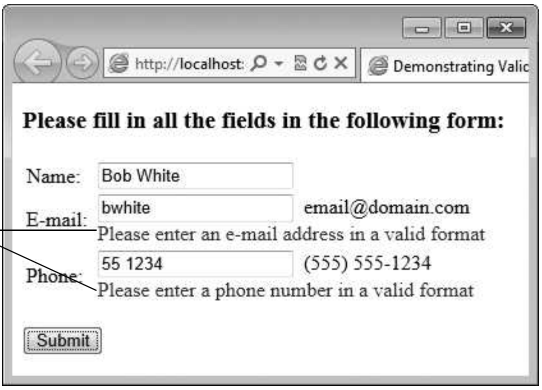


Please fill in all the fields in the following form:

Name:
Please enter your name
E-mail: email@domain.com
Please enter your e-mail address
Phone: (555) 555-1234
Please enter your phone number

c) Web Form after the user enters a name, an invalid e-mail address and an invalid phone number in the `TextBox`es, then presses the **Submit** `Button`; the validation controls display error messages in response to the invalid e-mail and phone number values

`RegularExpressionValidator` controls

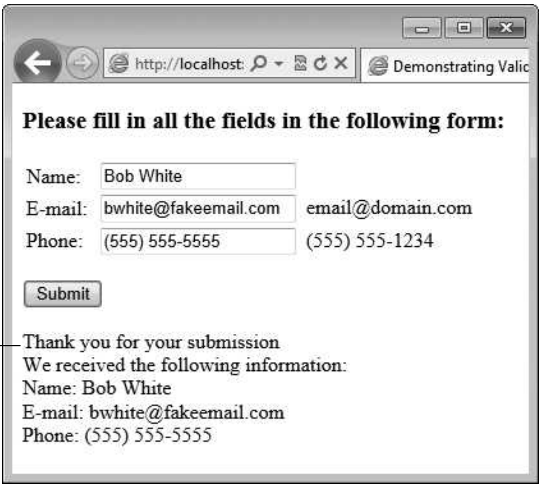


Please fill in all the fields in the following form:

Name: Bob White
E-mail: bwhite email@domain.com
Please enter an e-mail address in a valid format
Phone: 55 1234 (555) 555-1234
Please enter a phone number in a valid format

d) The Web Form after the user enters valid values for all three `TextBox`es and presses the **Submit** `Button`

`outputLabel` is displayed once the user provides valid form input and submits the form



Please fill in all the fields in the following form:

Name: Bob White
E-mail: bwhite@fakeemail.com email@domain.com
Phone: (555) 555-5555 (555) 555-1234

Thank you for your submission
We received the following information:
Name: Bob White
E-mail: bwhite@fakeemail.com
Phone: (555) 555-5555

Fig. 23.25 | Validators in a Web Form that retrieves user contact information. (Part 2 of 2.)

Step 2: Creating the GUI

To create the page, perform the following steps:

1. Type "Please fill in all the fields in the following form:", then use the **Block Format** ComboBox in the IDE's toolbar to change the text to **Heading 3** format and press *Enter* to create a new paragraph.
2. Insert a three-row and two-column table. You'll add elements to the table momentarily.
3. Click below the table and add a Button. Set its (ID) property to submitButton and its Text property to Submit. By default, a Button control in a Web Form sends the contents of the form back to the server for processing. Select the Button then use the **Block Format** ComboBox in the IDE's toolbar to wrap the Button in a **Paragraph** format—this places additional space above and below the Button.
4. Click to the right of the Button, then press *Enter* to create a new paragraph. Add a Label. Set its (ID) property to outputLabel and clear its Text property—you'll set it *programmatically* when the user clicks the submitButton. Set the outputLabel's **Visible** property to false, so the Label does *not* appear in the client's browser when the page loads for the *first* time. You'll *programmatically* display this Label after the user submits valid data.

Next you'll add text and controls to the table you created in *Step 2* above. Perform the following steps:

1. In the left column, type the text "Name:" in the first row, "E-mail:" in the second row and "Phone:" in the third row.
2. In the right column of the first row, add a TextBox and set its (ID) property to nameTextBox.
3. In the right column of the second row, add a TextBox and set its (ID) property to emailTextBox. Set its TextMode property to Email—this produces an HTML5 e-mail input field when the web page is rendered in the client web browser. Click to the right of the TextBox and type the text "email@domain.com" to show an example of what the user should enter in that TextBox.
4. In the right column of the third row, add a TextBox and set its (ID) property to phoneTextBox. Set its TextMode property to Phone—this produces an HTML5 phone input field when the web page is rendered in the client web browser. Click to the right of the TextBox and type the text "(555) 555-1234" to show an example of what the user should enter in that TextBox.

*Step 3: Using **RequiredFieldValidator** Controls*

We use three **RequiredFieldValidator** controls (found in the **Validation** section of the **Toolbox**) to ensure that the name, e-mail address and phone number TextBoxes are *not empty* when the form is submitted. A RequiredFieldValidator makes an *input control* a required field. If such a field is *empty*, validation fails. Add a RequiredFieldValidator as follows:

1. Click to the right of the nameTextBox in the table and press *Enter* to move to the next line.

2. Add a RequiredFieldValidator, set its (ID) to nameRequiredFieldValidator and set the ForeColor property to Red.
3. Set the validator's **ControlToValidate** property to nameTextBox to indicate that this validator verifies the nameTextBox's contents.
4. Set the validator's **ErrorMessage** property to "Please enter your name". This is displayed on the Web Form only if the validation fails.
5. Set the validator's **Display** property to Dynamic, so the validator occupies space on the Web Form only when validation fails. When this occurs, space is allocated dynamically, causing the controls below the validator to shift downward to accommodate the ErrorMessage, as seen in Fig. 23.25(a)–(c).

Repeat these steps to add two more RequiredFieldValidators in the second and third rows of the table. Set their (ID) properties to emailRequiredFieldValidator and phoneRequiredFieldValidator, respectively, and set their ErrorMessage properties to "Please enter your email address" and "Please enter your phone number", respectively.

*Step 4: Using **RegularExpressionValidator** Controls*

This example also uses two **RegularExpressionValidator** controls to ensure that the e-mail address and phone number entered by the user are in a valid format. Visual Studio Express 2012 for Web provides several *predefined* regular expressions that you can simply select to take advantage of this powerful validation control. Add a RegularExpressionValidator as follows:

1. Click to the right of the emailRequiredFieldValidator in the second row of the table and add a RegularExpressionValidator, then set its (ID) to emailRegularExpressionValidator and its ForeColor property to Red.
2. Set the ControlToValidate property to emailTextBox to indicate that this validator verifies the emailTextBox's contents.
3. Set the validator's ErrorMessage property to "Please enter an e-mail address in a valid format".
4. Set the validator's Display property to Dynamic, so the validator occupies space on the Web Form only when validation fails.

Repeat the preceding steps to add another RegularExpressionValidator in the third row of the table. Set its (ID) property to phoneRegularExpressionValidator and its ErrorMessage property to "Please enter a phone number in a valid format", respectively.

A RegularExpressionValidator's **ValidationExpression** property specifies the regular expression that validates the ControlToValidate's contents. Clicking the ellipsis next to property ValidationExpression in the **Properties** window displays the **Regular Expression Editor** dialog, which contains a list of **Standard expressions** for phone numbers, zip codes and other formatted information. For the emailRegularExpressionValidator, we selected the standard expression **Internet e-mail address**. If the user enters text in the emailTextBox that does not have the correct format and either clicks in a different text box or attempts to submit the form, the ErrorMessage text is displayed in *red*.

For the phoneRegularExpressionValidator, we selected **U.S. phone number** to ensure that a phone number contains an optional three-digit area code either in parentheses and followed by an optional space or without parentheses and followed by a required hyphen. After an optional area code, a phone number must contain three digits, a hyphen and another four digits. For example, `(555) 123-4567`, `555-123-4567` and `123-4567` are all valid phone numbers.

Submitting the Web Form's Contents to the Server

If all five validators are successful (that is, each `TextBox` is filled in, and the e-mail address and phone number provided are valid), clicking the **Submit** button sends the form's data to the server. As shown in Fig. 23.25(d), the server then responds by displaying the submitted data in the `outputLabel`.

Examining the Code-Behind File for a Web Form That Receives User Input

Figure 23.26 shows the code-behind file for this app. Notice that this code-behind file does not contain any implementation related to the validators. We'll say more about this soon. In this example, we respond to the page's **`Load`** event to process the data submitted by the user. Like the `Init` event, the `Load` event occurs each time the page loads into a web browser—the difference is that on a *postback* you cannot access the posted data in the controls from the `Init` handler. The event handler for this event is **`Page_Load`** (lines 8–33). The event handler for the `Load` event is created for you when you add a new Web Form. To complete the event handler, insert the code from Fig. 23.26.

```
 1  // Fig. 23.26: Validation.aspx.cs
 2  // Code-behind file for the form demonstrating validation controls.
 3  using System;
 4
 5  public partial class Validation : System.Web.UI.Page
 6  {
 7     // Page_Load event handler executes when the page is loaded
 8     protected void Page_Load( object sender, EventArgs e )
 9     {
10        // disable unobtrusive validation
11        UnobtrusiveValidationMode =
12           System.Web.UI.UnobtrusiveValidationMode.None;
13
14        // if this is not the first time the page is loading
15        // (i.e., the user has already submitted form data)
16        if ( IsPostBack )
17        {
18           Validate(); // validate the form
19
20           // if the form is valid
21           if ( IsValid )
22           {
23              // retrieve the values submitted by the user
24              string name = nameTextBox.Text;
25              string email = emailTextBox.Text;
26              string phone = phoneTextBox.Text;
27
```

Fig. 23.26 | Code-behind file for the form demonstrating validation controls. (Part 1 of 2.)

```
28              // show the the submitted values
29              outputLabel.Text = "Thank you for your submission<br/>" +
30                 "We received the following information:<br/>";
31              outputLabel.Text +=
32                 String.Format( "Name: {0}{1}E-mail:{2}{1}Phone:{3}",
33                    name, "<br/>", email, phone);
34              outputLabel.Visible = true; // display the output message
35           } // end if
36        } // end if
37     } // end method Page_Load
38  } // end class Validation
```

Fig. 23.26 | Code-behind file for the form demonstrating validation controls. (Part 2 of 2.)

ASP.NET 4.5 Unobtrusive Validation

Prior to ASP.NET 4.5, when you used the validation controls presented in this section, ASP.NET would embed substantial amounts of JavaScript code in a web page to perform the work of the validation controls in the client web browser. ASP.NET 4.5 now uses **unobtrusive validation**, which significantly reduces the amount of JavaScript that gets embedded into the web page—this, in turn, can improve the performance of your website by making pages load faster. When you create an ASP.NET Web Forms website, everything you need for unobtrusive validation is normally configured for you—*unless* you create an *ASP.NET Empty Web Site*, as we've done for the examples in this chapter so far. To enable this app to execute correctly in the web browser, lines 11–12 *disable* unobtrusive validation. In the online Chapter 29, Web App Development with ASP.NET: A Deeper Look, we'll create a website in which unobtrusive validation is properly enabled.

Differentiating Between the First Request to a Page and a Postback

Web programmers using ASP.NET often design their web pages so that the current page *reloads* when the user *submits* the form; this enables the program to receive input, process it as necessary and display the results in the same page when it's loaded the second time. These pages usually contain a form that, when submitted, sends the values of all the controls to the server and causes the current page to be requested again. This event is known as a **postback**. Line 16 uses the **`IsPostBack`** property of class `Page` to determine whether the page is being loaded due to a postback. The first time that the web page is requested, `IsPostBack` is `false`, and the page displays only the form for user input. When the postback occurs (from the user clicking **Submit**), `IsPostBack` is `true`.

Server-Side Web Form Validation

Server-side Web Form validation must be implemented *programmatically*. Line 18 calls the current `Page`'s **`Validate`** method to validate the information in the request. This validates the information as specified by the validation controls in the Web Form. Line 21 uses the **`IsValid`** property of class `Page` to check whether the validation succeeded. If this property is set to `true` (that is, validation succeeded and the Web Form is valid), then we display the Web Form's information. Otherwise, the web page loads without any changes, except that any validator that failed now displays its `ErrorMessage`.

Processing the Data Entered by the User
Lines 24–26 retrieve the values of `nameTextBox`, `emailTextBox` and `phoneTextBox`. When data is posted to the web server, the data that the user entered is accessible to the web app through the web controls' properties. Next, lines 29–33 set `outputLabel`'s `Text` to display a message that includes the name, e-mail and phone information that was submitted to the server. In lines 29, 30 and 33, notice the use of `<br/>` rather than `\n` to start new lines in the `outputLabel`—`<br/>` is the markup for a line break in a web page. Line 34 sets the `outputLabel`'s `Visible` property to `true`, so the user can see the thank-you message and submitted data when the page reloads in the client web browser.

23.7 Session Tracking
Originally, critics accused the Internet and e-businesses of failing to provide the customized service typically experienced in "brick-and-mortar" stores. To address this problem, businesses established mechanisms by which they could *personalize* users' browsing experiences, tailoring content to individual users. Businesses can achieve this level of service by tracking each customer's movement through the Internet and combining the collected data with information provided by the consumer, including billing information, personal preferences, interests and hobbies.

Personalization
Personalization makes it possible for businesses to communicate effectively with their customers and also improves users' ability to locate desired products and services. Companies that provide content of particular interest to users can establish relationships with customers and build on those relationships over time. Furthermore, by targeting consumers with personal offers, recommendations, advertisements, promotions and services, businesses create customer loyalty. Websites can use sophisticated technology to allow visitors to customize home pages to suit their individual needs and preferences. Similarly, online shopping sites often store personal information for customers, tailoring notifications and special offers to their interests. Such services encourage customers to visit sites more frequently and make purchases more regularly.

Privacy
A trade-off exists between personalized business service and protection of privacy. Some consumers embrace tailored content, but others fear the possible adverse consequences if the info they provide to businesses is released or collected by tracking technologies. Consumers and privacy advocates ask: What if the business to which we give personal data sells or gives that information to other organizations without our knowledge? What if we do not want our actions on the Internet—a supposedly anonymous medium—to be tracked and recorded by unknown parties? What if unauthorized parties gain access to sensitive private data, such as credit-card numbers or medical history? These are questions that must be addressed by programmers, consumers, businesses and lawmakers alike.

Recognizing Clients
To provide personalized services to consumers, businesses must be able to recognize clients when they request information from a site. As we've discussed, the request–response system on which the web operates is facilitated by HTTP. Unfortunately, HTTP is a *stateless*

protocol—it *does not* provide information that would enable web servers to maintain state information regarding particular clients. This means that web servers cannot determine whether a request comes from a particular client or whether the same or different clients generate a series of requests.

To circumvent this problem, sites can provide mechanisms by which they identify individual clients. A *session* represents a unique client on a website. If the client leaves a site and then returns later, the client will still be recognized as the same user. When the user closes the browser, the session typically ends. To help the server distinguish among clients, each client must identify itself to the server. Tracking individual clients is known as **session tracking**. One popular session-tracking technique uses cookies (discussed in Section 23.7.1); another uses ASP.NET's `HttpSessionState` object (used in Section 23.7.2). Additional session-tracking techniques are beyond this book's scope.

23.7.1 Cookies

Cookies provide you with a tool for personalizing web pages. A cookie is a piece of data stored by web browsers in a small text file on the user's computer. A cookie maintains information about the client during and between browser sessions. The first time a user visits the website, the user's computer might receive a cookie from the server; this cookie is then reactivated each time the user revisits that site. The collected information is intended to be an anonymous record containing data that's used to personalize the user's future visits to the site. For example, cookies in a shopping app might store unique identifiers for users. When a user adds items to an online shopping cart or performs another task resulting in a request to the web server, the server receives a cookie containing the user's unique identifier. The server then uses the unique identifier to locate the shopping cart and perform any necessary processing.

In addition to identifying users, cookies also can indicate users' shopping *preferences*. When a Web Form receives a request from a client, the Web Form can examine the cookie(s) it sent to the client during previous communications, identify the user's preferences and immediately display products of interest to the client.

Every HTTP-based interaction between a client and a server includes a header containing information either about the request (when the communication is from the client to the server) or about the response (when the communication is from the server to the client). When a Web Form receives a request, the header includes information such as the request type and any cookies that have been sent previously from the server to be stored on the client machine. When the server formulates its response, the header information contains any cookies the server wants to store on the client computer and other information, such as the type of data in the response.

The **expiration date** of a cookie determines how long the cookie remains on the client's computer. If you do not set an expiration date for a cookie, the web browser maintains the cookie for the duration of the browsing session. Otherwise, the web browser maintains the cookie until the expiration date occurs. Cookies are deleted when they **expire**.

Portability Tip 23.1

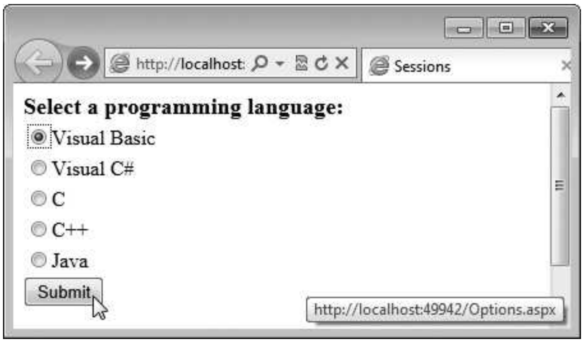
Users may disable cookies in their web browsers to help ensure their privacy. Such users will experience difficulty using web apps that depend on cookies to maintain state information.

23.7.2 Session Tracking with `HttpSessionState`

The next web app demonstrates session tracking using class **`HttpSessionState`**. When you execute this app, the `Options.aspx` page (Fig. 23.27(a)), which is the app's **Start Page**, allows the user to select a programming language from a group of radio buttons. [*Note:* You might need to right click `Options.aspx` in the **Solution Explorer** and select **Set As Start Page** before running this app.] When the user clicks **Submit**, the selection is sent to the web server for processing. The web server uses an `HttpSessionState` object to store the chosen language and the ISBN number for one of our books on that topic. Each user that visits the site has a unique `HttpSessionState` object, so the selections made by one user are maintained separately from all other users. After storing the selection, the server returns the page to the browser (Fig. 23.27(b)) and displays the user's selection and some information about the user's unique session (which we show just for demonstration purposes). The page also includes links that allow the user to choose between selecting another programming language or viewing the `Recommendations.aspx` page (Fig. 23.27(e)), which lists recommended books pertaining to the programming language(s) that the user selected previously. If the user clicks the link for book recommendations, the information stored in the user's unique `HttpSessionState` object is read and used to form the list of recommendations. To test this app:

1. Select **Open Web Site...** from the **FILE** menu.
2. In the **Open Web Site** dialog, ensure that **File System** is selected, then navigate to this chapter's examples, select the `Sessions` folder and click the **Open** `Button`.
3. Select `Options.aspx` in the **Solution Explorer**, then type *Ctrl* + *F5* to execute the web app in your default web browser.

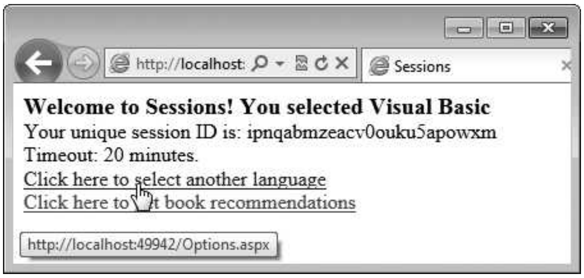
a) User selects a language from the `Options.aspx` page, then presses **Submit** to send the selection to the server



Select a programming language:

- [✓ Visual Basic]
- Visual C#
- C
- C++
- Java

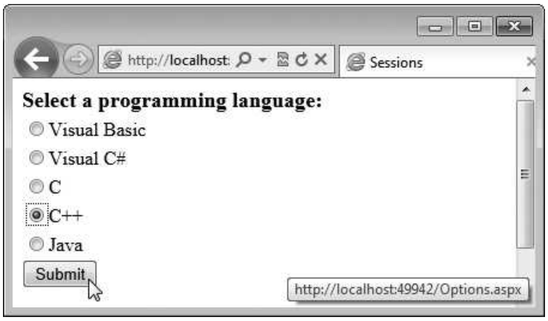
b) `Options.aspx` page is updated to hide the controls for selecting a language and to display the user's selection; the user clicks the hyperlink to return to the list of languages and make another selection



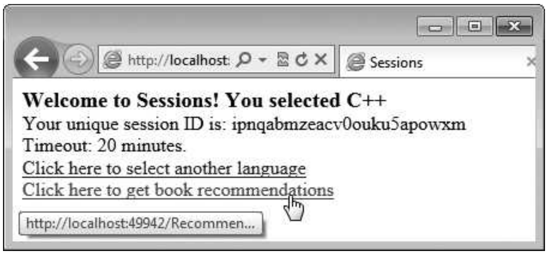
Welcome to Sessions! You selected Visual Basic
Your unique session ID is: ipnqabmzeacv0ouku5apowxm
Timeout: 20 minutes.
Click here to select another language
Click here to get book recommendations

Fig. 23.27 | ASPX file that presents a list of programming languages. (Part 1 of 2.)

c) User selects another language from the `Options.aspx` page, then presses **Submit** to send the selection to the server

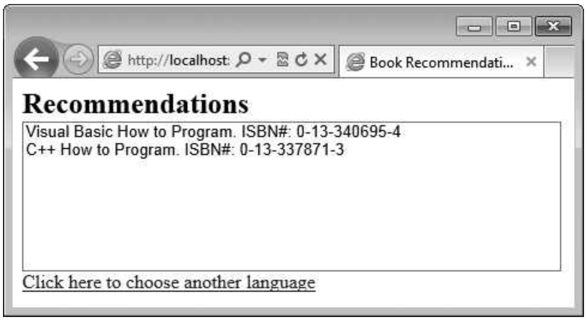

http://localhost:
Sessions
Select a programming language:
Visual Basic
Visual C#
C
C++
Java
Submit
http://localhost:49942/Options.aspx

d) `Options.aspx` page is updated to hide the controls for selecting a language and to display the user's selection; the user clicks the hyperlink to get a list of book recommendations



Welcome to Sessions! You selected C++
Your unique session ID is: ipnqabmzeacv0ouku5apowxm
Timeout: 20 minutes.
Click here to select another language
Click here to get book recommendations

e) `Recommendations.aspx` displays the list of recommended books based on the user's selections



Recommendations
Visual Basic How to Program. ISBN#: 0-13-340695-4
C++ How to Program. ISBN#: 0-13-337871-3
Click here to choose another language

Fig. 23.27 | ASPX file that presents a list of programming languages. (Part 2 of 2.)

Creating the Web Site

To begin, follow the steps in Section 23.4.1 to create an **Empty Web Site** named `Sessions`, then add two Web Forms named `Options.aspx` and `Recommendations.aspx` to the project. Set the `Options.aspx` document's `Title` property to `"Sessions"` and the `Recommendations.aspx` document's `Title` property to `"Book Recommendations"`. To ensure that `Options.aspx` is the first page to load for this app, right click it in the **Solution Explorer** and select **Set As Start Page**.

23.7.3 `Options.aspx`: Selecting a Programming Language

The `Options.aspx` page Fig. 23.27(a) contains the following controls arranged *vertically*:

1. A `Label` with its `(ID)` property set to `promptLabel` and its `Text` property set to `"Select a programming language:"`. We used the techniques shown in *Step 5* of Section 23.4.1 to create a *CSS style* for this label named `.labelStyle`, and set the style's `font-size` attribute to `large` and the `font-weight` attribute to `bold`.

2. The user selects a programming language by clicking one of the radio buttons in a `RadioButtonList`. Each radio button has a `Text` property and a `Value` property. The `Text` property is displayed next to the radio button and the `Value` property represents a value that's sent to the server when the user selects that radio button and submits the form. In this example, we'll use the `Value` property to represent the ISBN for the recommended book. Create a `RadioButtonList` with its `(ID)` property set to `languageList`. Use the **ListItem Collection Editor** to add five radio buttons with their `Text` properties set to `Visual Basic`, `Visual C#`, `C`, `C++` and `Java`, and their `Value` properties set to `0-13-340695-4`, `0-13-337933-7`, `0-13-299044-X`, `0-13-337871-3` and `0-13-294094-9`, respectively.
3. A `Button` with its `(ID)` property set to `submitButton` and its `Text` property set to `Submit`. In this example, we'll handle this `Button`'s `Click` event. You can create its event handler by double clicking the `Button` in **Design** view.
4. A `Label` with its `(ID)` property set to `responseLabel` and its `Text` property set to `"Welcome to Sessions!"`. This `Label` should be placed immediately to the right of the `Button` so that the `Label` appears at the top of the page when we hide the preceding controls on the page. Reuse the CSS style you created in *Step 1* by setting this `Label`'s `CssClass` property to `labelStyle`.
5. Two more `Labels` with their `(ID)` properties set to `idLabel` and `timeoutLabel`, respectively. Clear the text in each `Label`'s `Text` property—you'll set these *programmatically* with information about the current user's session.
6. A `HyperLink` with its `(ID)` property set to `languageLink` and its `Text` property set to `"Click here to choose another language"`. Set its `NavigateUrl` property by clicking the ellipsis next to the property in the **Properties** window and selecting `Options.aspx` from the **Select URL** dialog.
7. A `HyperLink` with its `(ID)` property set to `recommendationsLink` and its `Text` property set to `"Click here to get book recommendations"`. Set its `NavigateUrl` property by clicking the ellipsis next to the property in the **Properties** window and selecting `Recommendations.aspx` from the **Select URL** dialog.
8. Initially, the controls in *Steps 4–7* will not be displayed, so set each control's `Visible` property to `false`.

`Session` Property of a `Page`

Every ASP.NET Web app includes a user-specific `HttpSessionState` object, which is accessible through property **`Session`** of class `Page`. Throughout this section, we use this property to manipulate the current user's `HttpSessionState` object. When the user first requests a page in a Web app, a unique `HttpSessionState` object is created by ASP.NET and assigned to the `Page`'s `Session` property. The same object is also available to the app's other pages.

Code-Behind File for `Options.aspx`

Fig. 23.28 presents the code-behind file for the `Options.aspx` page. When this page is requested, the `Page_Load` event handler (lines 10–40) executes before the response is sent to the client. Since the first request to a page is *not* a *postback*, the code in lines 16–38 *does not* execute the first time the page loads.

```
// Fig. 23.28: Options.aspx.cs
// Processes user's selection of a programming language by displaying
// links and writing information in an HttpSessionState object.
using System;

public partial class Options : System.Web.UI.Page
{
   // if postback, hide form and display links to make additional
   // selections or view recommendations
   protected void Page_Load( object sender, EventArgs e )
   {
      if ( IsPostBack )
      {
         // user has submitted information, so display message
         // and appropriate hyperlinks
         responseLabel.Visible = true;
         idLabel.Visible = true;
         timeoutLabel.Visible = true;
         languageLink.Visible = true;
         recommendationsLink.Visible = true;

         // hide other controls used to make language selection
         promptLabel.Visible = false;
         languageList.Visible = false;
         submitButton.Visible = false;

         // if the user made a selection, display it in responseLabel
         if ( languageList.SelectedItem != null )
            responseLabel.Text += " You selected " +
               languageList.SelectedItem.Text;
         else
            responseLabel.Text += " You did not select a language.";

         // display session ID
         idLabel.Text = "Your unique session ID is: " + Session.SessionID;

         // display the timeout
         timeoutLabel.Text = "Timeout: " + Session.Timeout + " minutes.";
      } // end if
   } // end method Page_Load

   // record the user's selection in the Session
   protected void submitButton_Click( object sender, EventArgs e )
   {
      // if the user made a selection
      if ( languageList.SelectedItem != null )
         // add name-value pair to Session
         Session.Add( languageList.SelectedItem.Text,
            languageList.SelectedItem.Value );
   } // end method submitButton_Click
} // end class Options
```

Fig. 23.28 | Process user's selection of a programming language by displaying links and writing information in an `HttpSessionState` object.

Postback Processing

When the user presses **Submit**, a postback occurs. The form is submitted to the server and `Page_Load` executes. Lines 16–20 display the controls shown in Fig. 23.27(b) and lines 23–25 hide the controls shown in Fig. 23.27(a). Next, lines 28–32 ensure that the user selected a language and, if so, display a message in the `responseLabel` indicating the selection. Otherwise, the message `"You did not select a language"` is displayed.

The ASP.NET app contains information about the `HttpSessionState` object (property `Session` of the `Page` object) for the current client. The object's **`SessionID`** property (displayed in line 35) contains the **unique session ID**—a sequence of random letters and numbers. The *first* time a client connects to the web server, a unique session ID is created for that client and a temporary cookie is written to the client so the server can identify the client on subsequent requests. When the client makes additional requests, the client's session ID from that temporary cookie is compared with the session IDs stored in the web server's memory to retrieve the client's `HttpSessionState` object. (Since users can disable cookies in their browsers, you can also use cookieless sessions. For more information, see `msdn.microsoft.com/en-us/library/aa479314.aspx`.) `HttpSessionState` property **`Timeout`** (displayed in line 38) specifies the maximum amount of time that an `HttpSessionState` object can be inactive before it's discarded. By default, if the user does not interact with this web app for 20 minutes, the `HttpSessionState` object is discarded by the server and a new one will be created if the user interacts with the app again. Figure 23.29 lists some common `HttpSessionState` properties.

Properties	Description
`Count`	Specifies the number of key–value pairs in the `Session` object.
`IsNewSession`	Indicates whether this is a new session (that is, whether the session was created during loading of this page).
`Keys`	Returns a collection containing the `Session` object's keys.
`SessionID`	Returns the session's unique ID.
`Timeout`	Specifies the maximum number of minutes during which a session can be inactive (that is, no requests are made) before the session expires. By default, this property is set to 20 minutes.

Fig. 23.29 | `HttpSessionState` properties.

Method **`submitButton_Click`**

We store the user's selection in an `HttpSessionState` object when the user clicks the **Submit** `Button`. The `submitButton_Click` event handler (lines 43–50) adds a key–value pair to the `HttpSessionState` object for the current user, specifying the language chosen and the ISBN number for a book on that language. The `HttpSessionState` object is a dictionary—a data structure that stores **key–value pairs**. A program uses the key to store and retrieve the associated value in the dictionary. We covered dictionaries in Chapter 21.

The key–value pairs in an `HttpSessionState` object are often referred to as **session items**. They're placed in an `HttpSessionState` object by calling its **`Add`** method. If the user made a selection (line 46), lines 48–49 get the selection and its corresponding value from the `languageList` by accessing its `SelectedItem`'s `Text` and `Value` properties, respectively, then call `HttpSessionState` method `Add` to add this name–value pair as a session item in the `HttpSessionState` object (`Session`).

If the app adds a session item that has the same name as an item previously stored in the `HttpSessionState` object, the session item is replaced—session item names *must* be unique. Another common syntax for placing a session item in the `HttpSessionState` object is `Session[`*Name*`] = `*Value*. For example, we could have replaced lines 48–49 with

```
Session[ languageList.SelectedItem.Text ] =
   languageList.SelectedItem.Value
```

Software Engineering Observation 23.1

A Web Form should not use instance variables to maintain client state information, because each new request or postback is handled by a new instance of the page. Instead, maintain client state information in `HttpSessionState` *objects, because such objects are specific to each client.*

Software Engineering Observation 23.2

A benefit of using `HttpSessionState` *objects (rather than cookies) is that they can store any type of object (not just* `Strings`*) as attribute values. This provides you with increased flexibility in determining the type of state information to maintain for clients.*

23.7.4 `Recommendations.aspx`: Displaying Recommendations Based on Session Values

After the postback of `Options.aspx`, the user may request book recommendations. The book-recommendations hyperlink forwards the user to the page `Recommendations.aspx` (Fig. 23.27(e)) to display the recommendations based on the user's language selections. The page contains the following controls arranged *vertically*:

1. A `Label` with its `(ID)` property set to `recommendationsLabel` and its `Text` property set to `"Recommendations"`. We created a CSS style for this label named `.labelStyle`, and set the `font-size` attribute to `x-large` and the `font-weight` attribute to `bold`. (See *Step 5* in Section 23.4.1 for information on creating a CSS style.)
2. A `ListBox` with its `(ID)` property set to `booksListBox`. We created a CSS style for this `ListBox` named `.listBoxStyle`. In the **Position** category, we set the `width` attribute to `450px` and the `height` attribute to `125px`. The `px` indicates that the measurement is in pixels.
3. A `HyperLink` with its `(ID)` property set to `languageLink` and its `Text` property set to `"Click here to choose another language"`. Set its `NavigateUrl` property by clicking the ellipsis next to the property in the **Properties** window and selecting `Options.aspx` from the **Select URL** dialog. When the user clicks this link, the `Options.aspx` page will be reloaded. Requesting the page in this manner *is not* considered a postback, so the original form in Fig. 23.27(a) will be displayed.

Code-Behind File for `Recommendations.aspx`

Figure 23.30 presents the *code-behind file* for `Recommendations.aspx`. Event handler `Page_Init` (lines 8–28) retrieves the session information. If a user has not selected a language in the `Options.aspx` page, the `HttpSessionState` object's **`Count`** property will be `0` (line 11). This property provides the number of session items contained in a `HttpSessionState` object. If the `Count` is `0`, then we display the text **No Recommendations** (line 22), hide the `ListBox` (line 23) and update the `Text` of the `HyperLink` back to `Options.aspx` (line 26).

```
 1  // Fig. 23.30: Recommendations.aspx.cs
 2  // Creates book recommendations based on a Session object.
 3  using System;
 4
 5  public partial class Recommendations : System.Web.UI.Page
 6  {
 7     // read Session items and populate ListBox with recommendations
 8     protected void Page_Init( object sender, EventArgs e )
 9     {
10        // determine whether Session contains any information
11        if ( Session.Count != 0 )
12        {
13           // display Session's name-value pairs
14           foreach ( string keyName in Session.Keys )
15              booksListBox.Items.Add( keyName +
16                 " How to Program. ISBN#: " + Session[ keyName ] );
17        } // end if
18        else
19        {
20           // if there are no session items, no language was chosen, so
21           // display appropriate message and clear and hide booksListBox
22           recommendationsLabel.Text = "No Recommendations";
23           booksListBox.Visible = false;
24
25           // modify languageLink because no language was selected
26           languageLink.Text = "Click here to choose a language";
27        } // end else
28     } // end method Page_Init
29  } // end class Recommendations
```

Fig. 23.30 | Session data used to provide book recommendations to the user.

If the user chose at least one language, the loop in lines 14–16 iterates through the `HttpSessionState` object's keys (line 14) by accessing the `HttpSessionState`'s **`Keys`** property, which returns a collection containing all the keys in the session. Lines 15–16 concatenate the `keyName`, the `String " How to Program. ISBN#: "` and the key's corresponding value, which is returned by `Session[keyName]`. This `String` is the recommendation that's added to the `ListBox`.

23.8 Case Study: Database-Driven ASP.NET Guestbook

Many websites allow users to provide feedback about the website in a guestbook. Typically, users click a link on the website's home page to request the guestbook page. This page usually consists of a form that contains fields for the user's name, e-mail address, message/

feedback and so on. Data submitted on the guestbook form is then stored in a database located on the server.

In this section, we create a guestbook Web Form app. The GUI (Fig. 23.31) contains a **GridView** data control, which displays all the entries in the guestbook in tabular format. This control is located in the **Toolbox**'s **Data** section. We explain how to create and configure this data control shortly. The GridView displays **abc** in **Design** mode to indicate data that will be retrieved from a data source at runtime. You'll learn how to create and configure the GridView shortly.

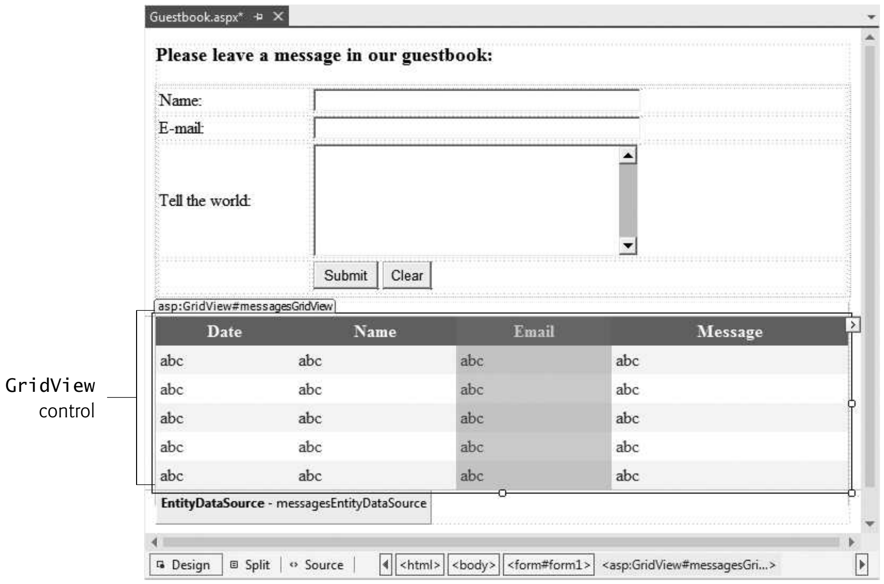

Guestbook.aspx*
Please leave a message in our guestbook:
Name:
E-mail:
Tell the world:
Submit
Clear
asp:GridView#messagesGridView
Date
Name
Email
Message
abc
GridView control
EntityDataSource - messagesEntityDataSource
Design
Split
Source
<html>
<body>
<form#form1>
<asp:GridView#messagesGri...>

Fig. 23.31 | Guestbook app GUI in **Design** mode.

The Guestbook Database

The app stores the guestbook information in a SQL Server database called Guestbook.mdf located on the web server. (We provide this database in the databases folder with this chapter's examples.) The database contains a single table named Messages.

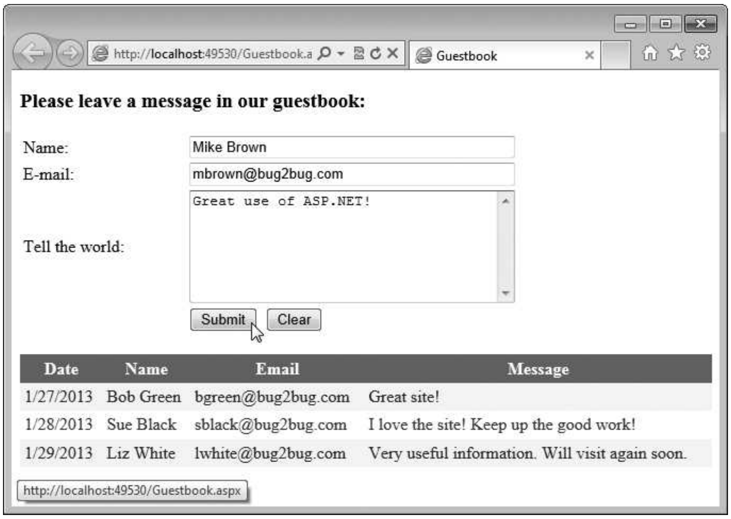
Testing the App

To test this app:

1. Select **Open Web Site...** from the **FILE** menu.
2. In the **Open Web Site** dialog, ensure that **File System** is selected, then navigate to this chapter's examples, select the Guestbook folder and click the **Open** Button.
3. Select Guestbook.aspx in the **Solution Explorer**, then type *Ctrl* + *F5* to execute the web app in your default web browser.

Figure 23.32(a) shows the user submitting a new entry. Figure 23.32(b) shows the new entry as the last row in the GridView.

a) User enters data for the name, e-mail and message, then presses **Submit** to send the data to the server


http://localhost:49530/Guestbook.a
Guestbook

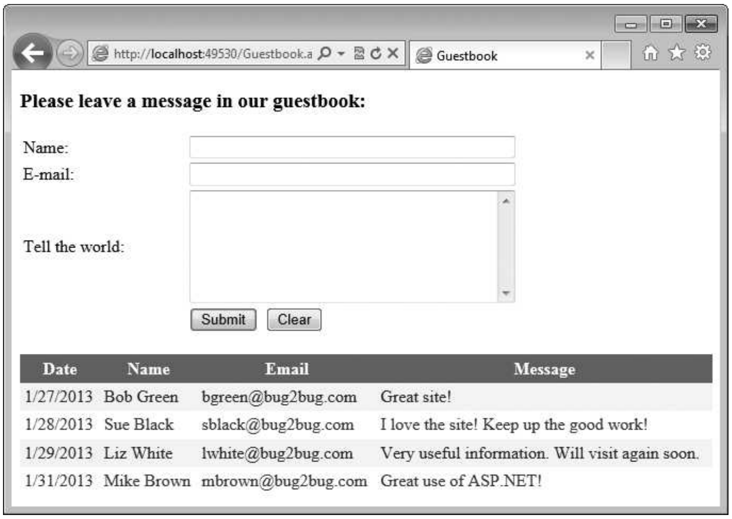
Please leave a message in our guestbook:

Name:	Mike Brown
E-mail:	mbrown@bug2bug.com
Tell the world:	Great use of ASP.NET!

Date	Name	Email	Message
1/27/2013	Bob Green	bgreen@bug2bug.com	Great site!
1/28/2013	Sue Black	sblack@bug2bug.com	I love the site! Keep up the good work!
1/29/2013	Liz White	lwhite@bug2bug.com	Very useful information. Will visit again soon.

http://localhost:49530/Guestbook.aspx

b) Server stores the data in the database, then refreshes the `GridView` with the updated data



Please leave a message in our guestbook:

Name:	
E-mail:	
Tell the world:	

Date	Name	Email	Message
1/27/2013	Bob Green	bgreen@bug2bug.com	Great site!
1/28/2013	Sue Black	sblack@bug2bug.com	I love the site! Keep up the good work!
1/29/2013	Liz White	lwhite@bug2bug.com	Very useful information. Will visit again soon.
1/31/2013	Mike Brown	mbrown@bug2bug.com	Great use of ASP.NET!

Fig. 23.32 | Sample execution of the **Guestbook** app.

23.8.1 Building a Web Form that Displays Data from a Database

You'll now build this GUI and set up the data binding between the `GridView` control and the database. Many of these steps are similar to those performed in Chapter 22 to access and interact with a database in a Windows app. We discuss the *code-behind file* in Section 23.8.2.

Step 1: Creating the Web Site

To begin, follow the steps in Section 23.4.1 to create an **Empty Web Site** named `Guestbook` then add a Web Form named `Guestbook.aspx` to the project. Set the document's `Title` property to `"Guestbook"`. To ensure that `Guestbook.aspx` loads when you execute this app, right click it in the **Solution Explorer** and select **Set As Start Page**.

Step 2: Creating the Form for User Input

In **Design** mode, add the text `Please leave a message in our guestbook:`, then use the **Block Format** `ComboBox` in the IDE's toolbar to change the text to **Heading 3** format. Insert a table with four rows and two columns. Place the appropriate text (see Fig. 23.31) in the top three cells in the table's left column. Then place `TextBoxes` named `nameTextBox`, `emailTextBox` and `messageTextBox` in the top three table cells in the right column. Configure the `TextBoxes` as follows:

- Select **FORMAT > New Style...** to display the **New Style** dialog. In the **Selector** field, specify `.textBoxWidth` as the new style's name. Select the **Category** named **Position**, then set the **width:** to `300px` and click **OK** to create the style and dismiss the dialog. Next set the `CssClass` property for both the `nameTextBox` and `emailTextBox` to `textBoxWidth`. This uses the style to set both `TextBoxes` to `300` pixels wide.
- Select **FORMAT > New Style...** to display the **New Style** dialog. In the **Selector** field, specify `.textBoxHeight` as the new style's name. Select the **Category** named **Position**, then set the **height:** to `100px` and click **OK** to create the style and dismiss the dialog. Next set `messageTextBox`'s `CssClass` property to

```
    textBoxWidth textBoxHeight
```

  This uses both the `.textBoxWidth` and `.textBoxHeight` styles to set `messageTextBox`'s width to `300` pixels and height to `100` pixels. Also set `messageTextBox`'s `TextMode` property to `MultiLine` so the user can type a message containing multiple lines of text.

Finally, add `Buttons` named `submitButton` and `clearButton` to the bottom-right table cell. Set the buttons' `Text` properties to `Submit` and `Clear`, respectively. We discuss the buttons' event handlers when we present the code-behind file. You can create these event handlers now by double clicking each `Button` in **Design** view.

Step 3: Adding a `GridView` Control to the Web Form

Add a `GridView` named `messagesGridView` that will display the guestbook entries. This control is located in the **Toolbox**'s **Data** section. The colors for the `GridView` are specified through the **Auto Format...** link in the **GridView Tasks** *smart-tag menu* that opens when you place the `GridView` on the page. Clicking this link displays an **AutoFormat** dialog with several choices. In this example, we chose **Professional**. We show how to set the `GridView`'s data source (that is, where it gets the data to display in its rows and columns) shortly.

Step 4: Creating the Entity Data Model

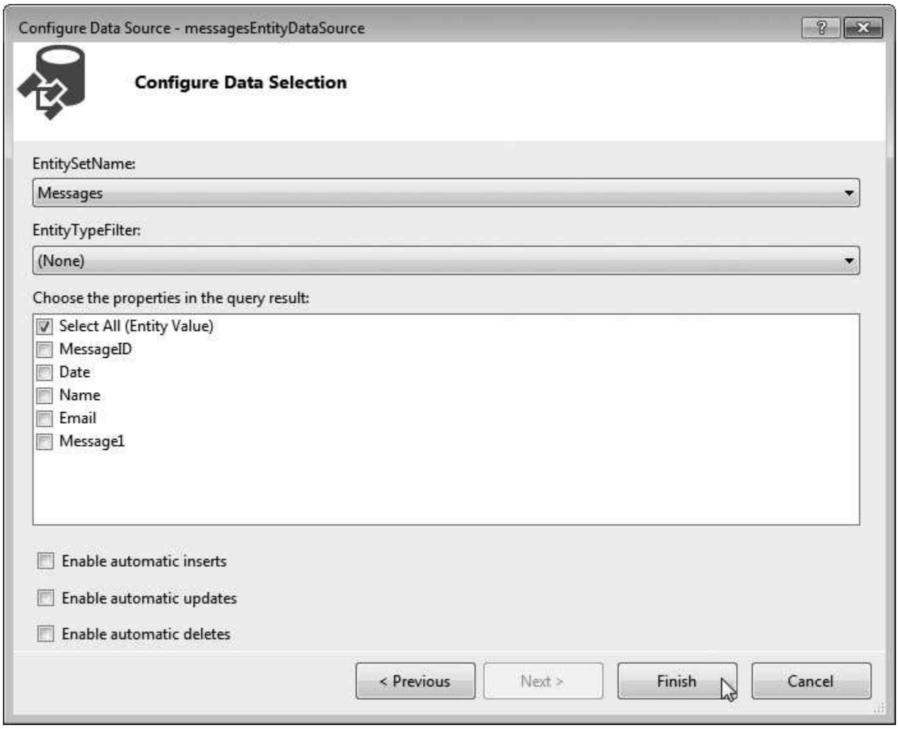
Next, you'll add an entity data model to the project. Perform the following steps:

1. Right click the project name in the Solution Explorer and select **Add > Add New Item...** to display the **Add New Item** dialog.
2. Select **ADO.NET Entity Data Model**, change the **Name** to `GuestbookModel.edmx` and click **Add**. A dialog appears asking if you would like to put your new entity data model classes in the `App_Code` folder; click **Yes**. The IDE will create an `App_Code` folder and place the entity data model classes information in that folder. For security reasons, this folder can be accessed only by the web app on the server—clients cannot access this folder over a network.

3. Next, the **Entity Data Model Wizard** dialog appears. Ensure that **Generate from database** is selected so that you can generate the model from the `Guestbook.mdf` database, then click **Next >**.
4. In the **Entity Data Model Wizard** dialog's **Choose Your Data Connection** step, click **New Connection...** then use the **Connection Properties** dialog to locate the `Guestbook.mdf` database file (included in the `databases` folder with this chapter's examples). Click **OK** to create the connection, then click **Next >** to complete the **Choose Your Data Connection** step.
5. A dialog appears asking if you would like to copy the database file into your project. Click **Yes**. The IDE will create an `App_Data` folder and place the `Guestbook.mdf` file in that folder. Like the `App_Code` folder, this folder can be accessed only by the web app on the server.
6. In the **Entity Data Model Wizard** dialog's **Choose Your Database Objects and Settings** step, select the `Messages` table from the database. By default, the IDE names the model `GuestbookModel`. Ensure that **Pluralize or singularize generated object names** is checked, keep the other default settings and click **Finish**. The IDE displays the `GuestbookModel` in the editor, where you can see that a `Message` has a `MessageID`, `Date`, `Name`, `Email` and `Message1` property. `Message1` was renamed from `Message` by the IDE so that it does not conflict with the entity data model's `Message` class.
7. Select **BUILD > Build Solution** to ensure that the new entity data model classes are compiled.

*Step 5: Binding the **`GridView`** to the **`Messages`** Table of the **`Guestbook`** Database*
You can now configure the `GridView` to display the database's data.

1. In the **GridView Tasks** smart-tag menu, select **<New data source...>** from the **Choose Data Source** `ComboBox` to display the **Data Source Configuration Wizard** dialog.
2. In this example, we use a **`EntityDataSource`** control that allows the app to interact with the `Guestbook.mdf` database. Select **Entity**, then in the **Specify an ID for the data source** field enter `messagesEntityDataSource` and click **OK** to begin the **Configure Data Source** wizard.
3. In the **Configure ObjectContext** step, select `GuestbookEntities` in the **Named Connection** `ComboBox`, then click **Next >**.
4. The **Configure Data Selection** screen (Fig. 23.33) allows you to specify which data the `EntityDataSource` should retrieve from the data context. The **EntitySetName** drop-down list contains `DbContext` properties that represent database's tables. For the `Guestbook` database, select `Messages` from the drop-down list. In the **Choose the properties in the query result:** pane, ensure that the **Select All** checkbox is selected to indicate that you want to retrieve all the columns in the `Messages` table.
5. Click **Finish** to complete the wizard. A control named `messagesEntityDataSource` now appears on the Web Form directly below the `GridView`. It's represented in **Design** mode as a gray box containing its type and name. It will *not* appear on the web page—the gray box simply provides a way to manipulate the



EntitySetName: Messages

EntityTypeFilter: (None)

Choose the properties in the query result:
- [✓] Select All (Entity Value)
- [ ] MessageID
- [ ] Date
- [ ] Name
- [ ] Email
- [ ] Message1

- [ ] Enable automatic inserts
- [ ] Enable automatic updates
- [ ] Enable automatic deletes

Fig. 23.33 | Configuring the `EntityDataSource`.

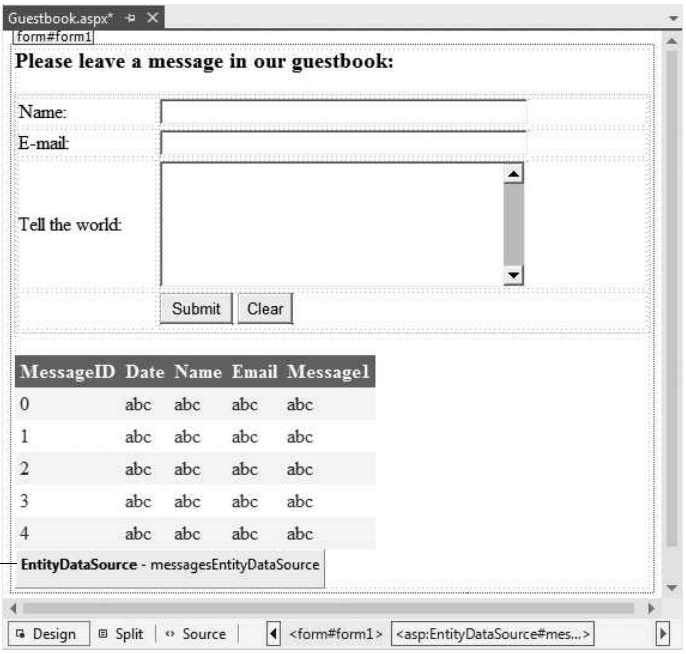
control visually through **Design** mode—similar to how the objects in the component tray are used in **Design** mode for a Windows `Forms` app.

6. Click the `messagesEntityDataSource`, then select **Refresh Schema** from its smart-tag menu. The `GridView` updates to display column headers that correspond to the columns in the `Messages` table (Fig. 23.34). The rows each contain either a number (which signifies an *autoincremented column*) or **abc** (which indicates string data). The actual data from the `Guestbook.mdf` database file will appear in these rows when you view the ASPX file in a web browser.

Step 6: Customizing the Columns of the Data Source Displayed in the `GridView`

It's not necessary for site visitors to see the `MessageID` column when viewing past guestbook entries—this column is merely a unique primary key required by the `Messages` table within the database. So, let's modify the `GridView` to prevent this column from displaying on the Web Form. We'll also modify the column **Message1** to read **Message**.

1. In the **GridView Tasks** *smart tag menu*, click **Edit Columns** to display the **Fields** dialog (Fig. 23.35).
2. Select **MessageID** in the **Selected fields** pane, then click the ✕ `Button`. This removes the `MessageID` column from the `GridView`.



Please leave a message in our guestbook:

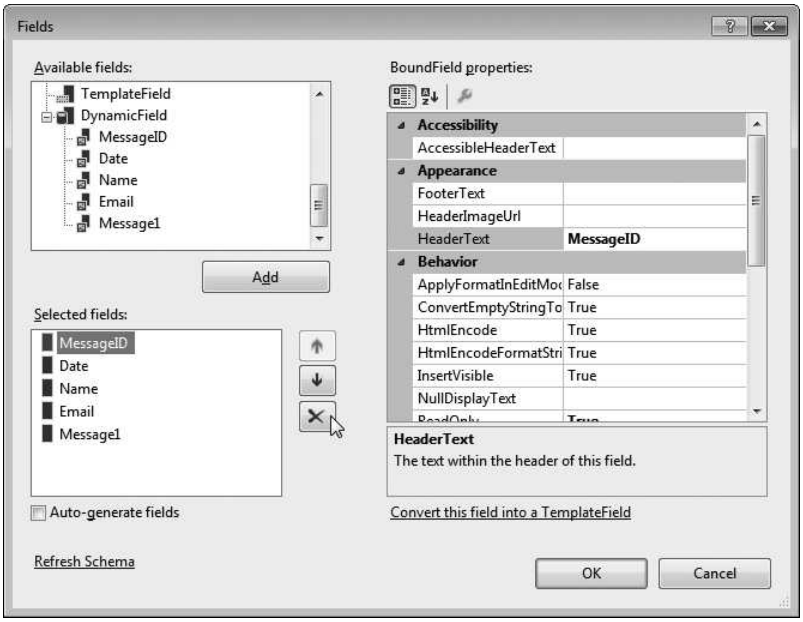
Name:
E-mail:
Tell the world:

MessageID	Date	Name	Email	Message1
0	abc	abc	abc	abc
1	abc	abc	abc	abc
2	abc	abc	abc	abc
3	abc	abc	abc	abc
4	abc	abc	abc	abc

EntityDataSource - messagesEntityDataSource

`EntityDataSource` control

Fig. 23.34 | **Design** mode displaying `EntityDataSource` control for a `GridView`.



Accessibility	
AccessibleHeaderText	
Appearance	
FooterText	
HeaderImageUrl	
HeaderText	**MessageID**
Behavior	
ApplyFormatInEditMo	False
ConvertEmptyStringTo	True
HtmlEncode	True
HtmlEncodeFormatStri	True
InsertVisible	True
NullDisplayText	

HeaderText
The text within the header of this field.

Fig. 23.35 | Removing the `MessageID` column from the `GridView`.

3. Select **Message1** in the **Selected fields** pane and change its `HeaderText` property to `Message`. The IDE renamed this field to prevent a naming conflict in the entity data model classes. Click **OK** to return to the main IDE window.
4. Next create a style to specify the width of the `GridView`. Select **FORMAT > New Style...** to display the **New Style** dialog. In the **Selector** field, specify `.gridViewWidth` as the new style's name. Select the **Category** named **Position**, then set the **width:** to `650px` and click **OK** to create the style and dismiss the dialog. Next set the `CssClass` property for the `messagesGridView` to `gridViewWidth`.

The `GridView` should now appear as shown in Fig. 23.31.

23.8.2 Modifying the Code-Behind File for the Guestbook App

After building the Web Form and configuring the data controls used in this example, double click the **Submit** and **Clear** buttons in **Design** view to create their corresponding `Click` event handlers in the code-behind file (Fig. 23.36). The IDE generates empty event handlers, so we must add the appropriate code to make these buttons work properly. The event handler for `clearButton` (lines 39–44) clears each `TextBox` by setting its `Text` property to an empty string. This resets the form for a new guestbook submission.

```
 1  // Fig. 23.36: Guestbook.aspx.cs
 2  // Code-behind file that defines event handlers for the guestbook.
 3  using System;
 4
 5  public partial class Guestbook : System.Web.UI.Page
 6  {
 7     // Submit Button adds a new guestbook entry to the database,
 8     // clears the form and displays the updated list of guestbook entries
 9     protected void submitButton_Click( object sender, EventArgs e )
10     {
11        // use GuestbookEntities DbContext to add a new message
12        using ( GuestbookEntities dbcontext = new GuestbookEntities() )
13        {
14           // create a new Message to add to the database; Message is
15           // the entity data model class representing a table row
16           Message message = new Message();
17
18           // set new Message's properties
19           message.Date = DateTime.Now.ToShortDateString();
20           message.Name = nameTextBox.Text;
21           message.Email = emailTextBox.Text;
22           message.Message1 = messageTextBox.Text;
23
24           // add new Message to GuestbookEntities DbContext
25           dbcontext.Messages.Add( message );
26           dbcontext.SaveChanges(); // save changes to the database
27        } // end using statement
28
```

Fig. 23.36 | Code-behind file for the guestbook app. (Part 1 of 2.)

```
29         // clear the TextBoxes
30         nameTextBox.Text = String.Empty;
31         emailTextBox.Text = String.Empty;
32         messageTextBox.Text = String.Empty;
33
34         // update the GridView with the new database table contents
35         messagesGridView.DataBind();
36      } // submitButton_Click
37
38      // Clear Button clears the Web Form's TextBoxes
39      protected void clearButton_Click( object sender, EventArgs e )
40      {
41         nameTextBox.Text = String.Empty;
42         emailTextBox.Text = String.Empty;
43         messageTextBox.Text = String.Empty;
44      } // clearButton_Click
45   } // end class Guestbook
```

Fig. 23.36 | Code-behind file for the guestbook app. (Part 2 of 2.)

Lines 9–36 contain `submitButton`'s event-handling code, which adds the user's information to the `Guestbook` database's `Messages` table. The `using` statement in lines 12–27 begins by creating a `GuestbookEntities` object to interact with the database. Recall that the `using` statement will call `Dispose` on this `GuestbookEntities` object when the `using` statement terminates. This is a good practice for an ASP.NET web page request, so that we don't maintain a connection to the database beyond the request.

Line 16 creates an object of the entity data model's `Message` class, which represents a row in the database's `Messages` table. Lines 19–22 set the new `Message` object's properties to the values that should be stored in the database. Line 25 calls the `Add` method of the `GuestbookEntities` object's `Messages` property, which represents the `Messages` table in the database. This adds a new record to the entity data model's representation of the table. Line 26 then saves the changes into the database.

After the data is inserted into the database, lines 30–32 clear the `TextBoxes`, and line 35 invokes `messagesGridView`'s **`DataBind` method** to refresh the data that the `GridView` displays. This causes `messagesEntityDataSource` (the `GridView`'s data source) to obtain the `Messages` table's updated data from the database.

23.9 Online Case Study: ASP.NET AJAX

In the online Chapter 29, Web App Development with ASP.NET: A Deeper Look, you learn the difference between a traditional web app and an **AJAX (Asynchronous JavaScript and XML) web app**. You also learn how to use **ASP.NET AJAX** to quickly and easily improve the user experience for your web apps, giving them responsiveness comparable to that of desktop apps. To demonstrate ASP.NET AJAX capabilities, you enhance the validation example by displaying the submitted form information without reloading the entire page. The only modifications to this web app appear in `Validation.aspx` file. You use AJAX-enabled controls to add this feature.

23.10 Online Case Study: Password-Protected Books Database App

In the online Chapter 29, Web App Development with ASP.NET: A Deeper Look, we include a web app case study in which a user logs into a password-protected website to view a list of publications by a selected author. The app consists of several pages and provides website registration and login capabilities. You'll learn about ASP.NET master pages, which allow you to specify a common look-and-feel for all the pages in your app. We also introduce the **Web Site Administration Tool** and use it to configure the portions of the app that can be accessed only by users who are logged into the website.

23.11 Wrap-Up

In this chapter, we introduced web-app development using ASP.NET and Visual Studio Express 2012 for Web. We began by discussing the simple HTTP transactions that take place when you request and receive a web page through a web browser. You then learned about the three tiers (that is, the client or top tier, the business logic or middle tier and the information or bottom tier) that comprise most web apps.

Next, we explained the role of ASPX files (that is, Web Form files) and code-behind files, and the relationship between them. We discussed how ASP.NET compiles and executes web apps so that they can be displayed in a web browser. You also learned how to build an ASP.NET web app using Visual Studio Express For Web.

The chapter demonstrated several common ASP.NET web controls used for displaying text and images on a Web Form. We also discussed validation controls, which allow you to ensure that user input on a web page satisfies certain requirements.

We discussed the benefits of maintaining a user's state information across multiple pages of a website. We then demonstrated how you can include such functionality in a web app by using session tracking with `HttpSessionState` objects.

Finally, we built a guestbook app that allows users to submit comments about a website. You learned how to save the user input in a database and how to display past submissions on the web page.

Summary

Section 23.1 Introduction

- ASP.NET technology is Microsoft's technology for web-app development.
- Web Form files have the file-name extension `.aspx` and contain the web page's GUI. A Web Form file represents the web page that's sent to the client browser.
- The file that contains the programming logic of a Web Form is called the code-behind file.

Section 23.2 Web Basics

- URIs (Uniform Resource Identifiers) identify resources on the Internet. URIs that start with `http://` are called URLs (Uniform Resource Locators).
- A URL contains information that directs a browser to the resource that the user wishes to access. Computers that run web server software make such resources available.

- In a URL, the hostname is the name of the server on which the resource resides. This computer usually is referred to as the host, because it houses and maintains resources.
- A hostname is translated into a unique IP address that identifies the server. This translation is performed by a domain-name system (DNS) server.
- The remainder of a URL specifies the location and name of a requested resource. For security reasons, the location is normally a virtual directory. The server translates the virtual directory into a real location on the server.
- When given a URL, a web browser uses HTTP to retrieve the web page found at that address.

Section 23.3 Multitier App Architecture

- Multitier apps divide functionality into separate tiers—logical groupings of functionality—that commonly reside on separate computers for security and scalability.
- The information tier (also called the bottom tier) maintains data pertaining to the app. This tier typically stores data in a relational database management system.
- The middle tier implements business logic, controller logic and presentation logic to control interactions between the app's clients and the app's data. The middle tier acts as an intermediary between data in the information tier and the app's clients.
- Business logic in the middle tier enforces business rules and ensures that data is reliable before the server app updates the database or presents the data to users.
- The client tier, or top tier, is the app's user interface, which gathers input and displays output. Users interact directly with the app through the user interface (typically viewed in a web browser), keyboard and mouse. In response to user actions, the client tier interacts with the middle tier to make requests and to retrieve data from the information tier. The client tier then displays to the user the data retrieved from the middle tier.

Section 23.4.1 Building the `WebTime` App

- **File System** websites are created and tested on your local computer. Such websites execute in IIS Express and can be accessed only by web browsers running on the same computer. You can later "publish" your website to a production web server for access via a local network or the Internet.
- **HTTP** websites are created and tested on an IIS web server and use HTTP to allow you to put your website's files on the server. If you own a website and have your own web server computer, you might use this to build a new website directly on that server computer.
- **FTP** websites use File Transfer Protocol (FTP) to allow you to put your website's files on the server. The server administrator must first create the website on the server for you. FTP is commonly used by so called "hosting providers" to allow website owners to share a server computer that runs many websites.
- A Web Form represents one page in a web app and contains a web app's GUI.
- You can view the Web Form's properties by selecting `DOCUMENT` in the **Properties** window. The `Title` property specifies the title that will be displayed in the web browser's title bar when the page is loaded.
- Controls and other elements are placed sequentially on a Web Form one after another in the order in which you drag-and-drop them onto the Web Form. The cursor indicates the insertion point in the page. This type of layout is known as relative positioning. You can also use absolute positioning in which controls are located exactly where you drop them on the Web Form.
- When a `Label` does not contain text, its name is displayed in square brackets in **Design** view as a placeholder for design and layout purposes. This text is not displayed at execution time.
- Formatting in a web page is performed with Cascading Style Sheets (CSS).

- A Web Form's `Init` event occurs when the page is requested by a web browser. The event handler for this event—named `Page_Init`—initialize the page.

Section 23.4.2 Examining **`WebTime.aspx`***'s Code-Behind File*

- A class declaration can span multiple source-code files—the separate portions of the class declaration in each file are known as partial classes. The `partial` modifier indicates that the class in a particular file is part of a larger class.
- Every Web Form class inherits from class `Page` in namespace `System.Web.UI`. Class `Page` represents the default capabilities of each page in a web app.
- The ASP.NET controls are defined in namespace `System.Web.UI.WebControls`.

Section 23.5 Standard Web Controls: Designing a Form

- An `Image` control's `ImageUrl` property specifies the location of the image to display.
- By default, the contents of a table cell are aligned vertically in the middle of the cell. You can change this with the cell's `valign` property.
- A `TextBox` control allows you to obtain text from the user and display text to the user.
- A `DropDownList` control is similar to the Windows `Forms ComboBox` control, but doesn't allow users to type text. You can add items to the `DropDownList` using the **ListItem Collection Editor**, which you can access by clicking the ellipsis next to the `DropDownList`'s `Items` property in the **Properties** window, or by using the **DropDownList Tasks** smart-tag menu.
- A `HyperLink` control adds a hyperlink to a Web Form. The `NavigateUrl` property specifies the resource or web page that will be requested when the user clicks the `HyperLink`.
- A `RadioButtonList` control provides a series of radio buttons from which the user can select only one. The **RadioButtonList Tasks** smart-tag menu provides an **Edit Items...** link to open the **ListItem Collection Editor** so that you can create the items in the list.
- A `Button` control triggers an action when clicked.

Section 23.6 Validation Controls

- A validation control determines whether the data in another web control is in the proper format.
- When the page is sent to the client, the validator is converted into `JavaScript` that performs the validation in the client web browser.
- Some client browsers might not support scripting or the user might disable it. For this reason, you should always perform validation on the server.
- A `RequiredFieldValidator` control ensures that its `ControlToValidate` is not empty when the form is submitted. The validator's `ErrorMessage` property specifies what to display on the Web Form if the validation fails. When the validator's `Display` property is set to `Dynamic`, the validator occupies space on the Web Form only when validation fails.
- A `RegularExpressionValidator` uses a regular expression to ensure data entered by the user is in a valid format. Visual Studio Express 2012 for Web provides several predefined regular expressions that you can simply select to validate e-mail addresses, phone numbers and more. A `RegularExpressionValidator`'s `ValidationExpression` property specifies the regular expression to use for validation.
- A Web Form's `Load` event occurs each time the page loads into a web browser. The event handler for this event is `Page_Load`.
- ASP.NET pages are often designed so that the current page reloads when the user submits the form; this enables the program to receive input, process it as necessary and display the results in the same page when it's loaded the second time.

- Submitting a web form is known as a postback. Class Page's IsPostBack property returns true if the page is being loaded due to a postback.
- Server-side Web Form validation must be implemented programmatically. Class Page's Validate method validates the information in the request as specified by the Web Form's validation controls. Class Page's IsValid property returns true if validation succeeded.

Section 23.7 Session Tracking

- Personalization makes it possible for e-businesses to communicate effectively with their customers and also improves users' ability to locate desired products and services.
- To provide personalized services to consumers, e-businesses must be able to recognize clients when they request information from a site.
- HTTP is a stateless protocol—it does not provide information regarding particular clients.
- Tracking individual clients is known as session tracking.

Section 23.7.1 Cookies

- A cookie is a piece of data stored in a small text file on the user's computer. A cookie maintains information about the client during and between browser sessions.
- The expiration date of a cookie determines how long the cookie remains on the client's computer. If you do not set an expiration date for a cookie, the web browser maintains the cookie for the duration of the browsing session.

Section 23.7.2 Session Tracking with **HttpSessionState**

- Session tracking is implemented with class HttpSessionState.

Section 23.7.3 **Options.aspx***: Selecting a Programming Language*

- Each radio button in a RadioButtonList has a Text property and a Value property. The Text property is displayed next to the radio button and the Value property represents a value that's sent to the server when the user selects that radio button and submits the form.
- Every Web Form includes a user-specific HttpSessionState object, which is accessible through property Session of class Page.
- HttpSessionState property SessionID contains a client's unique session ID. The first time a client connects to the web server, a unique session ID is created for that client and a temporary cookie is written to the client so the server can identify the client on subsequent requests. When the client makes additional requests, the client's session ID from that temporary cookie is compared with the session IDs stored in the web server's memory to retrieve the client's HttpSessionState object.
- HttpSessionState property Timeout specifies the maximum amount of time that an HttpSessionState object can be inactive before it's discarded. Twenty minutes is the default.
- The HttpSessionState object is a dictionary—a data structure that stores key–value pairs. A program uses the key to store and retrieve the associated value in the dictionary.
- The key–value pairs in an HttpSessionState object are often referred to as session items. They're placed in an HttpSessionState object by calling its Add method. Another common syntax for placing a session item in the HttpSessionState object is Session[*KeyName*] = *Value.*
- If an app adds a session item that has the same name as an item previously stored in the HttpSessionState object, the session item is replaced—session items names *must* be unique.

Section 23.7.4 `Recommendations.aspx`*: Displaying Recommendations Based on Session Values*

- The `Count` property returns the number of session items stored in an `HttpSessionState` object.
- `HttpSessionState`'s `Keys` property returns a collection containing all the keys in the session.

Section 23.8 Case Study: Database-Driven ASP.NET Guestbook

- A `GridView` data control displays data in tabular format. This control is located in the **Toolbox**'s **Data** section.

Section 23.8.1 Building a Web Form that Displays Data from a Database

- To use a SQL Server Express database file in an ASP.NET web app, you must first add the file to the project's `App_Data` folder. For security reasons, this folder can be accessed only by the web app on the server—clients cannot access this folder over a network. The web app interacts with the database on behalf of the client.
- An `EntityDataSource` control enables data bound Web Form controls to interact with a database via the ADO.NET Entity Framework.

Section 23.8.2 Modifying the Code-Behind File for the Guestbook App

- To insert data into a database from a web page, you create an object of your entity data model's class that represents a row in a table, set the object's properties, then use the `Add` method to add the object to the `DbContext`'s object that represents the database table. When you call `SaveChanges` on the `DbContext`, the new row is added to the database table.
- A `GridView`'s `DataBind` method refreshes the data that the `GridView` displays.

Terminology

`Add` method of class `HttpSessionState`
`Add` method of an entity data model class that represents a database table
ASP.NET
ASP.NET server control
ASPX file
`.aspx` filename extension
bottom tier
business logic
`Button` control
client tier
code-behind file
controller logic
`ControlToValidate` property of a validation control
cookie
`Count` property of class `HttpSessionState`
`DataBind` method of a `GridView`
`Display` property of a validation control
DNS (domain name system) server
`DOCUMENT` in the **Properties** window
`DropDownList` control
`EntityDataSource` control
`ErrorMessage` property of a validation control
expiration date of a cookie
`GridView` control
host
hostname
HTTP (HyperText Transfer Protocol)
`HttpSessionState` class
`HyperLink` control
IIS (Internet Information Services)
`Image` control
`ImageUrl` property of an `Image` web control
information tier
`Init` event of a Web Form
IP (Internet Protocol) address
`IsPostBack` property of `Page` class
`IsValid` property of `Page` class
key–value pair
`Keys` property of `HttpSessionState` class
`Load` event of Web Form
middle tier
multitier app
n-tier app
`NavigateUrl` property of a `HyperLink` control
`Page` class
`Page_Init` event handler

Page_Load event handler
partial class
partial modifier
personalization
postback
presentation logic
RadioButtonList control
RegularExpressionValidator validation control
RequiredFieldValidator control
session item
Session property of Page class
session tracking
SessionID property of HttpSessionState class
System.Web.UI namespace
System.Web.UI.WebControls namespace
Target property of a HyperLink control
TextBox control
tier in a multitier app
Timeout property of HttpSessionState class
Title property of a Web Form
top tier
unique session ID of an ASP.NET client
Validate property of Page class
validation control
ValidationExpression property of a RegularExpressionValidator control
validator
Visible property of an ASP.NET control
Visual Studio Express 2012 for Web
web-app development
web control
Web Form
web server
Web.config ASP.NET configuration file

Self-Review Exercises

23.1 State whether each of the following is *true* or *false*. If *false*, explain why.

a) Web Form file names end in .aspx.
b) App.config is a file that stores configuration settings for an ASP.NET web app.
c) A maximum of one validation control can be placed on a Web Form.
d) An EntityDataSource control enables data bound Web Form controls to interact with a database via the ADO.NET Entity Framework.

23.2 Fill in the blanks in each of the following statements:

a) Web apps contain three basic tiers: ________, ________, and ________.
b) The ________ web control is similar to the ComboBox Windows control.
c) A control which ensures that the data in another control is in the correct format is called a(n) ________.
d) A(n) ________ occurs when a page requests itself.
e) Every ASP.NET page inherits from class ________.
f) The ________ file contains the functionality for an ASP.NET page.

Answers to Self-Review Exercises

23.1 a) True. b) False. Web.config is the file that stores configuration settings for an ASP.NET web app. c) False. An unlimited number of validation controls can be placed on a Web Form. d) True.

23.2 a) bottom (information), middle (business logic), top (client). b) DropDownList. c) validator. d) postback. e) Page. f) code-behind.

Exercises

23.3 *(WebTime Modification)* Modify the WebTime example to contain drop-down lists that allow the user to modify such Label properties as BackColor, ForeColor and Font-Size. Configure these drop-down lists so that a postback occurs whenever the user makes a selection—to do this, set their AutoPostBack properties to true. When the page reloads, it should reflect the specified changes to the properties of the Label displaying the time.

23.4 *(Page Hit Counter)* Create an ASP.NET page that uses session tracking to keep track of how many times the client computer has visited the page. Set the `HttpSessionState` object's `Timeout` property to `1440` (the number of minutes in one day) to keep the session in effect for one day into the future. Display the number of page hits every time the page loads.

23.5 *(Guestbook App Modification)* Add validation to the guestbook app in Section 23.8. Use validation controls to ensure that the user provides a name, a valid e-mail address and a message.

23.6 *(Project: `WebControls` Modification)* Modify the example of Section 23.5 to add functionality to the **Register** `Button`. When the user clicks the `Button`, validate all of the input fields to ensure that the user has filled out the form completely and entered a valid email address and phone number. If any of the fields are not valid, appropriate messages should be displayed by validation controls. If the fields are all valid, direct the user to another page that displays a message indicating that the registration was successful followed by the registration information that was submitted from the form.

23.7 *(Project: Web-Based Address Book)* Using the techniques you learned in Section 23.8, create a web-based Address book with similar functionality to the **Address Book** app that you created in Section 22.9. Display the address book's contents in a `GridView`. Allow the user to search for entries with a particular last name.