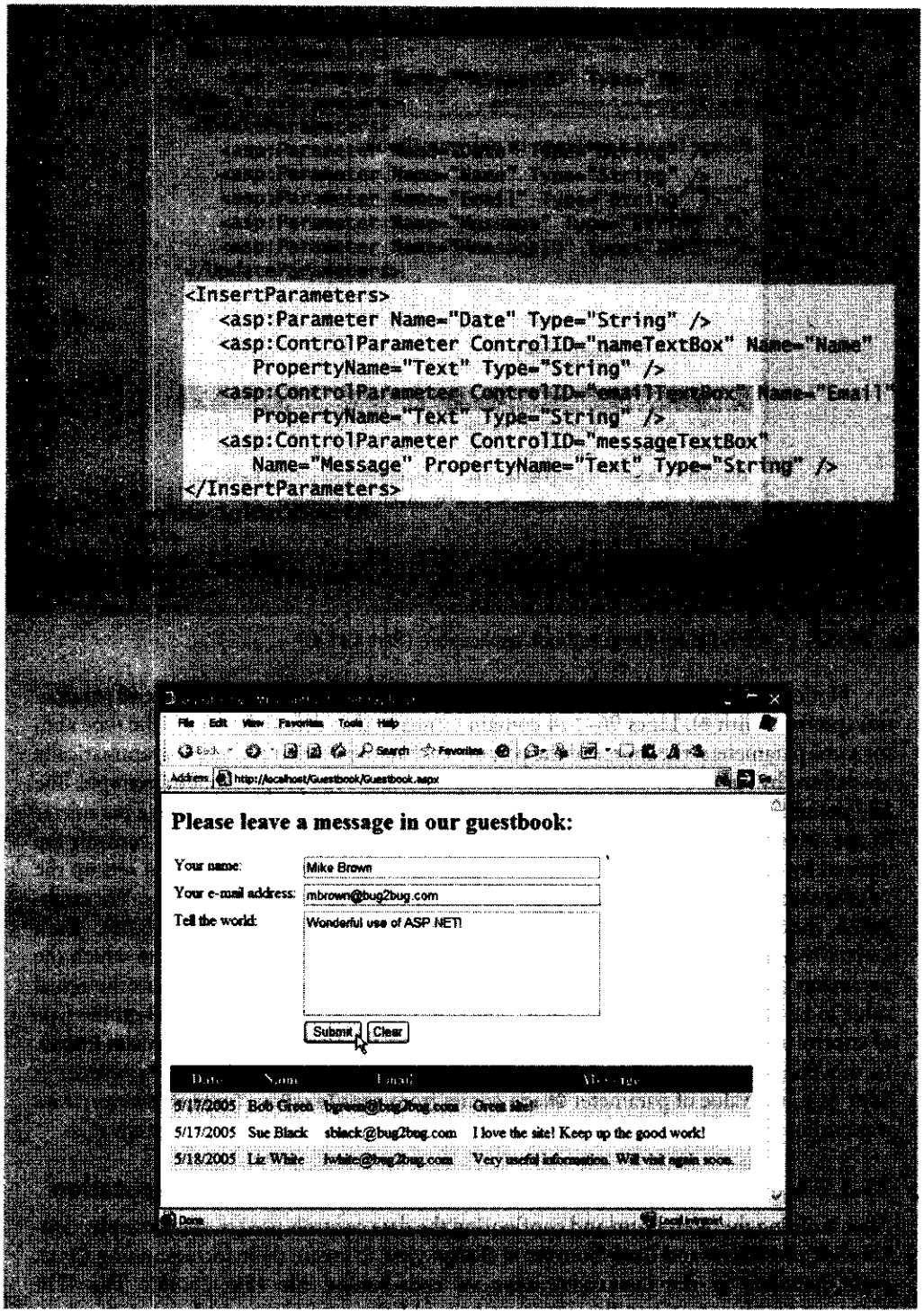


Fig. 25.37 | ASPX file for the guestbook application. (Part 3 of 4.)

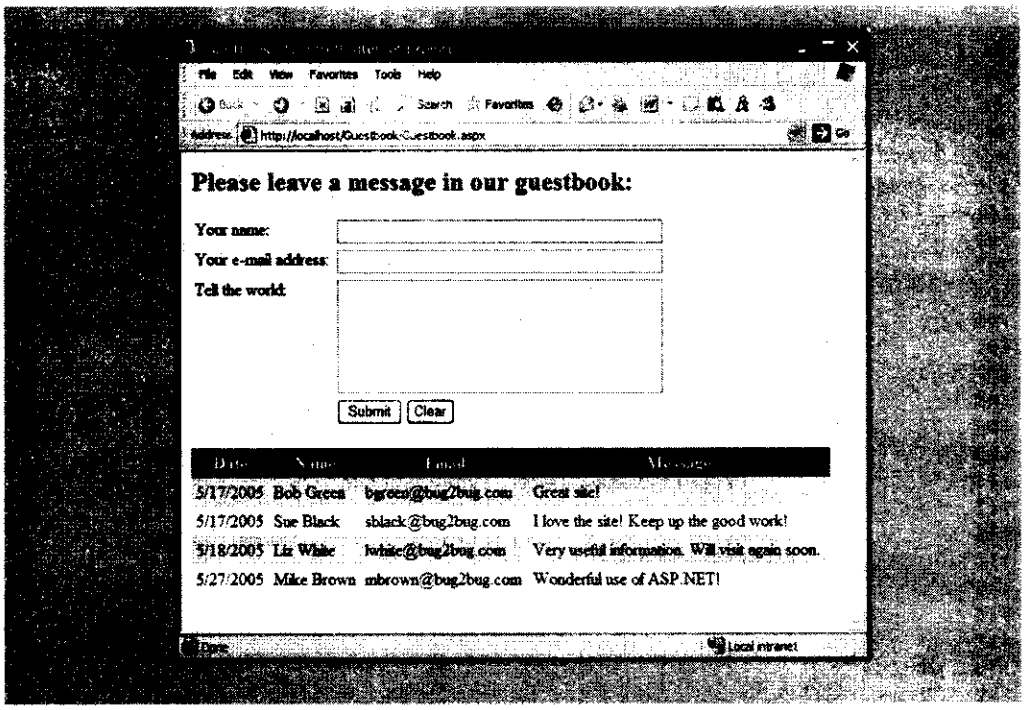
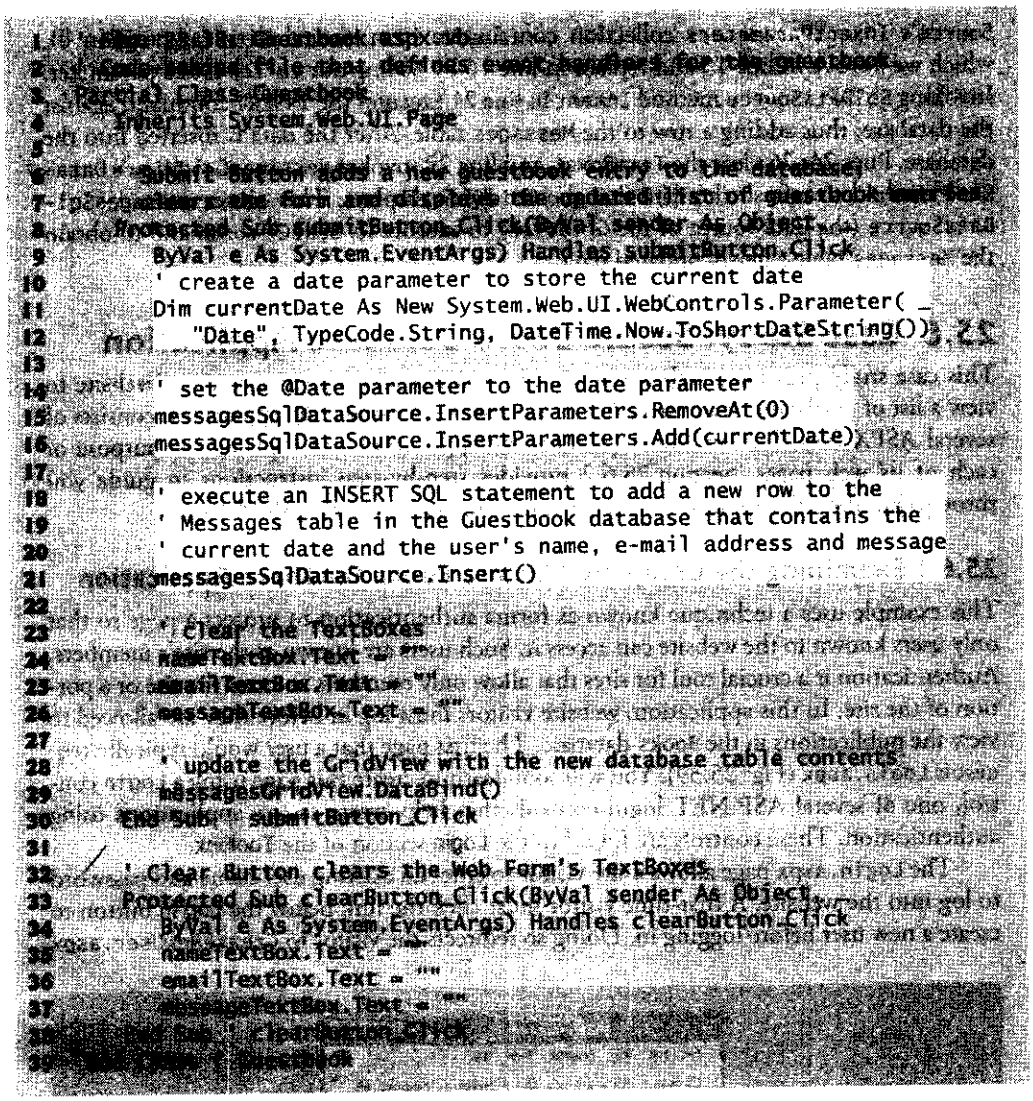


Fig. 25.37 | ASPX file for the guestbook application. (Part 4 of 4.)

Notice that the SQL commands used by the `SqlDataSource` contain several parameters (prefixed with `@`). Lines 96–114 contain elements that define the name, the type and, for some parameters, the source of the parameter. Parameters that are set programmatically are defined by **`Parameter`** elements containing `Name` and `Type` properties. For example, line 107 defines the `Date` parameter of `Type` `String`. This corresponds to the `@Date` parameter in the `InsertCommand` (line 91). Parameters that obtain their values from controls are defined by **`ControlParameter`** elements. Lines 108–113 contain markup that sets up the relationships between the `INSERT` parameters and the Web Form's `TextBoxes`. We established these relationships in the **Command and Parameter Editor** (Fig. 25.36). Each `ControlParameter` contains a `ControlID` property indicating the control from which the parameter gets its value. The `PropertyName` specifies the property that contains the actual value to be used as the parameter value. The IDE sets the `PropertyName` based on the type of control specified by the `ControlID` (indirectly via the **Command and Parameter Editor**). In this case, we use only `TextBoxes`, so the `PropertyName` of each `ControlParameter` is `Text` (e.g., the value of parameter `@Name` comes from `nameTextBox.Text`). However, if we were using a `DropDownList`, for example, the `PropertyName` would be `SelectedValue`.

25.5.2 Modifying the Code-Behind File for the Guestbook Application

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 `Guestbook.aspx.vb` code-behind file (Fig. 25.38). The IDE generates empty event handlers, so we must add the appropriate code to make these



```
1  ' Guestbook.aspx.vb
2  ' Code-behind file that defines event handlers for the Guestbook
3  Partial Class Guestbook
4     Inherits System.Web.UI.Page
5
6     ' Submit Button adds a new guestbook entry to the database
7     ' then clears and displays the updated list of guestbook entries
8     Protected Sub submitButton_Click(ByVal sender As Object, _
9        ByVal e As System.EventArgs) Handles submitButton.Click
10       ' create a date parameter to store the current date
11       Dim currentDate As New System.Web.UI.WebControls.Parameter( _
12          "Date", TypeCode.String, DateTime.Now.ToShortDateString())
13
14       ' set the @Date parameter to the date parameter
15       messagesSqlDataSource.InsertParameters.RemoveAt(0)
16       messagesSqlDataSource.InsertParameters.Add(currentDate)
17
18       ' execute an INSERT SQL statement to add a new row to the
19       ' Messages table in the Guestbook database that contains the
20       ' current date and the user's name, e-mail address and message
21       messagesSqlDataSource.Insert()
22
23       ' clear the TextBoxes
24       nameTextBox.Text = ""
25       emailTextBox.Text = ""
26       messageTextBox.Text = ""
27
28       ' update the GridView with the new database table contents
29       messagesGridView.DataBind()
30    End Sub ' submitButton_Click
31
32    ' Clear Button clears the Web Form's TextBoxes
33    Protected Sub clearButton_Click(ByVal sender As Object, _
34       ByVal e As System.EventArgs) Handles clearButton.Click
35       nameTextBox.Text = ""
36       emailTextBox.Text = ""
37       messageTextBox.Text = ""
38    End Sub ' clearButton_Click
39 End Class ' Guestbook
```

Fig. 25.38 | Code-behind file for the guestbook application.

buttons work properly. The event handler for `clearButton` (lines 33–38) clears each `TextBox` by setting its `Text` property to an empty string. This resets the form for a new guestbook submission.

Lines 8–30 contain the event-handling code for `submitButton`, which adds the user's information to the `Messages` table of the `Guestbook` database. Recall that we configured `messagesSqlDataSource`'s `INSERT` command to use the values of the `TextBoxes` on the Web Form as the parameter values inserted into the database. We have not yet specified the date value to be inserted, though. Lines 11–12 assign a `String` representation of the current date (e.g., `"3/27/06"`) to a new object of type `Parameter`. This `Parameter` object is identified as `"Date"` and is given the current date as a default value. The `SqlData-`

`Source`'s `InsertParameters` collection contains an item named `Date` (at position 0), which we `Remove` in line 15 and replace in line 16 by `Adding` our `currentDate` parameter. Invoking `SqlDataSource` method `Insert` in line 21 executes the `INSERT` command against the database, thus adding a row to the `Messages` table. After the data is inserted into the database, lines 24–26 clear the `TextBoxes`, and line 29 invokes `messagesGridView`'s `DataBind` method to refresh the data that the `GridView` displays. This causes `messagesSqlDataSource` (the data source of the `GridView`) to execute its `SELECT` command to obtain the `Messages` table's newly updated data.

25.6 Case Study: Secure Books Database Application

This case study presents a web application in which a user logs into a secure website to view a list of publications by an author of the user's choosing. The application consists of several ASPX files. Section 25.6.1 presents the application and explains the purpose of each of its web pages. Section 25.6.2 provides step-by-step instructions to guide you through building the application and presents the markup in the ASPX files.

25.6.1 Examining the Completed Secure Books Database Application

This example uses a technique known as **forms authentication** to protect a page so that only users known to the website can access it. Such users are known as the site's members. Authentication is a crucial tool for sites that allow only members to enter the site or a portion of the site. In this application, website visitors must log in before they are allowed to view the publications in the `Books` database. The first page that a user would typically request is `Login.aspx` (Fig. 25.39). You will soon learn to create this page using a `Login` control, one of several **ASP.NET login controls** that help create secure applications using authentication. These controls are found in the **Login** section of the **Toolbox**.

The `Login.aspx` page allows a site visitor to enter an existing user name and password to log into the website. A first-time visitor must click the link below the **Log In** button to create a new user before logging in. Doing so redirects the visitor to `CreateNewUser.aspx`

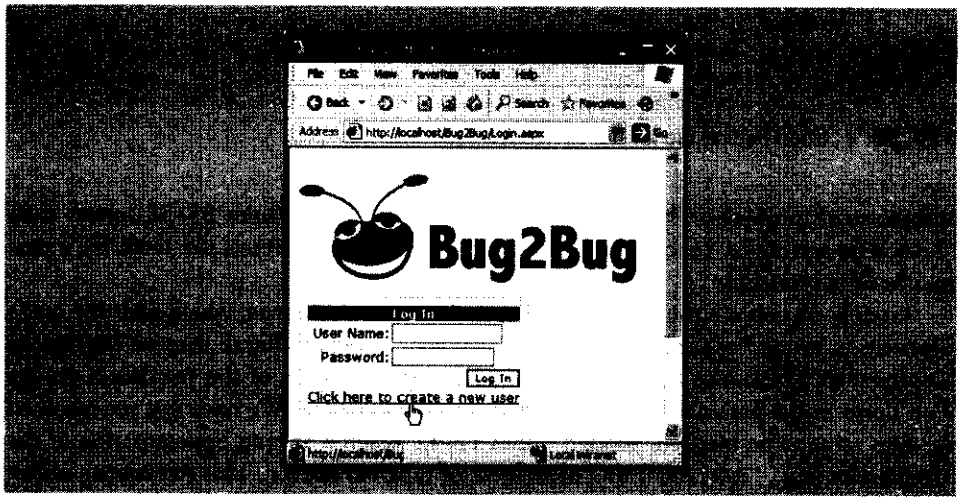


Fig. 25.39 | `Login.aspx` page of the secure books database application.

(Fig. 25.40), which contains a `CreateUserWizard` control that presents the visitor with a user registration form. We discuss the `CreateUserWizard` control in detail in Section 25.6.2. In Fig. 25.40, we use the password `pa$$word` for testing purposes—as you will learn, the `CreateUserWizard` requires that the password contain special characters for security purposes. Clicking **Create User** establishes a new user account. After creating the account, the user is automatically logged in and shown a success message (Fig. 25.41).

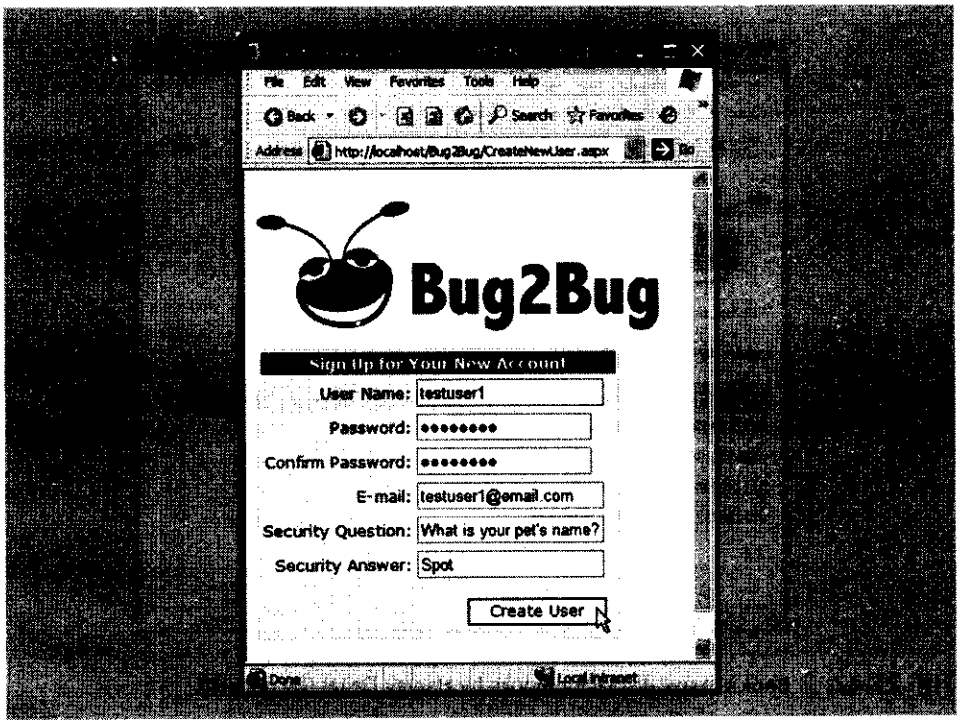


Fig. 25.40 | `CreateNewUser.aspx` page of the secure books database application.

Fig. 25.41 | Message displayed to indicate that a user account was created successfully.

Clicking the **Continue** button on the confirmation page sends the user to `Books.aspx` (Fig. 25.42), which provides a drop-down list of authors and a table containing the ISBNs, titles, edition numbers and copyright years of books in the database. By default, all the books by Harvey Deitel are displayed. Links appear at the bottom of the table that allow you to access additional pages of data. When the user chooses an author, a postback occurs, and the page is updated to display information about books written by the selected author (Fig. 25.43).

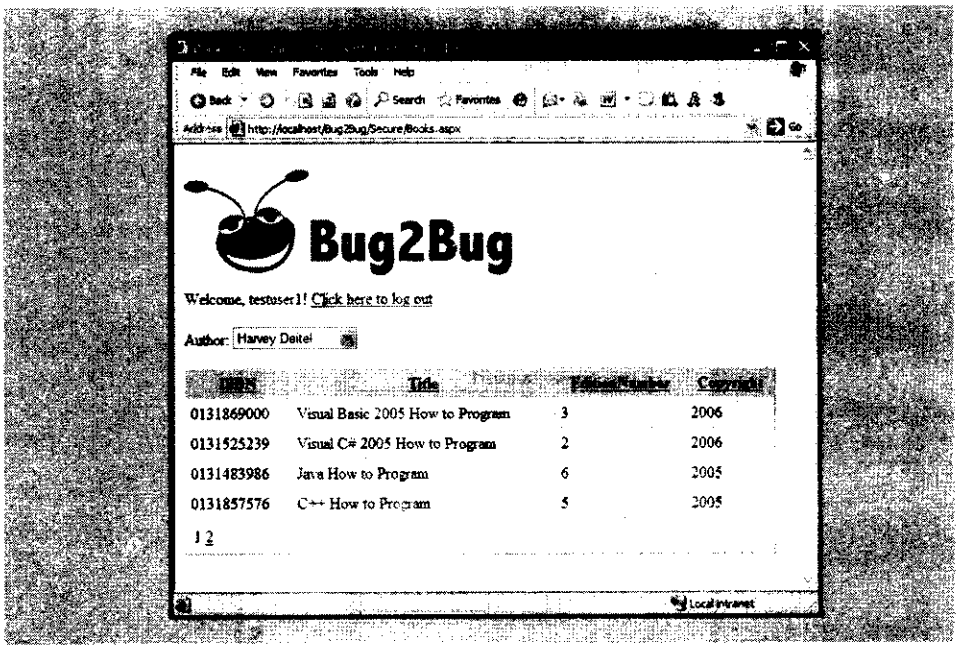


Fig. 25.42 | `Books.aspx` displaying books by Harvey Deitel (by default).

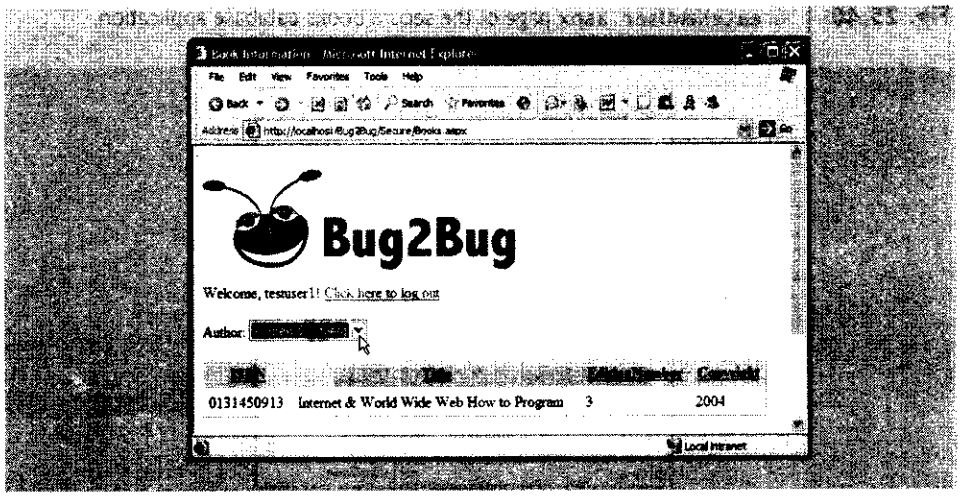


Fig. 25.43 | `Books.aspx` displaying books by Andrew Goldberg.

Note that once the user creates an account and is logged in, `Books.aspx` displays a welcome message customized for the particular logged-in user. As you will soon see, a `LoginName` control provides this functionality. After you add this control to the page, ASP.NET handles the details of determining the user name.

Clicking the **Click here to log out** link logs the user out, then sends the user back to `Login.aspx` (Fig. 25.44). This link is created by a `LoginStatus` control, which handles the log out details. After logging out, the user would need to log in through `Login.aspx` to view the book listing again. The `Login` control on this page receives the user name and password entered by a visitor. ASP.NET compares these values with user names and passwords stored in a database on the server. If there is a match, the visitor is **authenticated** (i.e., the user's identity is confirmed). We explain the authentication process in detail in Section 25.6.2. When an existing user is successfully authenticated, `Login.aspx` redirects the user to `Books.aspx` (Fig. 25.42). If the user's login attempt fails, an appropriate error message is displayed (Fig. 25.45).

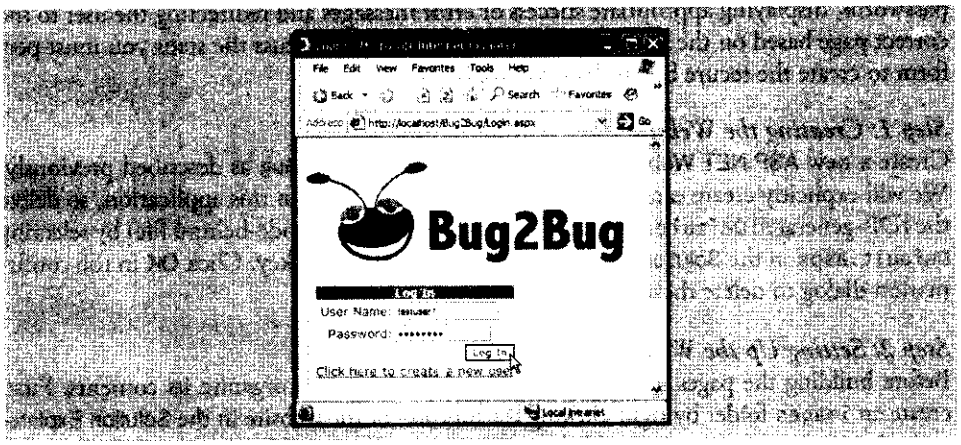


Fig. 25.44 | Logging in using the `Login` control.

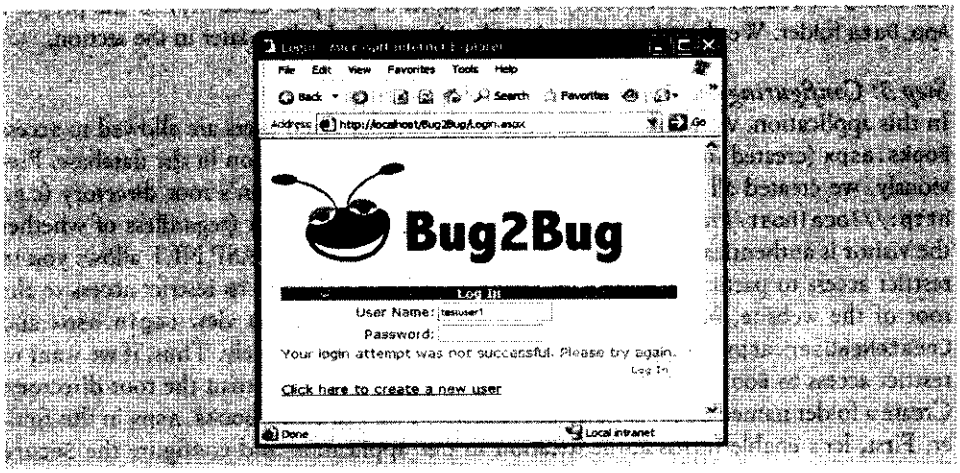


Fig. 25.45 | Error message displayed for an unsuccessful login attempt.

Notice that `Login.aspx`, `CreateNewUser.aspx` and `Books.aspx` share the same page header containing the logo image from the fictional company Bug2Bug. Instead of placing this image at the top of each page, we use a **master page** to achieve this. As we demonstrate shortly, a master page defines common GUI elements that are inherited by each page in a set of **content pages**. Just as Visual Basic classes can inherit instance variables and methods from existing classes, content pages inherit elements from master pages—this is known as **visual inheritance**.

25.6.2 Creating the Secure Books Database Application

Now that you are familiar with how this application behaves, you'll learn how to create it from scratch. Thanks to the rich set of login and data controls provided by ASP.NET, you will not have to write *any* code to create this application. In fact, the application does not contain any code-behind files. All of the functionality is specified through properties of controls, many of which are set through wizards and other visual programming tools. ASP.NET hides the details of authenticating users against a database of user names and passwords, displaying appropriate success or error messages and redirecting the user to the correct page based on the authentication results. We now discuss the steps you must perform to create the secure books database application.

Step 1: Creating the Website

Create a new **ASP.NET Web Site** at `http://localhost/Bug2Bug` as described previously. We will explicitly create each of the ASPX files that we need in this application, so delete the IDE-generated `Default.aspx` file (and its corresponding code-behind file) by selecting `Default.aspx` in the **Solution Explorer** and pressing the *Delete* key. Click **OK** in the confirmation dialog to delete these files.

Step 2: Setting Up the Website's Folders

Before building the pages in the website, we create folders to organize its contents. First, create an `Images` folder by right clicking the location of the website in the **Solution Explorer** and selecting **New Folder**, then add the `bug2bug.png` file to it. This image can be found in the examples directory for this chapter. Next, add the `Books.mdf` database file (located in the `exampleDatabases` subdirectory of the chapter's examples directory) to the project's `App_Data` folder. We show how to retrieve data from this database later in the section.

Step 3: Configuring the Application's Security Settings

In this application, we want to ensure that only authenticated users are allowed to access `Books.aspx` (created in *Step 9* and *Step 10*) to view the information in the database. Previously, we created all of our ASPX pages in the web application's root directory (e.g., `http://localhost/`*ProjectName*). By default, any website visitor (regardless of whether the visitor is authenticated) can view pages in the root directory. ASP.NET allows you to restrict access to particular folders of a website. We do not want to restrict access to the root of the website, however, because all users must be able to view `Login.aspx` and `CreateNewUser.aspx` to log in and create user accounts, respectively. Thus, if we want to restrict access to `Books.aspx`, it must reside in a directory other than the root directory. Create a folder named `Secure`. Later in the section, we will create `Books.aspx` in this folder. First, let's enable forms authentication in our application and configure the `Secure` folder to restrict access to authenticated users only.

Select **Website > ASP.NET Configuration** to open the **Web Site Administration Tool** in a web browser (Fig. 25.46). This tool allows you to configure various options that determine how your application behaves. Click either the **Security** link or the **Security** tab to open a web page in which you can set security options (Fig. 25.47), such as the type of authentication the application should use. In the **Users** column, click **Select authentication type**. On the resulting page (Fig. 25.48), select the radio button next to **From the internet** to indicate that users will log in via a form on the website in which the user can enter a username and password (i.e., the application will use forms authentication). The default setting—**From a local network**—relies on users' Windows user names and passwords for authentication purposes. Click the **Done** button to save this change.

Now that forms authentication is enabled, the **Users** column on the main page of the **Web Site Administration Tool** (Fig. 25.49) provides links to create and manage users. As you saw in Section 25.6.1, our application provides the `CreateNewUser.aspx` page in which users can create their own accounts. Thus, while it is possible to create users through the **Web Site Administration Tool**, we do not do so here.

Even though no users exist at the moment, we configure the `Secure` folder to grant access only to authenticated users (i.e., deny access to all unauthenticated users). Click the **Create access rules** link in the **Access Rules** column of the **Web Site Administration Tool** (Fig. 25.49) to view the **Add New Access Rule** page (Fig. 25.50). This page is used to create an **access rule**—a rule that grants or denies access to a particular web application directory for a specific user or group of users. Click the `Secure` directory in the left column of the page to identify the directory to which our access rule applies. In the middle column, select the radio button marked **Anonymous users** to specify that the rule applies to users who

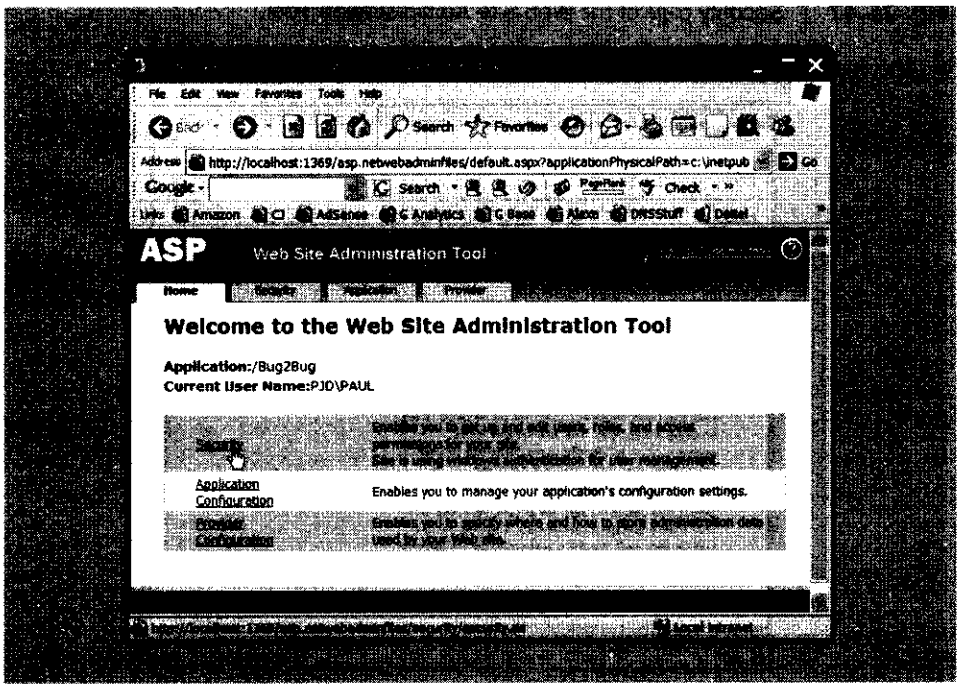


Fig. 25.46 | **Web Site Administration Tool** for configuring a web application.

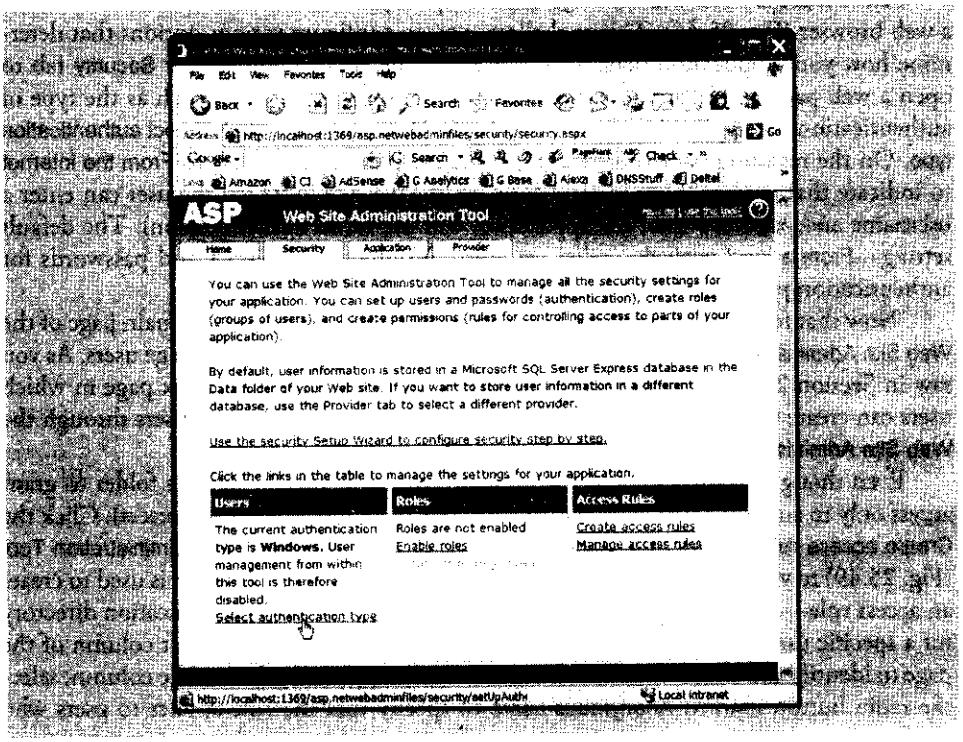


Fig. 25.47 | **Security** page of the **Web Site Administration Tool**.

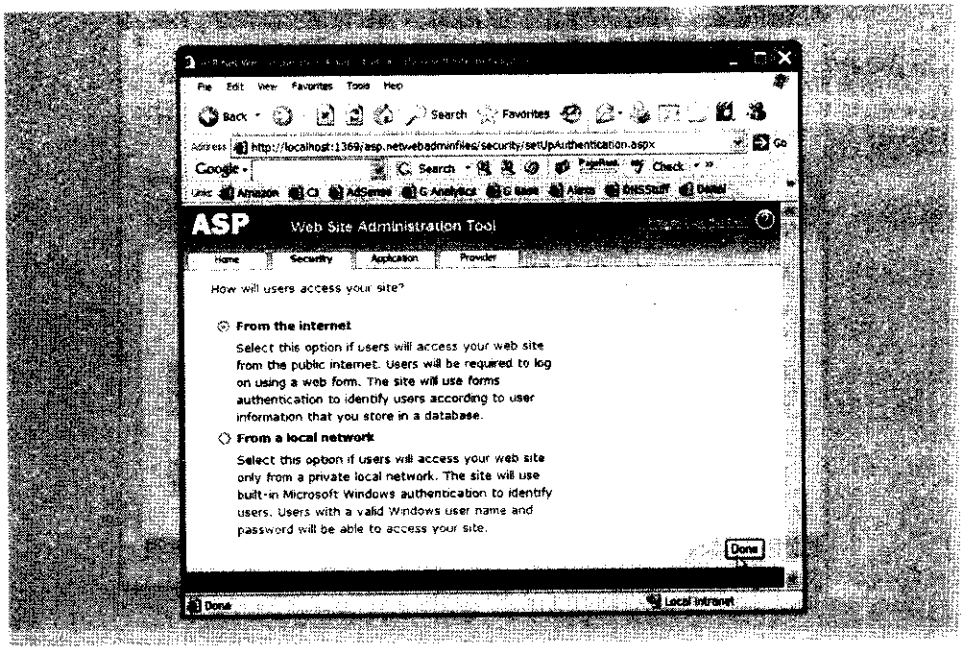


Fig. 25.48 | Choosing the type of authentication used by an ASP.NET web application.

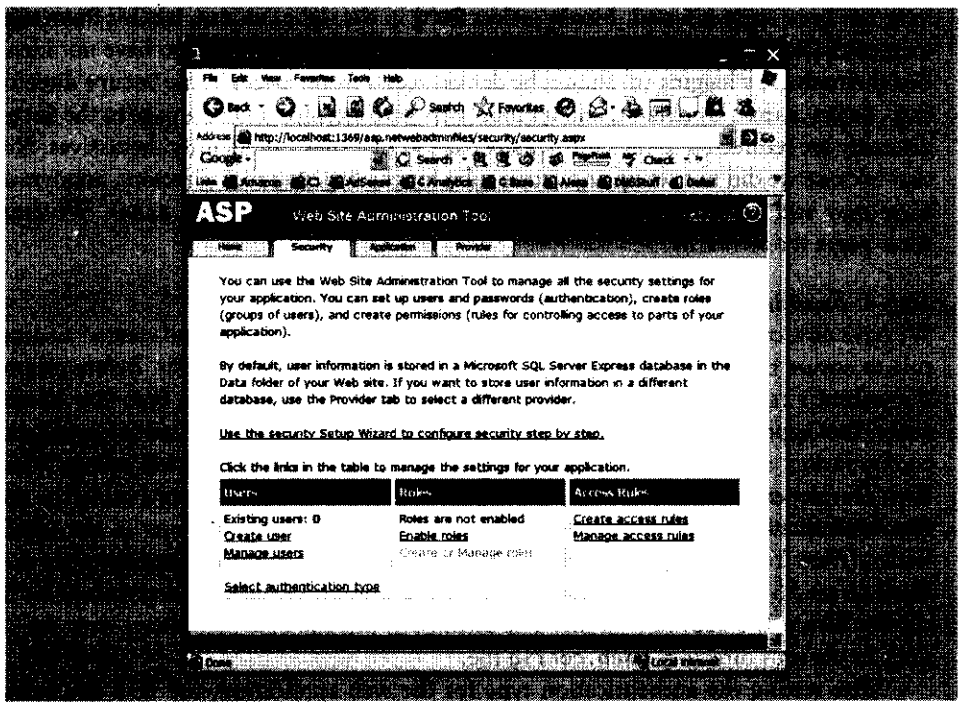


Fig. 25.49 | Main page of the **Web Site Administration Tool** after enabling forms authentication.

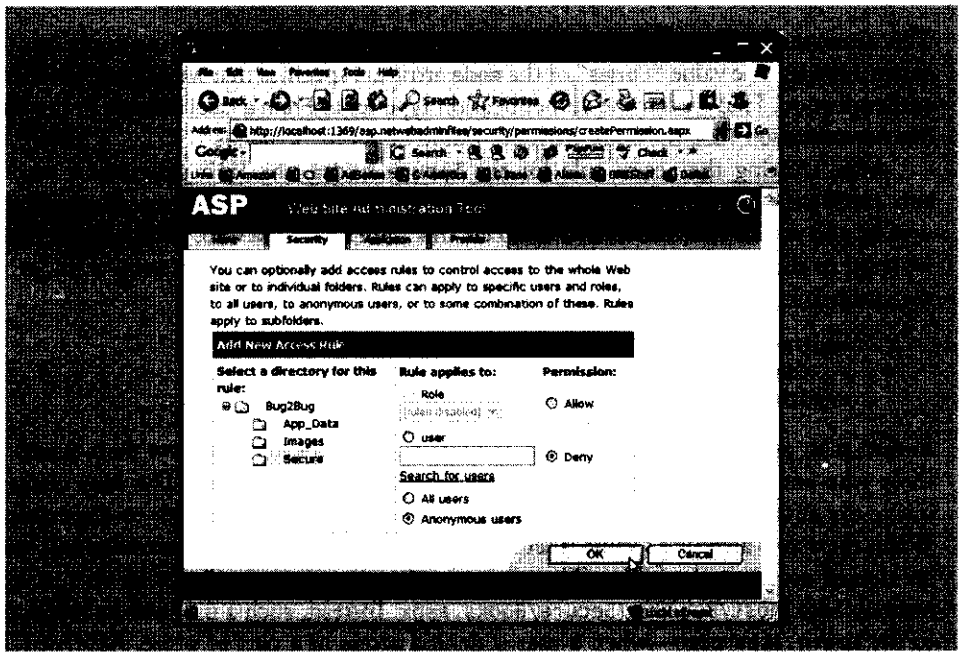


Fig. 25.50 | **Add New Access Rule** page used to configure directory access.

have not been authenticated. Finally, select **Deny** in the right column, labeled **Permission**, then click **OK**. This rule indicates that **anonymous users** (i.e., users who have not identified themselves by logging in) should be denied access to any pages in the `Secure` directory (e.g., `Books.aspx`). By default, anonymous users who attempt to load a page in the `Secure` directory are redirected to the `Login.aspx` page so that they can identify themselves. Note that because we did not set up any access rules for the `Bug2Bug` root directory, anonymous users may still access pages there (e.g., `Login.aspx`, `CreateNewUser.aspx`). We create these pages momentarily.

Step 4: Examining the Autogenerated `Web.config` *Files*

We have now configured the application to use forms authentication and created an access rule to ensure that only authenticated users can access the `Secure` folder. Before creating the website's content, we examine how the changes made through the **Web Site Administration Tool** appear in the IDE. Recall that `Web.config` is an XML file used for application configuration, such as enabling debugging or storing database connection strings. Visual Web Developer generates two `Web.config` files in response to our actions using the **Web Site Administration Tool**—one in the application's root directory and one in the `Secure` folder. [*Note:* You may need to click the **Refresh** button in the **Solution Explorer** to see these files.] In an ASP.NET application, a page's configuration settings are determined by the current directory's `Web.config` file. The settings in this file take precedence over the settings in the root directory's `Web.config` file.

After setting the authentication type for the web application, the IDE generates a `Web.config` file at `http://localhost/Bug2Bug/Web.config`, which contains an **`authentication`** element

```
<authentication mode="Forms" />
```

This element appears in the root directory's `Web.config` file, so the setting applies to the entire website. The value `"Forms"` of the **`mode`** attribute specifies that we want to use forms authentication. Had we left the authentication type set to **From a local network** in the **Web Site Administration Tool**, the `mode` attribute would be set to `"Windows"`.

After creating the access rule for the `Secure` folder, the IDE generates a second `Web.config` file in that folder. This file contains an **`authorization`** element that indicates who is, and who is not, authorized to access this folder over the web. In this application, we want to allow only authenticated users to access the contents of the `Secure` folder, so the authorization element appears as

```
<authorization>
   <deny users="?" />
</authorization>
```

Rather than grant permission to each individual authenticated user, we deny access to those who are not authenticated (i.e., those who have not logged in). The **`deny`** element inside the `authorization` element specifies the users to whom we wish to deny access. When the `users` attribute's value is set to `"?"`, all anonymous (i.e., unauthenticated) users are denied access to the folder. Thus, an unauthenticated user will not be able to load `http://localhost/Bug2Bug/Secure/Books.aspx`. Instead, such a user will be redirected to the `Login.aspx` page—when a user is denied access to a part of a site, ASP.NET by default sends the user to a page named `Login.aspx` in the application's root directory.

Step 5: Creating a Master Page

Now that you have established the application's security settings, you can create the application's web pages. We begin with the master page, which defines the elements we want to appear on each page. A master page is like a base class in a visual inheritance hierarchy, and content pages are like derived classes. The master page contains placeholders for custom content created in each content page. The content pages visually inherit the master page's content, then add content in place of the master page's placeholders.

For example, you might want to include a **navigation bar** (i.e., a series of buttons for navigating a website) on every page of a site. If the site encompasses a large number of pages, adding markup to create the navigation bar for each page can be time consuming. Moreover, if you subsequently modify the navigation bar, every page on the site that uses it must be updated. By creating a master page, you can specify the navigation bar markup in one file and have it appear on all the content pages, with only a few lines of markup. If the navigation bar changes, only the master page changes—any content pages that use it are updated the next time the page is requested.

In this example, we want the Bug2Bug logo to appear as a header at the top of every page, so we will place an `Image` control in the master page. Each subsequent page we create will be a content page based on this master page and thus will include the header. To create a master page, right click the location of the website in the **Solution Explorer** and select **Add New Item...**. In the **Add New Item** dialog, select **Master Page** from the template list and specify `Bug2Bug.master` as the filename. Master pages have the filename extension **`.master`** and, like Web Forms, can optionally use a code-behind file to define additional functionality. In this example, we do not need to specify any code for the master page, so leave the box labeled **Place code in a separate file** unchecked. Click **Add** to create the page.

The IDE opens the master page in **Source** mode (Fig. 25.51) when the file is first created. [*Note:* We added a line break in the `DOCTYPE` element for presentation purposes.] The

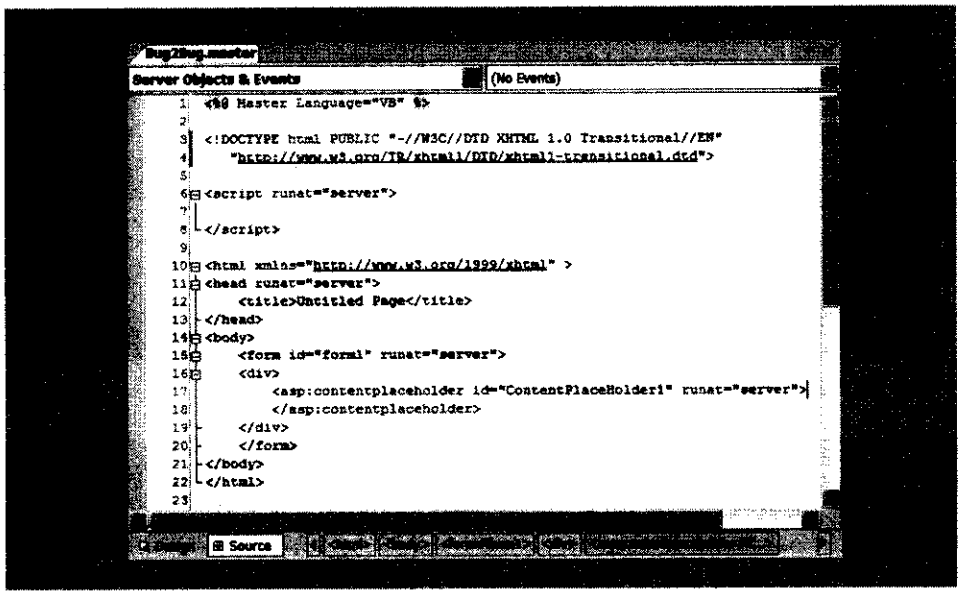


Fig. 25.51 | Master page in **Source** mode.

markup for a master page is almost identical to that of a Web Form. One difference is that a master page contains a **Master** directive (line 1 in Fig. 25.51), which specifies that this file defines a master page using the indicated Language for any code. Because we chose not to use a code-behind file, the master page also contains a **script** element (lines 6–8). Code that would usually be placed in a code-behind file can be placed in a script element. However, we remove the script element from this page, because we do not need to write any additional code. After deleting this block of markup, set the title of the page to Bug2Bug. Finally, notice that the master page contains a **ContentPlaceHolder** control (lines 17–18 of Fig. 25.51). This control serves as a placeholder for content that will be defined by a content page. You will see how to define content to replace the ContentPlaceHolder shortly.

At this point, you can edit the master page in **Design** mode (Fig. 25.52) as if it were an ASPX file. Notice that the ContentPlaceHolder control appears as a large rectangle with a gray bar indicating the control's type and ID. Using the **Properties** window, change the ID of this control to bodyContent.

To create a header in the master page that will appear at the top of each content page, we insert a table into the master page. Place the cursor to the left of the ContentPlaceHolder and select **Layout > Insert Table**. In the **Insert Table** dialog, click the **Template** radio button, then select **Header** from the drop-down list of available table templates. Click **OK** to create a table that fills the page and contains two rows. Drag and drop the ContentPlaceHolder into the bottom table cell. Change the valign property of this cell to top, so the ContentPlaceHolder vertically aligns with the top of the cell. Next, set the Height of the top table cell to 130. Add to this cell an Image control named headerImage with its ImageUrl property set to the bug2bug.png file in the project's Images folder. Figure 25.53 shows the markup and **Design** view of the completed master page. As you will see in *Step*

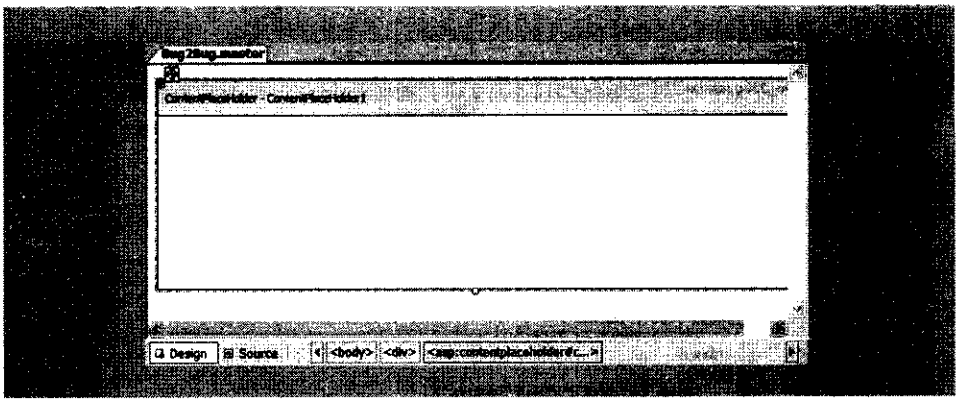


Fig. 25.52 | Master page in **Design** mode.

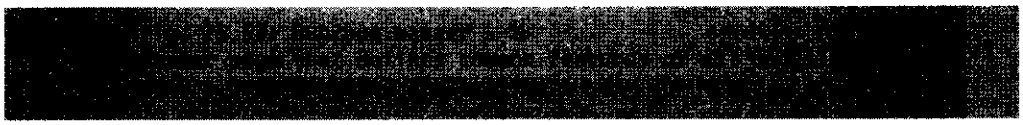


Fig. 25.53 | Bug2Bug.master page that defines a logo image header for all pages in the secure book database application. (Part 1 of 2.)

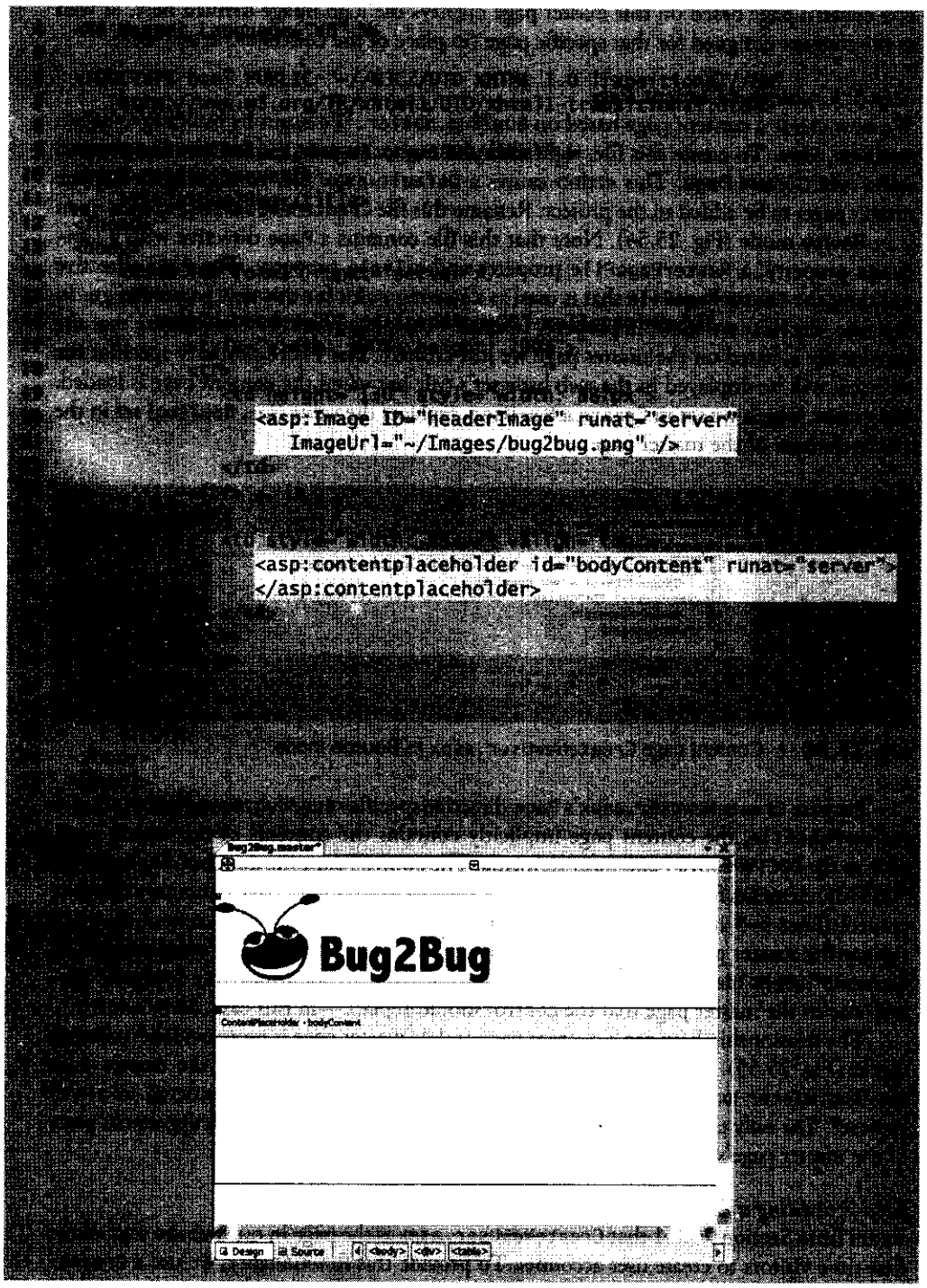


Fig. 25.53 | `Bug2Bug.master` page that defines a logo image header for all pages in the secure book database application. (Part 2 of 2.)

6, a content page based on this master page displays the logo image defined here, as well as the content designed for that specific page (in place of the `ContentPlaceHolder`).

Step 6: Creating a Content Page

We now create a content page based on `Bug2Bug.master`. We begin by building `CreateNewUser.aspx`. To create this file, right click the master page in the **Solution Explorer** and select **Add Content Page**. This action causes a `Default.aspx` file, configured to use the master page, to be added to the project. Rename this file `CreateNewUser.aspx`, then open it in **Source** mode (Fig. 25.54). Note that this file contains a `Page` directive with a `Language` property, a `MasterPageFile` property and a `Title` property. The `Page` directive indicates the **`MasterPageFile`** that is used as a starting point for this new page's design. In this case, the `MasterPageFile` property is set to `"~/Bug2Bug.master"` to indicate that the current file is based on the master page we just created. The **`Title`** property specifies the title that will be displayed in the web browser's title bar when the content page is loaded. This value, which we set to `Create a New User`, replaces the value (i.e., `Bug2Bug`) set in the `title` element of the master page.

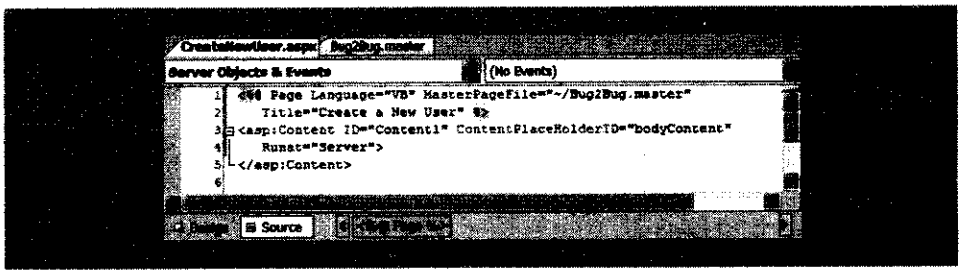


Fig. 25.54 | Content page `CreateNewUser.aspx` in **Source** mode.

Because `CreateNewUser.aspx`'s `Page` directive specifies `Bug2Bug.master` as the page's `MasterPageFile`, the content page implicitly contains the contents of the master page, such as the `DOCTYPE`, `html` and `body` elements. The content page file does not duplicate the XHTML elements found in the master page. Instead, the content page contains a **`Content`** control (lines 3–5 in Fig. 25.54), in which we will place page-specific content that will replace the master page's `ContentPlaceHolder` when the content page is requested. The `ContentPlaceHolderID` property of the `Content` control identifies the `ContentPlaceHolder` in the master page that the control should replace—in this case, `bodyContent`.

The relationship between a content page and its master page is more evident in **Design** mode (Fig. 25.55). The gray shaded region contains the contents of the master page `Bug2Bug.master` as they will appear in `CreateNewUser.aspx` when rendered in a web browser. The only editable part of this page is the `Content` control, which appears in place of the master page's `ContentPlaceHolder`.

Step 7: Adding a `CreateUserWizard` Control to a Content Page

Recall from Section 25.6.1 that `CreateNewUser.aspx` is the page in our website that allows first-time visitors to create user accounts. To provide this functionality, we use a **`CreateUserWizard`** control. Place the cursor inside the `Content` control in **Design** mode and double click `CreateUserWizard` in the **Login** section of the **Toolbox** to add it to the page at the

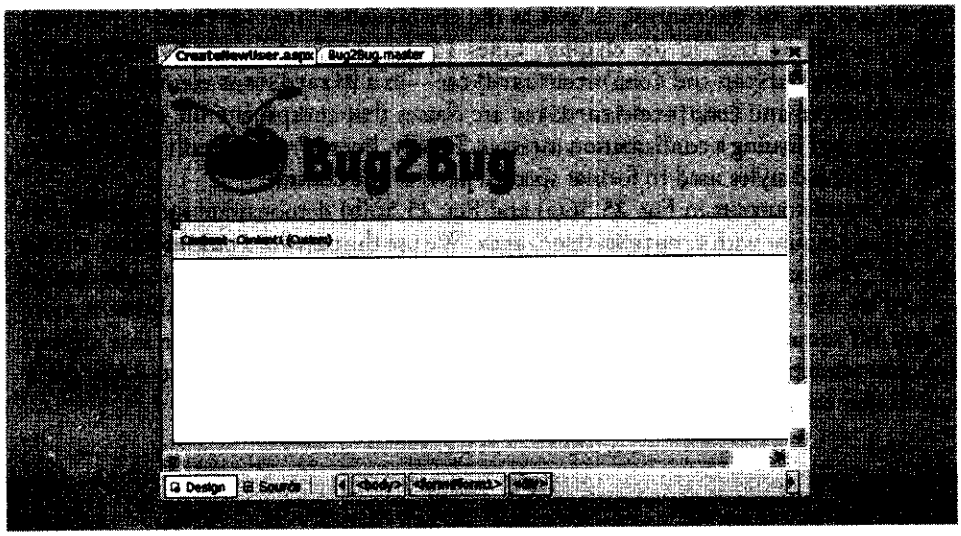


Fig. 25.55 | Content page `CreateNewUser.aspx` in **Design** mode.

current cursor position. You can also drag-and-drop the control onto the page. To change the `CreateUserWizard`'s appearance, open the **CreateUserWizard Tasks** smart tag menu, and click **Auto Format**. Select the **Professional** color scheme.

As discussed previously, a `CreateUserWizard` provides a registration form that site visitors can use to create a user account. ASP.NET creates a SQL Server database (named `ASPNETDB.MDF` and located in the `App_Data` folder) to store the user names, passwords and other account information of the application's users. ASP.NET also enforces a default set of requirements for filling out the form. Each field on the form is required, the password must contain at least seven characters (including at least one nonalphanumeric character) and the two passwords entered must match. The form also asks for a security question and answer that can be used to identify a user in case the user needs to reset or recover the account's password.

After the user fills in the form's fields and clicks the **Create User** button to submit the account information, ASP.NET verifies that all the form's requirements were fulfilled and attempts to create the user account. If an error occurs (e.g., the user name already exists), the `CreateUserWizard` displays a message below the form. If the account is created successfully, the form is replaced by a confirmation message and a button that allows the user to continue. You can view this confirmation message in **Design** mode by selecting **Complete** from the **Step** drop-down list in the **CreateUserWizard Tasks** smart tag menu.

When a user account is created, ASP.NET automatically logs the user into the site (we say more about the login process shortly). At this point, the user is authenticated and allowed to access the `Secure` folder. After we create `Books.aspx` later in this section, we set the `CreateUserWizard`'s `ContinueDestinationPageUrl` property to `~/Secure/Books.aspx` to indicate that the user should be redirected to `Books.aspx` after clicking the **Continue** button on the confirmation page.

Figure 25.56 presents the completed `CreateNewUser.aspx` file (reformatted for readability). Inside the `Content` control, the `CreateUserWizard` control is defined by the markup in lines 7–36. The start tag (lines 7–10) contains several properties that specify

formatting styles for the control, as well as the `ContinueDestinationPageUrl` property, which you will set later in the chapter. Lines 11–16 specify the wizard's two steps—`CreateUserWizardStep` and `CompleteWizardStep`—in a `WizardSteps` element. `CreateUserWizardStep` and `CompleteWizardStep` are classes that encapsulate the details of creating a user and issuing a confirmation message. Finally, lines 17–35 contain elements that define additional styles used to format specific parts of the control.

The sample outputs in Fig. 25.56(a) and Fig. 25.56(b) demonstrate successfully creating a user account with `CreateNewUser.aspx`. We use the password `pa$$word` for testing purposes. This password satisfies the minimum length and special character requirement imposed by ASP.NET, but in a real application, you should use a password that is more difficult for someone to guess. Figure 25.56(c) illustrates the error message that appears when you attempt to create a second user account with the same user name—ASP.NET requires that each user name be unique.

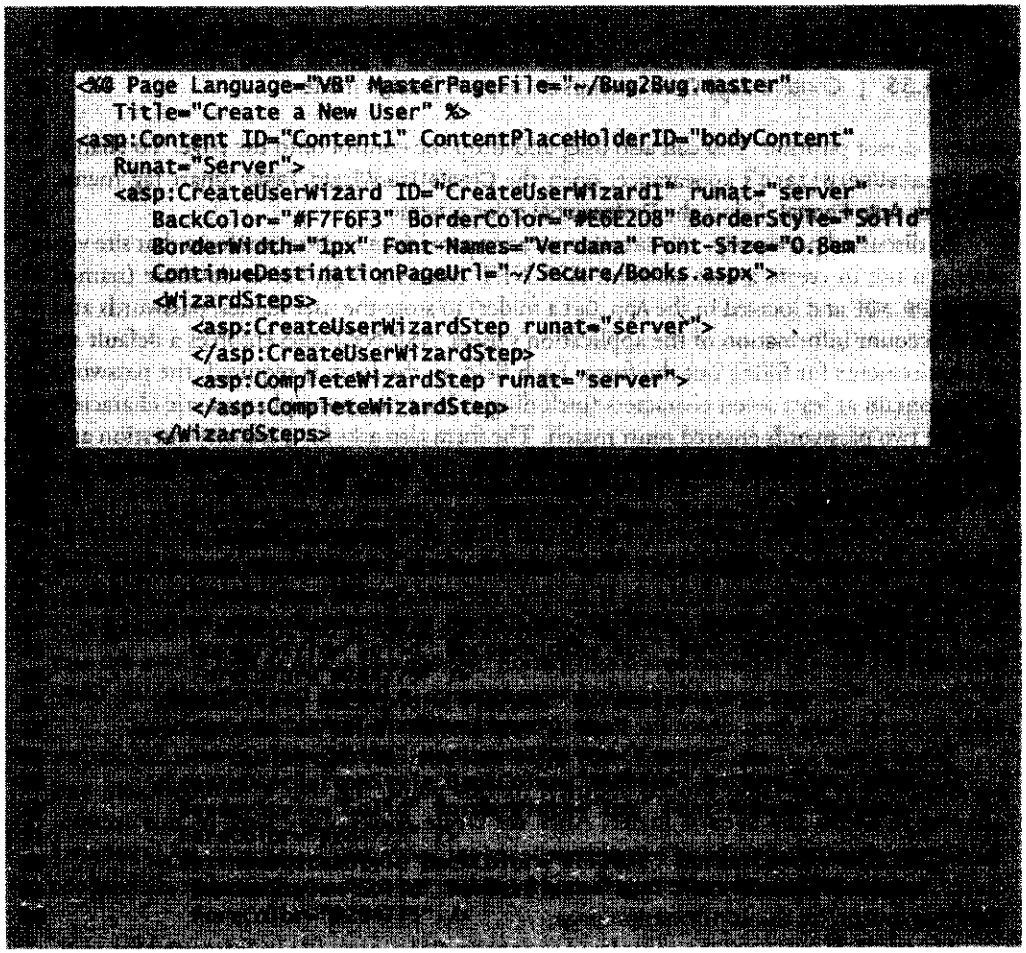


Fig. 25.56 | `CreateNewUser.aspx` content page that provides a user registration form. (Part 1 of 2.)

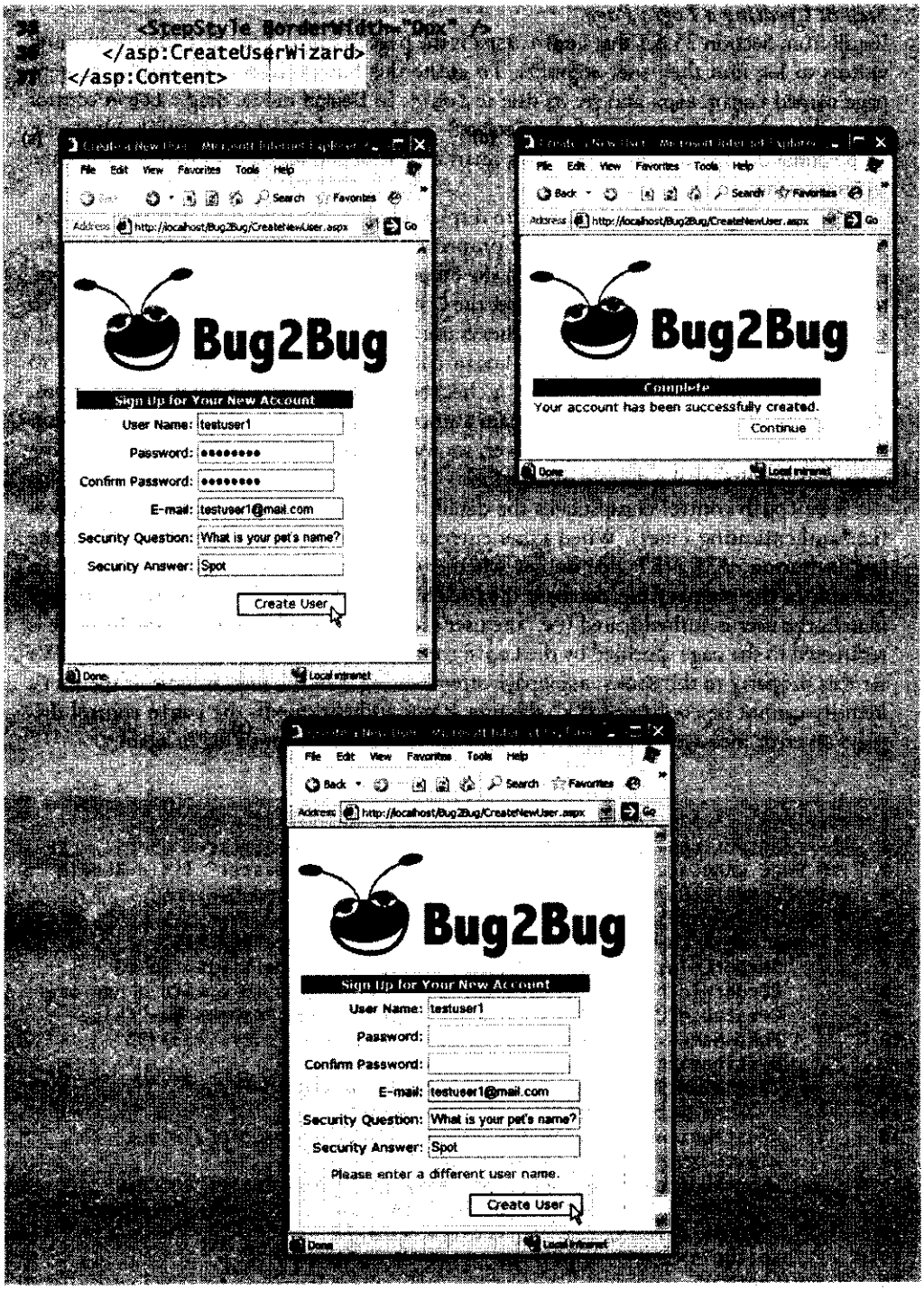


Fig. 25.56 | `CreateNewUser.aspx` content page that provides a user registration form. (Part 2 of 2.)

Step 8: Creating a Login Page

Recall from Section 25.6.1 that `Login.aspx` is the page in our website that allows returning visitors to log into their user accounts. To create this functionality, add another content page named `Login.aspx` and set its title to `Login`. In **Design** mode, drag a **Login** control (located in the **Login** section of the **Toolbox**) to the page's `Content` control. Open the **Auto Format** dialog from the **Login Tasks** smart tag menu and set the control's color scheme to **Professional**.

Next, configure the `Login` control to display a link to the page for creating new users. Set the `Login` control's `CreateUserUrl` property to `CreateNewUser.aspx` by clicking the ellipsis button to the property's right in the **Properties** window and selecting the `CreateNewUser.aspx` file in the dialog. Then set the `CreateUserText` property to `Click here to create a new user`. These property values cause a link to appear in the `Login` control.

Finally, change the value of the `Login` control's `DisplayRememberMe` property to `False`. By default, the control displays a checkbox and the text `Remember me next time`. This can be used to allow a user to remain authenticated beyond a single browser session on the user's current computer. However, we want to require that users log in each time they visit the site, so we disable this option.

The `Login` control encapsulates the details of logging a user into a web application (i.e., authenticating a user). When a user enters a user name and password, then clicks the **Log In** button, ASP.NET determines whether the items provided match those of an account in the membership database (i.e., `ASPNETDB.MDF` created by ASP.NET). If they match, the user is authenticated (i.e., the user's identity is confirmed), and the browser is redirected to the page specified by the `Login` control's `DestinationPageUrl` property. We set this property to the `Books.aspx` page after creating it in the next section. If the user's identity cannot be confirmed (i.e., the user is not authenticated), the `Login` control displays an error message (see Fig. 25.57), and the user can attempt to log in again.

```
1  <%-- Fig. 25.57: Login.aspx --%>
2  <%-- Content page using a Login control that authenticates users. --%>
3  <%@ Page Language="VB" MasterPageFile="~/Bug2Bug.master" Title="Login" %>
4  <asp:Content ID="Content1" ContentPlaceHolderID="bodyContent"
5     runat="Server">
6     <asp:Login ID="Login1" runat="server" BackColor="#F7F6F3"
7        BorderColor="#E6E2D8" BorderPadding="4" BorderStyle="Solid"
8        BorderWidth="1px" CreateUserText="Click here to create a new user"
9        CreateUserUrl="~/CreateNewUser.aspx" DisplayRememberMe="False"
10       Font-Names="Verdana" Font-Size="0.8em" ForeColor="#333333"
11       DestinationPageUrl="~/Secure/Books.aspx">
12       <TitleTextStyle BackColor="#5D7B9D" Font-Bold="True"
13          Font-Size="0.9em" ForeColor="White" />
14       <InstructionTextStyle Font-Italic="True" ForeColor="Black" />
15       <TextBoxStyle Font-Size="0.8em" />
16       <LoginButtonStyle BackColor="#FFFBFF" BorderColor="#CCCCCC"
17          BorderStyle="Solid" BorderWidth="1px" Font-Names="Verdana"
18          Font-Size="0.8em" ForeColor="#284775" />
19    </asp:Login>
20 </asp:Content>
```

Fig. 25.57 | `Login.aspx` content page using a `Login` control. (Part 1 of 2.)

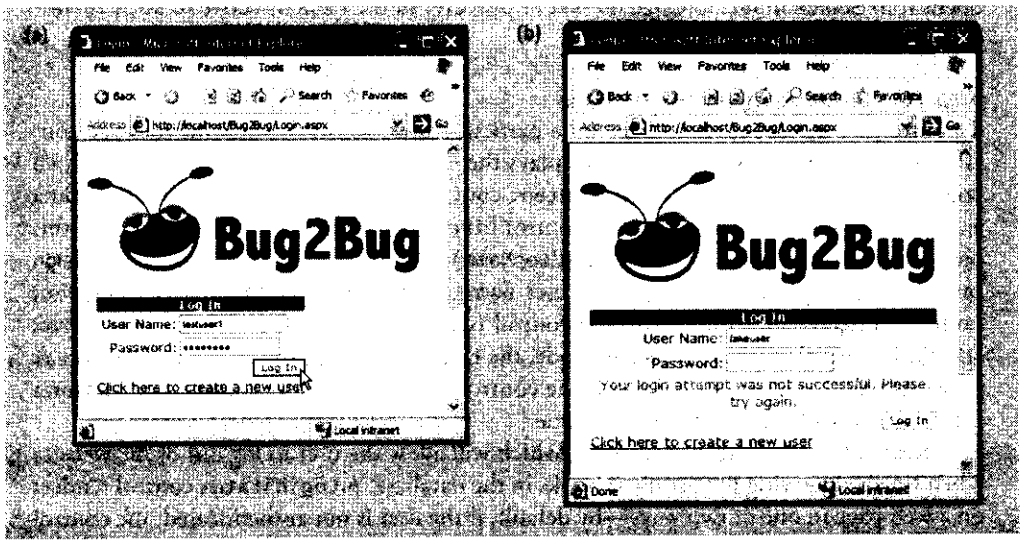


Fig. 25.57 | `Login.aspx` content page using a `Login` control. (Part 2 of 2.)

Figure 25.57 presents the completed `Login.aspx` file. Note that, as in `CreateNewUser.aspx`, the `Page` directive indicates that this content page inherits content from `Bug2Bug.master`. In the `Content` control that replaces the master page's `ContentPlaceHolder` with `ID` `bodyContent`, lines 6–19 create a `Login` control. Note the `CreateUserText` and `CreateUserUrl` properties (lines 8–9) that we set using the **Properties** window. Line 11 in the start tag for the `Login` control contains the `DestinationPageUrl` (you will set this property in the next step). The elements in lines 12–18 define various formatting styles applied to parts of the control. Note that all of the functionality related to actually logging the user in or displaying error messages is completely hidden from you.

When a user enters the user name and password of an existing user account, ASP.NET authenticates the user and writes to the client an **encrypted** cookie containing information about the authenticated user. Encrypted data is data translated into a code that only the sender and receiver can understand—thereby keeping it private. The encrypted cookie contains a `String` user name and a `Boolean` value that specifies whether this cookie should persist (i.e., remain on the client's computer) beyond the current session. Our application authenticates the user only for the current session.

Step 9: Creating a Content Page That Only Authenticated Users Can Access

A user who has been authenticated will be redirected to `Books.aspx`. We now create the `Books.aspx` file in the `Secure` folder—the folder for which we set an access rule denying access to anonymous users. If an unauthenticated user requests this file, the user will be redirected to `Login.aspx`. From there, the user can either log in or a create a new account, both of which will authenticate the user, thus allowing the user to return to `Books.aspx`.

To create `Books.aspx`, right click the `Secure` folder in the **Solution Explorer** and select **Add New Item...**. In the resulting dialog, select **Web Form** and specify the filename `Books.aspx`. Check the box **Select Master Page** to indicate that this Web Form should be created as a content page that references a master page, then click **Add**. In the **Select a Master Page** dialog, select `Bug2Bug.master` and click **OK**. The IDE creates the file and

opens it in **Source** mode. Change the `Title` property of the `Page` directive to `Book Information`.

Step 10: Customizing the Secure Page

To customize the `Books.aspx` page for a particular user, we add a welcome message containing a **`LoginName`** control, which displays the current authenticated user name. Open `Books.aspx` in **Design** mode. In the `Content` control, type `Welcome` followed by a comma and a space. Then drag a `LoginName` control from the **Toolbox** onto the page. When this page executes on the server, the text `[UserName]` that appears in this control in **Design** mode will be replaced by the current user name. In **Source** mode, type an exclamation point (`!`) directly after the `LoginName` control (with no spaces in between). [*Note:* If you add the exclamation point in **Design** mode, the IDE may insert extra spaces or a line break between this character and the preceding control. Entering the `!` in **Source** mode ensures that it appears adjacent to the user's name.]

Next, add a `LoginStatus` control, which will allow the user to log out of the website when finished viewing the listing of books in the database. A **`LoginStatus`** control renders on a web page in one of two ways—by default, if the user is not authenticated, the control displays a hyperlink with the text `Login`; if the user is authenticated, the control displays a hyperlink with the text `Logout`. Each link performs the stated action. Add a `LoginStatus` control to the page by dragging it from the **Toolbox** onto the page. In this example, any user who reaches this page must already be authenticated, so the control will always render as a `Logout` link. The **LoginStatus Tasks** smart tag menu allows you switch between the control's **Views**. Select the **Logged In** view to see the `Logout` link. To change the actual text of this link, modify the control's `LogoutText` property to `Click here to log out`. Next, set the `LogoutAction` property to `RedirectToLoginPage`.

Step 11: Connecting the `CreateUserWizard` and `Login` Controls to the Secure Page

Now that we have created `Books.aspx`, we can specify that this is the page to which the `CreateUserWizard` and `Login` controls redirect users after they are authenticated. Open `CreateNewUser.aspx` in **Design** mode and set the `CreateUserWizard` control's `ContinueDestinationPageUrl` property to `Books.aspx`. Next, open `Login.aspx` and select `Books.aspx` as the `DestinationPageUrl` of the `Login` control.

At this point, you can run the web application by selecting **Debug > Start Without Debugging**. First, create a user account on `CreateNewUser.aspx`, then notice how the `LoginName` and `LoginStatus` controls appear on `Books.aspx`. Next, log out of the site and log back in using `Login.aspx`.

Step 12: Generating a `DataSet` Based on the `Books.mdf` Database

Now, let's add the content (i.e., book information) to the secure page `Books.aspx`. This page will provide a `DropDownList` containing authors' names and a `GridView` displaying information about books written by the author selected in the `DropDownList`. A user will select an author from the `DropDownList` to cause the `GridView` to display information about only the books written by the selected author. As you will see, we create this functionality entirely in **Design** mode without writing any code.

To work with the `Books` database, we use an approach slightly different than in the preceding case study, in which we accessed the `Guestbook` database using a `SqlDataSource` control. Here we use an **`ObjectDataSource`** control, which encapsulates an object that

provides access to a data source. An `ObjectDataSource` can encapsulate a `TableAdapter` and use its methods to access the data in the database. This helps separate the data-access logic from the presentation logic. As you will see shortly, the SQL statements used to retrieve data do not appear in the ASPX page when using an `ObjectDataSource`.

The first step in accessing data using an `ObjectDataSource` is to create a `DataSet` that contains the data from the `Books` database required by the application. In Visual Basic 2005 Express, this occurs automatically when you add a data source to a project. In Visual Web Developer, however, you must explicitly generate the `DataSet`. Right click the project's location in the **Solution Explorer** and select **Add New Item...**. In the resulting dialog, select **DataSet** and specify `BooksDataSet.xsd` as the filename, then click **Add**. A dialog will appear that asks you whether the `DataSet` should be placed in an `App_Code` folder—a folder whose contents are compiled and made available to all parts of the project. Click **Yes** for the IDE to create this folder to store `BooksDataSet.xsd`.

Step 13: Creating and Configuring an `AuthorsTableAdapter`

Once the `DataSet` is added, the **Dataset Designer** will appear, and the **TableAdapter Configuration Wizard** will open. This wizard allows you to configure a `TableAdapter` for filling a `DataTable` in a `DataSet` with data from a database. The `Books.aspx` page requires two sets of data—a list of authors that will be displayed in the page's `DropDownList` (created shortly) and a list of books written by a specific author. We focus on the first set of data here—the authors. Thus, we use the **TableAdapter Configuration Wizard** first to configure an `AuthorsTableAdapter`. In the next step, we will configure a `TitlesTableAdapter`.

In the **TableAdapter Configuration Wizard**, select `Books.mdf` from the drop-down list. Then click **Next >** twice to save the connection string in the application's `Web.config` file and move to the **Choose a Command Type** screen.

In the wizard's **Choose a Command Type** screen, select **Use SQL statements** and click **Next >**. The next screen allows you to enter a `SELECT` statement for retrieving data from the database, which will then be placed in an `Authors DataTable` within the `BooksDataSet`. Enter the SQL statement

```
SELECT AuthorID, FirstName + ' ' + LastName AS Name FROM Authors
```

in the text box on the **Enter a SQL Statement** screen. This query selects the `AuthorID` of each row. This query's result will also contain the column `Name` that is created by concatenating each row's `FirstName` and `LastName`, separated by a space. The **AS** SQL keyword allows you to generate a column in a query result—called an **alias**—that contains a SQL expression's result (e.g., `FirstName + ' ' + LastName`). You'll soon see how we use this query's result to populate the `DropDownList` with items containing the authors' full names.

After entering the SQL statement, click the **Advanced Options...** button and uncheck **Generate Insert, Update and Delete statements**, since this application does not need to modify the database's contents. Click **OK** to close the **Advanced Options** dialog. Click **Next >** to advance to the **Choose Methods to Generate** screen. Leave the default names and click **Finish**. Notice that the **DataSet Designer** (Fig. 25.58) now displays a `DataTable` named `Authors` with `AuthorID` and `Name` members, and `Fill` and `GetData` methods.

Step 14: Creating and Configuring a `TitlesTableAdapter`

`Books.aspx` needs to access a list of books by a specific author and a list of authors. Thus we must create a `TitlesTableAdapter` that will retrieve the desired information from the

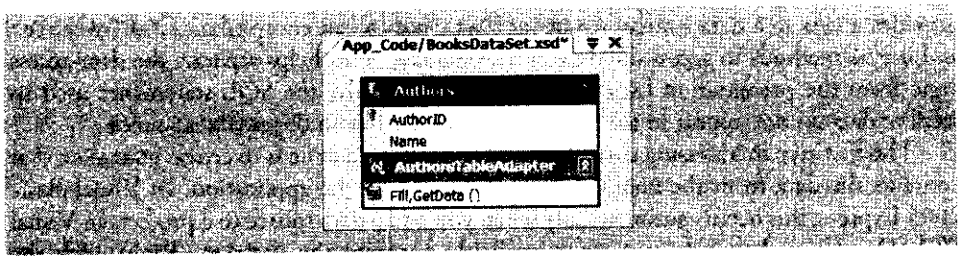


Fig. 25.58 | `Authors DataTable` in the **Dataset Designer**.

database's `Titles` table. Right click the **Dataset Designer** and from the menu that appears, select **Add > TableAdapter...** to launch the **TableAdapter Configuration Wizard**. Make sure the `BooksConnectionString` is selected as the connection in the wizard's first screen, then click **Next >**. Choose **Use SQL statements** and click **Next >**.

In the **Enter a SQL Statement** screen, open the **Advanced Options** dialog and uncheck **Generate Insert, Update and Delete statements**, then click **OK**. Our application allows users to filter the books displayed by the author's name, so we need to build a query that takes an `AuthorID` as a parameter and returns the rows in the `Titles` table for books written by that author. To build this complex query, click the **Query Builder...** button.

In the **Add Table** dialog that appears, select **AuthorISBN** and click **Add**. Then **Add** the **Titles** table, too. Our query requires access to data in both of these tables. Click **Close** to exit the **Add Table** dialog. In the **Query Builder** window's top pane (Fig. 25.59), check the box marked *** (All Columns)** in the **Titles** table. Next, in the middle pane, add a row with **Column** set to `AuthorISBN.AuthorID`. Uncheck the **Output** box, because we do not want

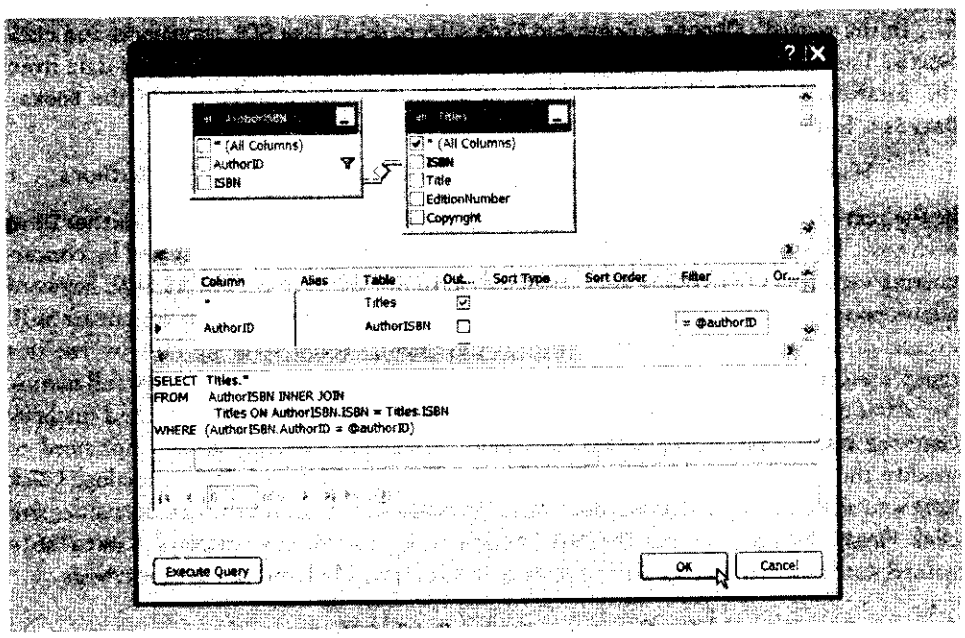


Fig. 25.59 | **Query Builder** for designing a query that selects books written by a particular author.

the **AuthorID** to appear in our query result. Add an `@authorID` parameter in this row's **Filter** column. The SQL statement generated by these actions retrieves information about all books written by the author specified by parameter `@authorID`. The statement first merges the data from the `AuthorISBN` and `Titles` tables. The `INNER JOIN` clause specifies that the `ISBN` columns of each table are compared to determine which rows are merged. The `INNER JOIN` results in a temporary table containing the columns of both tables. The `WHERE` clause of the SQL statement restricts the book information from this temporary table to a specific author (i.e., all rows in which the `AuthorID` column is equal to `@authorID`).

Click **OK** to exit the **Query Builder**, then in the **TableAdapter Configuration Wizard**, click **Next >**. On the **Choose Methods to Generate** screen, enter `FillByAuthorID` and `GetDataByAuthorID` as the names of the two methods to be generated for the `TitlesTableAdapter`. Click **Finish** to exit the wizard. You should now see a `Titles DataTable` in the **Dataset Designer** (Fig. 25.60).

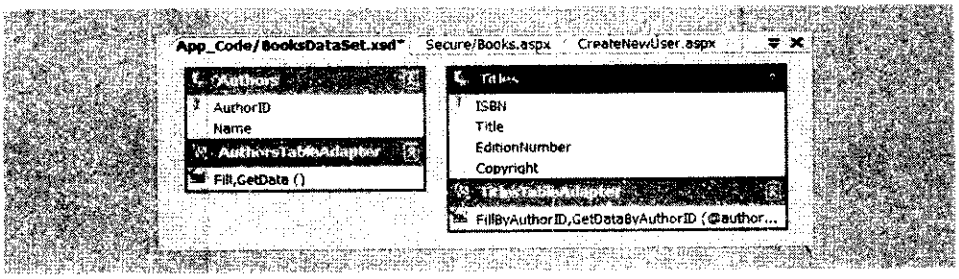


Fig. 25.60 | **Dataset Designer** after adding the `TitlesTableAdapter`.

Step 15: Adding a `DropDownList` *Containing Authors' First and Last Names*

Now that we have created a `BooksDataSet` and configured the necessary `TableAdapters`, we add controls to `Books.aspx` that will display the data on the web page. We first add the `DropDownList` from which users can select an author. Open `Books.aspx` in **Design** mode, then add the text `Author:` and a `DropDownList` control named `authorsDropDownList` in the page's `Content` control, below the existing content. The `DropDownList` initially displays the text `[Unbound]`. We now bind the list to a data source, so the list displays the author information placed in the `BooksDataSet` by the `AuthorsTableAdapter`. In the **DropDownList Tasks** smart tag menu, click **Choose Data Source...** to start the **Data Source Configuration Wizard**. Select **<New data source...>** from the **Select a data source** drop-down list in the first screen of the wizard. Doing so opens the **Choose a Data Source Type** screen. Select **Object** and set the ID to `authorsObjectDataSource`, then click **OK**.

An `ObjectDataSource` accesses data through another object, often called a **business object**. Recall that the middle tier of a three-tier application contains business logic that controls the way an application's top-tier user interface (in this case, `Books.aspx`) accesses the bottom tier's data (in this case, the `Books.mdf` database file). Thus, a business object represents the middle tier of an application and mediates interactions between the other two tiers. In an ASP.NET web application, a `TableAdapter` typically serves as the business object that retrieves the data from the bottom-tier database and makes it available to the top-tier user interface through a `DataSet`. In the **Choose a Business Object** screen of the **Configure Data Source** wizard (Fig. 25.61), selec `BooksDataSetTableAdapters.AuthorsTableAdapter`. [*Note:* You may need to save the project to see the `AuthorsTableAdapter`.]

`BooksDataSetTableAdapters` is a namespace declared by the IDE when you create `BooksDataSet`. Click **Next >** to continue.

The **Define Data Methods** screen (Fig. 25.62) allows you to specify which of the business object's methods (in this case, `AuthorsTableAdapter`) should be used to obtain the data accessed through the `ObjectDataSource`. You can choose only methods that return data, so the only choice is method `GetData`, which returns an `AuthorsDataTable`. Click **Finish** to close the **Configure Data Source** wizard and return to the **Data Source Configuration Wizard** for the `DropDownList` (Fig. 25.63). The new data source (i.e., `authors`

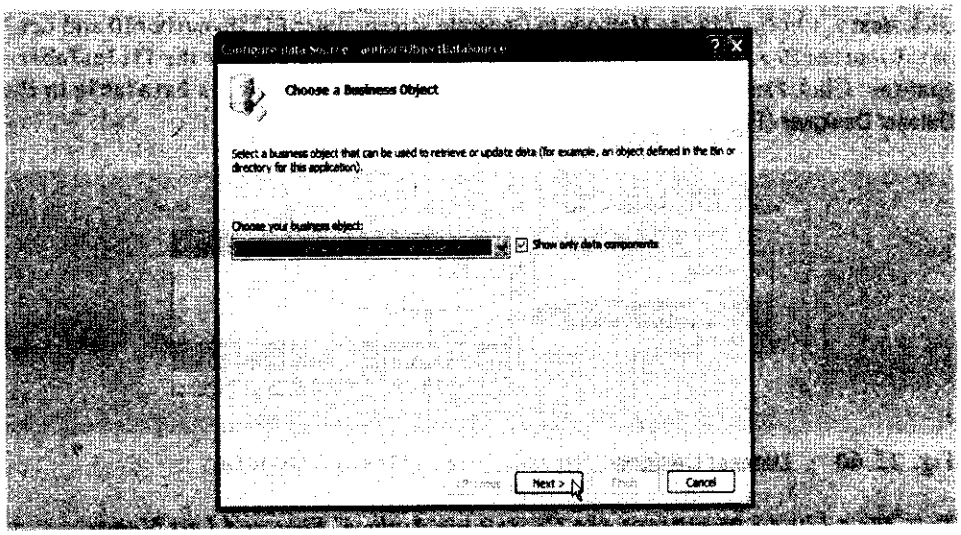


Fig. 25.61 | Choosing a business object for an `ObjectDataSource`.

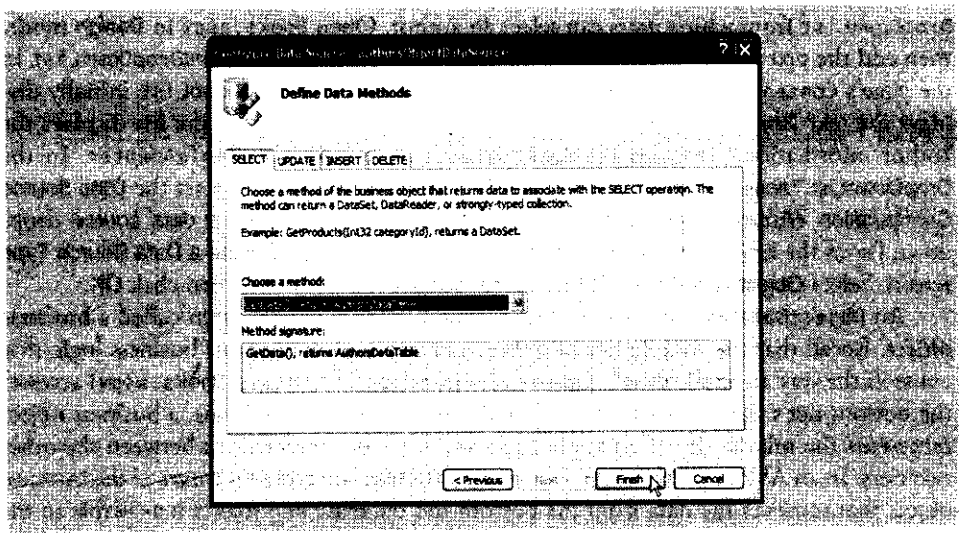


Fig. 25.62 | Choosing a data method of a business object for use with an `ObjectDataSource`.

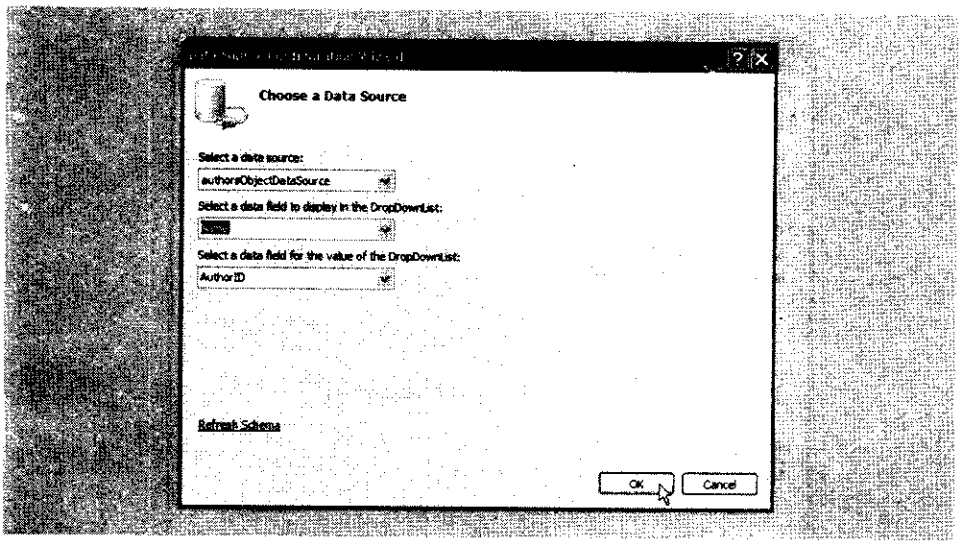


Fig. 25.63 | Choosing a data source for a `DropDownList`.

`ObjectDataSource`) should be selected in the top drop-down list. The other two drop-down lists on this screen allow you to configure how the `DropDownList` control uses the data from the data source. Set `Name` as the data field to display and `AuthorID` as the data field to use as the value. Thus, when `authorsDropDownList` is rendered in a web browser, the list items display the author names, but the underlying values associated with each item are the author `AuthorID`s. Finally, click **OK** to bind the `DropDownList` to the specified data.

The last step in configuring the `DropDownList` on `Books.aspx` is to set the control's **`AutoPostBack`** property to `True`. This property indicates that a postback occurs each time the user selects an item in the `DropDownList`. As you will see shortly, this causes the page's `GridView` (created in the next step) to display new data.

Step 16: Creating a `GridView` to Display the Selected Author's Books

We now add a `GridView` to `Books.aspx` for displaying the book information by the author selected in the `authorsDropDownList`. Add a `GridView` named `titlesGridView` below the other controls in the page's `Content` control.

To bind the `GridView` to data from the `Books` database, select **<New data source...>** from the **Choose Data Source** drop-down list in the **GridView Tasks** smart tag menu. When the **Data Source Configuration Wizard** opens, select **Object** and set the `ID` of the data source to `titlesObjectDataSource`, then click **OK**. In the **Choose a Business Object** screen, select the `BooksDataSetTableAdapters.TitlesTableAdapter` from the drop-down list to indicate the object that will be used to access the data. Click **Next >**. In the **Define Data Methods** screen, leave the default selection of `GetDataByAuthorID` as the method that will be invoked to obtain the data for display in the `GridView`. Click **Next >**.

Recall that `TitlesTableAdapter` method `GetDataByAuthorID` requires a parameter to indicate the `AuthorID` for which data should be retrieved. The **Define Parameters** screen (Fig. 25.64) allows you to specify where to obtain the value of the `@authorID` parameter in the SQL statement executed by `GetDataByAuthorID`. Select **Control** from the **Parameter source** drop-down list. Select `authorsDropDownList` as the **ControlID** (i.e., the `ID` of the

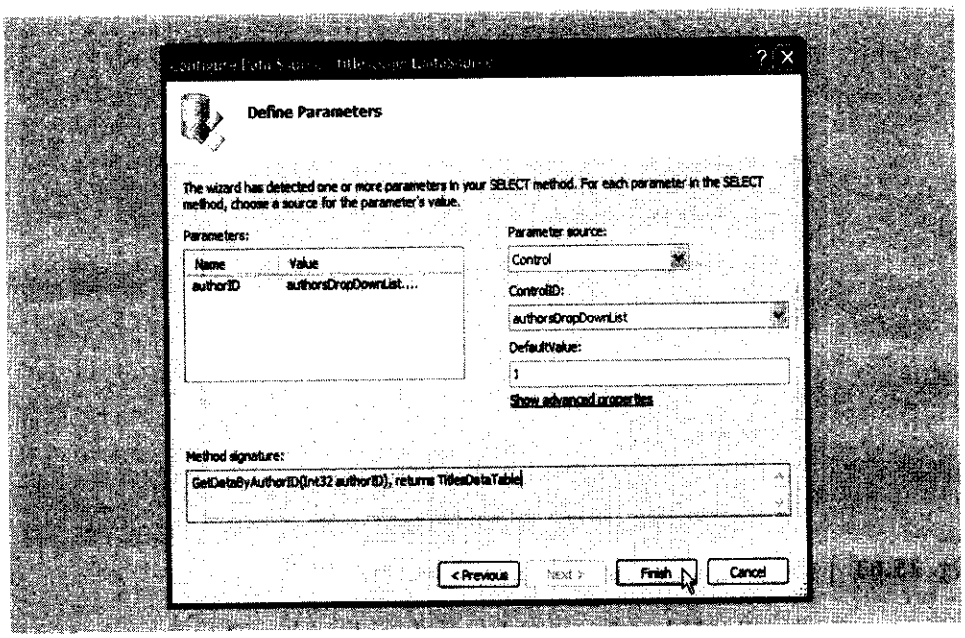


Fig. 25.64 | Choosing the data source for a parameter in a business object's data method.

parameter source control). Next, enter `1` as the **DefaultValue**, so books by Harvey Deitel (who has `AuthorID` `1` in the database) display when the page first loads (i.e., before the user has made any selections using the `authorsDropDownList`). Finally, click **Finish** to exit the wizard. The `GridView` is now configured to display the data retrieved by `TitlesTableAdapter.GetDataByAuthorID`, using the value of the current selection in `authorsDropDownList` as the parameter. Thus, when the user selects a new author and a postback occurs, the `GridView` displays a new set of data.

Now that the `GridView` is tied to a data source, we modify several of the control's properties to adjust its appearance and behavior. Set the `GridView`'s `CellPadding` property to `5`, set the `BackColor` of the `AlternatingRowStyle` to `LightYellow`, and set the `BackColor` of the `HeaderStyle` to `LightGreen`. Change the `Width` of the control to `600px` to accommodate long data values.

Next, in the **GridView Tasks** smart tag menu, check **Enable Sorting**. This causes the column headings in the `GridView` to turn into hyperlinks that allow users to sort the data in the `GridView`. For example, clicking the `Titles` heading in the web browser will cause the displayed data to appear sorted in alphabetical order. Clicking this heading a second time will cause the data to be sorted in reverse alphabetical order. ASP.NET hides the details required to achieve this functionality.

Finally, in the **GridView Tasks** smart tag menu, check **Enable Paging**. This causes the `GridView` to split across multiple pages. The user can click the numbered links at the bottom of the `GridView` control to display a different page of data. `GridView`'s **PageSize** property determines the number of entries per page. Set the `PageSize` property to `4` using the **Properties** window so that the `GridView` displays only four books per page. This technique for displaying data makes the site more readable and enables pages to load more quickly (because less data is displayed at one time). Note that, as with sorting data in a `GridView`, you do not

need to add any code to achieve paging functionality. Figure 25.65 displays the completed `Books.aspx` file in **Design** mode.

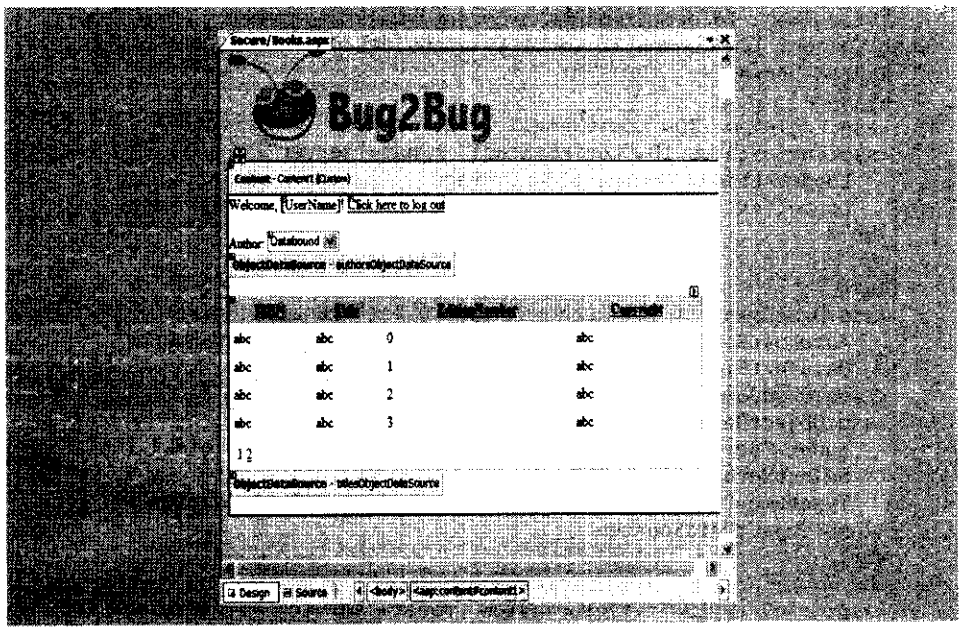


Fig. 25.65 | Completed `Books.aspx` in **Design** mode.

Step 17: Examining the Markup in Books.aspx

Figure 25.66 presents the markup in `Books.aspx` (reformatted for readability). Aside from the exclamation point in line 8, which we added manually in **Source** mode, all the remaining markup was generated by the IDE in response to the actions we performed in **Design** mode. The `Content` control (lines 5–53) defines page-specific content that will replace the `ContentPlaceHolder` named `bodyContent`. Recall that this control is located in the master page specified in line 3. Line 8 creates the `LoginName` control, which displays the authenticated user's name when the page is requested and viewed in a browser. Lines 9–11 create the `LoginStatus` control. Recall that this control is configured to redirect the user to the login page after logging out (i.e., clicking the hyperlink with the `LogoutText`).

Lines 15–18 define the `DropDownList` that displays the names of the authors in the `Books` database. Line 16 contains the control's `AutoPostBack` property, which indicates that changing the selected item in the list causes a postback to occur. The `DataSourceID` property in line 16 specifies that the `DropDownList`'s items are created based on the data obtained through the `authorsObjectDataSource` (defined in lines 19–23). Line 21 specifies that this `ObjectDataSource` accesses the `Books` database by calling method `GetData` of the `BooksDataSet`'s `AuthorsTableAdapter` (line 22).

Lines 26–42 create the `GridView` that displays information about the books written by the selected author. The start tag (lines 26–29) indicates that paging (with a page size of 4) and sorting are enabled in the `GridView`. The `AutoGenerateColumns` property indicates whether the columns in the `GridView` are generated at runtime based on the fields in the data source. This property is set to `False`, because the IDE-generated `Columns` element

```
1  <%-- Fig. 25.66: Books.aspx --%>
2  <%-- Displays information from the Books database. --%>
3  <%@ Page Language="VB" MasterPageFile="~/Bug2Bug.master"
4     Title="Book Information" %>
5  <asp:Content ID="Content1" ContentPlaceHolderID="bodyContent"
6     Runat="Server">
7     Welcome,
8     <asp:LoginName ID="LoginName1" runat="server" />!
9     <asp:LoginStatus ID="LoginStatus1" runat="server"
10       LogoutAction="RedirectToLoginPage"
11       LogoutText="Click here to log out" />
12    <br />
13    <br />
14    Author:
15    <asp:DropDownList ID="authorsDropDownList" runat="server"
16       AutoPostBack="True" DataSourceID="authorsObjectDataSource"
17       DataTextField="Name" DataValueField="AuthorID">
18    </asp:DropDownList>
19    <asp:ObjectDataSource ID="authorsObjectDataSource"
20       runat="server" OldValuesParameterFormatString="original_{0}"
21       SelectMethod="GetData"
22       TypeName="BooksDataSetTableAdapters.AuthorsTableAdapter">
23    </asp:ObjectDataSource>
24    <br />
25    <br />
26    <asp:GridView ID="titlesGridView" runat="server" AllowPaging="True"
27       AllowSorting="True" AutoGenerateColumns="False" CellPadding="5"
28       DataKeyNames="ISBN" DataSourceID="titlesObjectDataSource"
29       PageSize="4" Width="600px">
30       <Columns>
31          <asp:BoundField DataField="ISBN" HeaderText="ISBN"
32             ReadOnly="True" SortExpression="ISBN" />
33          <asp:BoundField DataField="Title" HeaderText="Title"
34             SortExpression="Title" />
35          <asp:BoundField DataField="EditionNumber"
36             HeaderText="EditionNumber" SortExpression="EditionNumber" />
37          <asp:BoundField DataField="Copyright" HeaderText="Copyright"
38             SortExpression="Copyright" />
39       </Columns>
40       <HeaderStyle BackColor="LightGray" />
41       <AlternatingRowStyle BackColor="LightGray" />
42    </asp:GridView>
43    <asp:ObjectDataSource ID="titlesObjectDataSource" runat="server"
44       OldValuesParameterFormatString="original_{0}"
45       SelectMethod="GetDataByAuthorID"
46       TypeName="BooksDataSetTableAdapters.TitlesTableAdapter">
47       <SelectParameters>
48          <asp:ControlParameter ControlID="authorsDropDownList"
49             DefaultValue="1" Name="authorID"
50             PropertyName="SelectedValue" Type="Int32" />
51       </SelectParameters>
52    </asp:ObjectDataSource>
53 </asp:Content>
```

Fig. 25.66 | Markup for the completed `Books.aspx` file. (Part 1 of 2.)

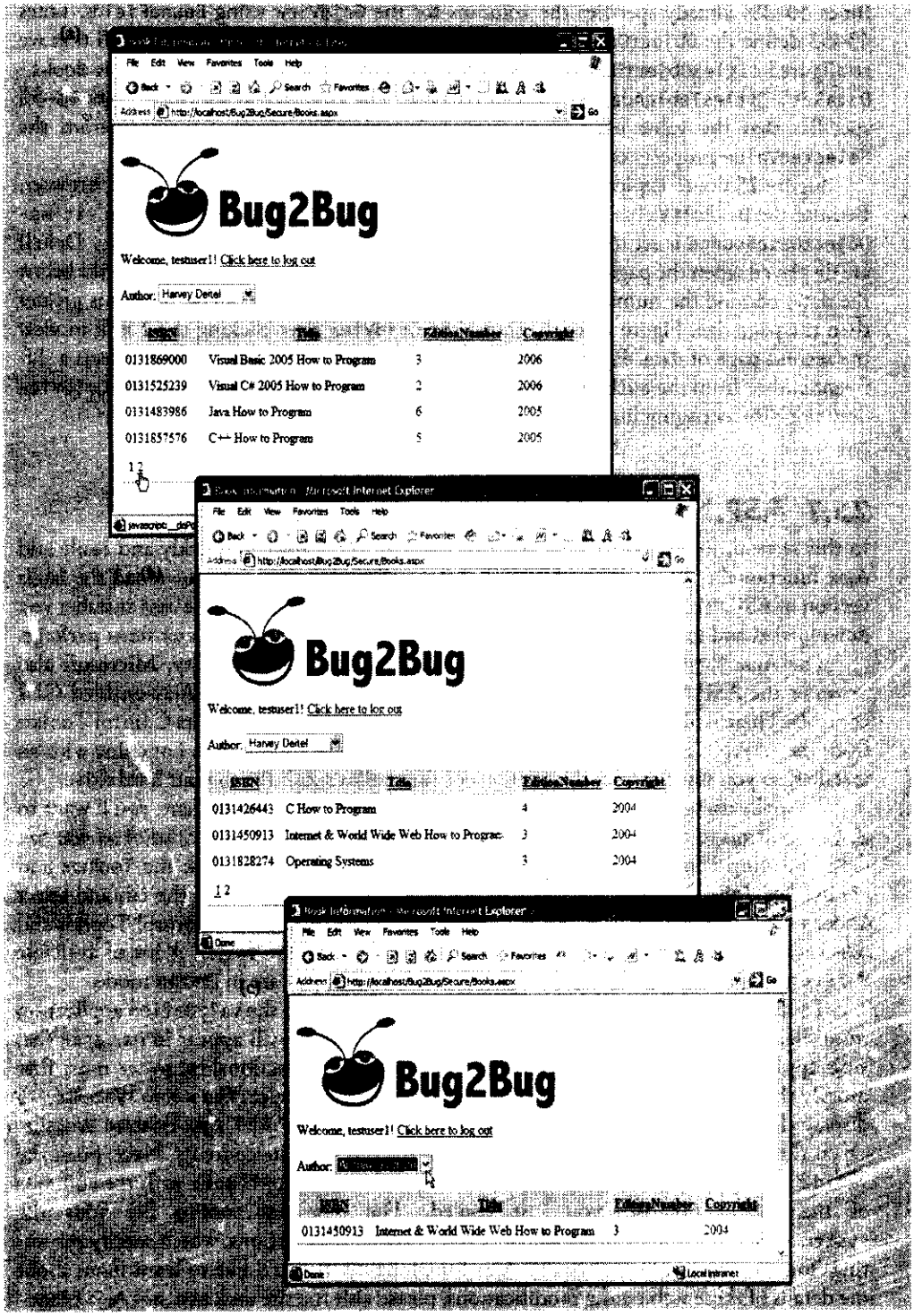


Fig. 25.66 | Markup for the completed `Books.aspx` file. (Part 2 of 2.)

(lines 30–39) already specifies the columns for the `GridView` using `BoundFields`. Lines 43–52 define the `ObjectDataSource` used to fill the `GridView` with data. Recall that we configured `titlesObjectDataSource` to use method `GetDataByAuthorID` of the `Books-DataSet`'s `TitlesTableAdapter` for this purpose. The `ControlParameter` in lines 48–50 specifies that the value of method `GetDataByAuthorID`'s parameter comes from the `SelectedValue` property of the `authorsDropDownList`.

Figure 25.66(a) depicts the default appearance of `Books.aspx` in a web browser. Because the `DefaultValue` property (line 49) of the `ControlParameter` for the `titles-ObjectDataSource` is set to `1`, books by the author with `AuthorID` `1` (i.e., Harvey Deitel) are displayed when the page first loads. Note that the `GridView` displays paging links below the data, because the number of rows of data returned by `GetDataByAuthorID` is greater than the page size. Figure 25.66(b) shows the `GridView` after clicking the `2` link to view the second page of data. Figure 25.66(c) presents `Books.aspx` after the user selects a different author from the `authorsDropDownList`. The data fits on one page, so the `GridView` does not display paging links.

25.7 ASP.NET Ajax

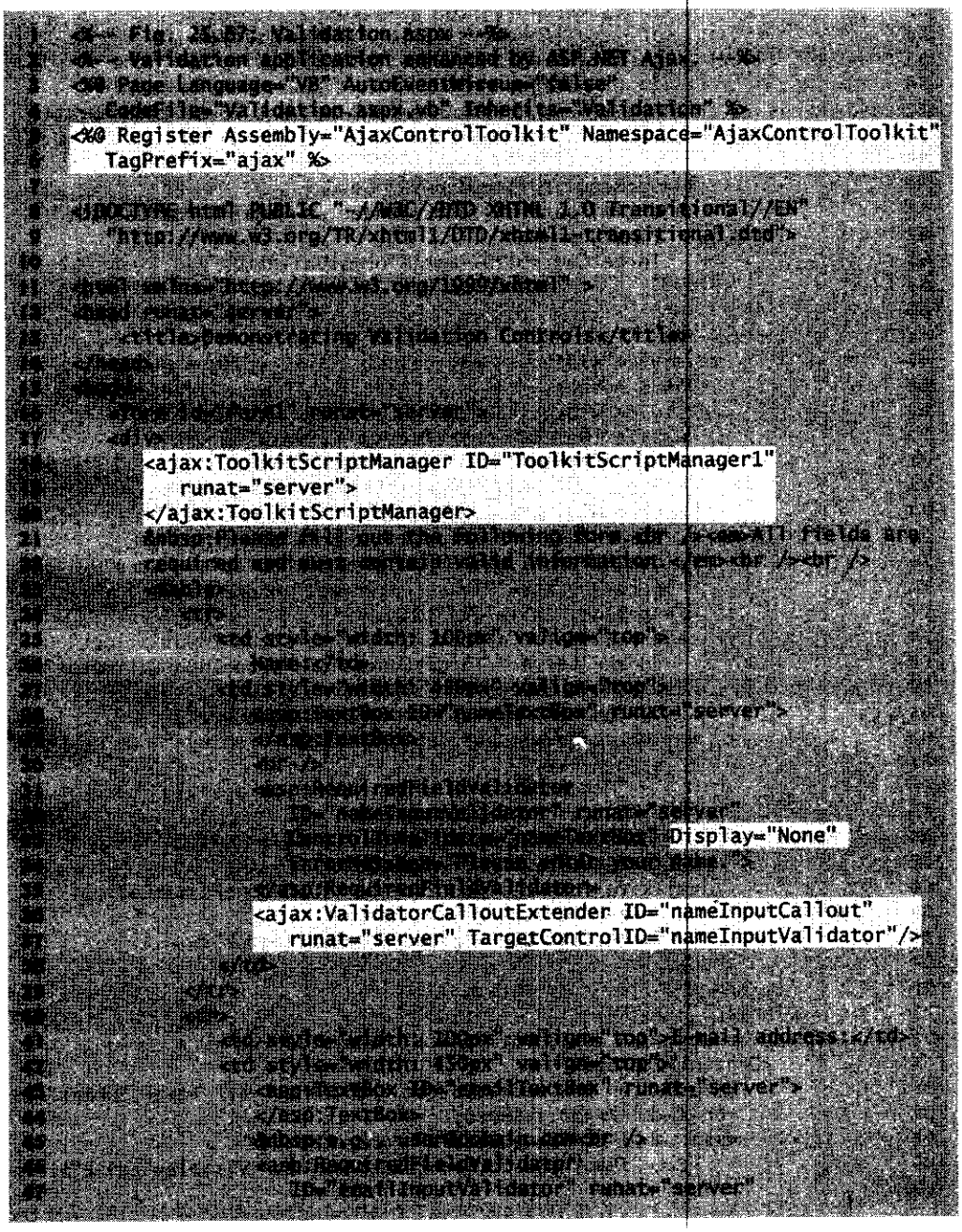
In this section, we introduce how you can use **ASP.NET Ajax** to quickly and easily add Ajax functionality to existing ASP.NET web applications. You can download the latest version of ASP.NET Ajax from `www.asp.net/ajax/downloads`. Run the `.msi` installer you downloaded and follow the on-screen instructions to install the **Ajax Extensions package**.

The Ajax Extensions package implements basic Ajax functionality. Microsoft also provides the **ASP.NET Ajax Control Toolkit**, which contains rich, Ajax-enabled GUI controls. There is also a link to the download the latest version of the Ajax Control Toolkit from the ASP.NET Ajax download page listed above. The toolkit does not come with an installer, so you must extract the contents of the toolkit's ZIP file to your hard drive.

To make using the ASP.NET Ajax Control Toolkit more convenient, you'll want to add its controls to the **Toolbox** in Visual Web Developer (or in Visual Studio) so you can drag and drop controls onto your Web Forms. To do so, right click the **Toolbox** and choose **Add Tab**. Type `Ajax Toolkit` in the new tab. Then right click the tab and select **Choose Items**. Navigate to the folder in which you extracted the Ajax Control Toolkit and select `AjaxControlToolkit.dll` from the `SampleWebSite\Bin` folder. A list of available Ajax controls will appear under the **Ajax Toolkit** tab when you are in **Design** mode.

To demonstrate ASP.NET Ajax capabilities we'll enhance the `Validation` application from Fig. 25.17. The only modifications to this application will appear in its `.aspx` file. This application was not initially set up to support Ajax functionality, so we must first modify the `web.config` file. First, in Visual Web Developer select **File > New Website...** to display the **New Website** dialog. Then, create an empty **ASP.NET Ajax-Enabled Website**. Open the `web.config` file in this new application and copy its contents. Next, open the `Validation` application and replace the contents of its `web.config` file with the contents of the `web.config` file you just copied. The new `web.config` file adds the `system.web.extensions`, `httpHandlers` and `httpModules` sections, which specify the settings for running scripts that enable Ajax functionality. If you'd like to learn more about the details of these web.config modifications, please visit the site `www.asp.net/ajax/documentation/live/configuringASPNETAJAX.aspx`.

We'll now use Ajax-enabled controls to add Ajax features to this application. Figure 25.67 is a modified `validation.aspx` file that enhances the application by using the `ToolkitScriptManager`, `UpdatePanel` and `ValidatorCalloutExtender` controls.



```
<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
   TagPrefix="ajax" %>

<ajax:ToolkitScriptManager ID="ToolkitScriptManager1"
   runat="server">
</ajax:ToolkitScriptManager>

Display="None"

<ajax:ValidatorCalloutExtender ID="nameInputCallout"
   runat="server" TargetControlID="nameInputValidator" />
```

Fig. 25.67 | `Validation` application enhanced by ASP.NET Ajax. (Part 1 of 3.)

```
                ControlToValidate="emailTextBox" Display="None"
                ErrorMessage="Please enter your e-mail address.">
             </asp:RequiredFieldValidator>
             <ajax:ValidatorCalloutExtender ID="emailInputCallout"
                runat="server" TargetControlID="emailInputValidator"/>
             <asp:RegularExpressionValidator
                ID="emailFormatValidator" runat="server"
                ControlToValidate="emailTextBox" Display="None"
                ErrorMessage=
                   "Please enter an e-mail address in a valid format."
                ValidationExpression=
                   "\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
             </asp:RegularExpressionValidator>
             <ajax:ValidatorCalloutExtender ID="emailFormatCallout"
                runat="server"
                TargetControlID="emailFormatValidator"/>
          </td>
       </tr>
       <tr>
          <td style="width: 100px; height: 22px;" valign="top">
          Phone number:</td>
          <td style="width: 450px; height: 22px;" valign="top">
             <asp:TextBox ID="phoneTextBox" runat="server">
             </asp:TextBox>
              e.g., (555) 555-1234<br />
             <asp:RequiredFieldValidator
                ID="phoneInputValidator" runat="server"
                ControlToValidate="phoneTextBox" Display="None"
                ErrorMessage="Please enter your phone number.">
             </asp:RequiredFieldValidator>
             <ajax:ValidatorCalloutExtender ID="phoneInputCallout"
                runat="server" TargetControlID="phoneInputValidator"/>
             <asp:RegularExpressionValidator
                ID="phoneFormatValidator" runat="server"
                ControlToValidate="phoneTextBox" Display="None"
                ErrorMessage=
                   "Please enter a phone number in a valid format."
                ValidationExpression=
                   "\(\d{3}\) \d{3}-\d{4}">
             </asp:RegularExpressionValidator>
             <ajax:ValidatorCalloutExtender ID="PhoneFormatCallout"
                runat="server"
                TargetControlID="phoneFormatValidator"/>
          </td>
       </tr>
    </table>
    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
       <ContentTemplate>
          <asp:Button ID="submitButton" runat="server" Text="Submit" />
          <br /><br /><br /> 
          <asp:Label ID="outputLabel" runat="server"
             Text="Thank you for your submission." Visible="False">
          </asp:Label>
```

Fig. 25.67 | Validation application enhanced by ASP.NET Ajax. (Part 2 of 3.)

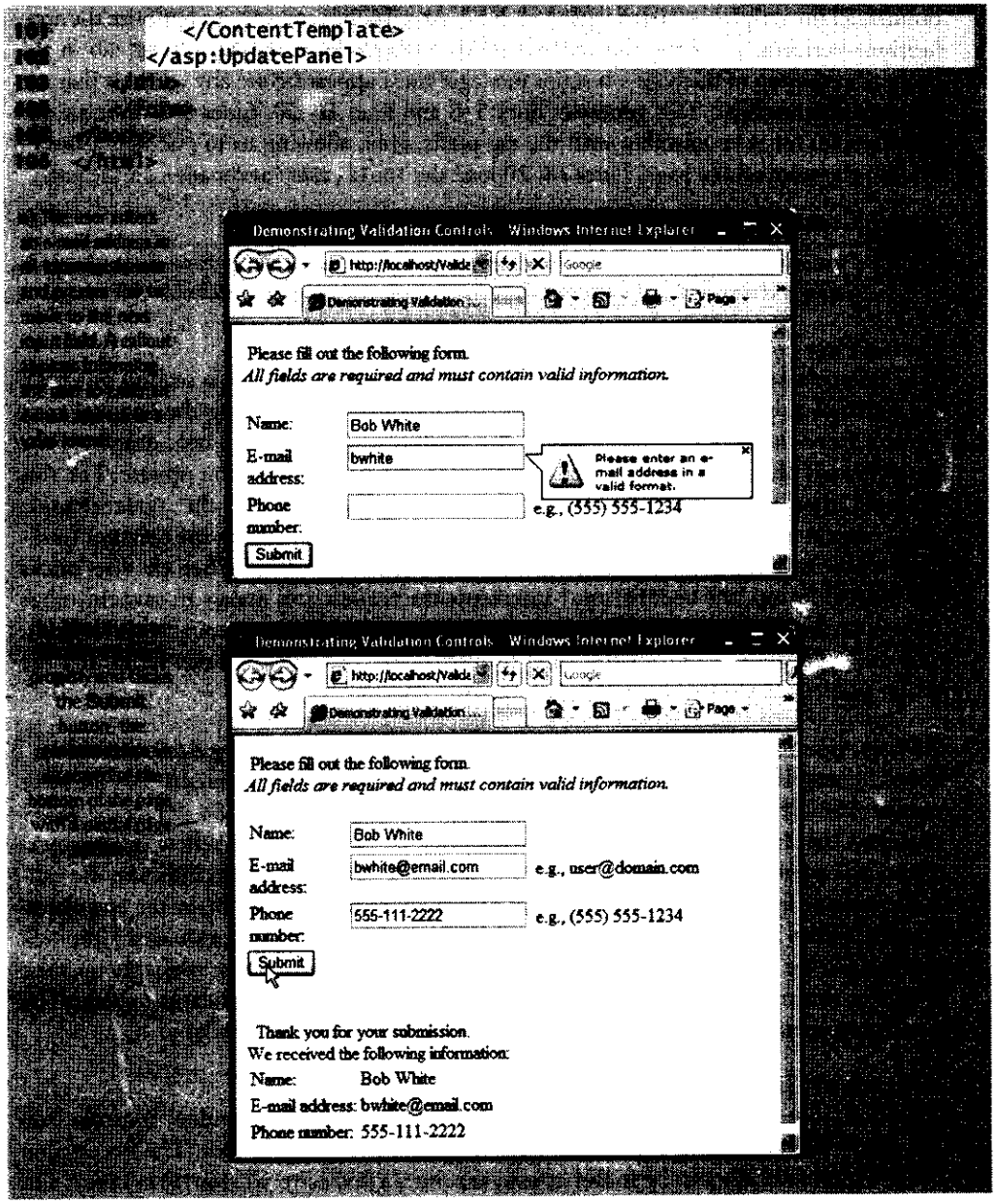


Fig. 25.67 | `Validation` application enhanced by ASP.NET Ajax. (Part 3 of 3.)

`ScriptManager` Control

The key control in every ASP.NET Ajax-enabled application is the **`ScriptManager`**, which manages the client-side scripts that enable asynchronous Ajax functionality. There can be only one `ScriptManager` per page. To incorporate controls from the Ajax Control Toolkit you should use the **`ToolkitScriptManager`** that comes with the toolkit contorls, rather than the `ScriptManager` from the ASP.NET Ajax Extensions. The `ToolkitScriptManager`

bundles all the scripts associated with ASP.NET Ajax Toolkit controls to optimize the application's performance. Drag the `ToolkitScriptManager` from the **Ajax Toolkit** tab in the toolbox to the top of the page—a script manager must appear before any controls that use the scripts it manages. This generates lines 5–6 and lines 18–20. Lines 5–6 associate the `AjaxControlToolkit` assembly with the tag prefix `ajax`, allowing us to put Ajax Control Toolkit elements on the page. Lines 18–20 load the `ToolkitScriptManager` on the page.

Common Programming Error 25.1

Putting more than one instance of the `ScriptManager` control on a Web Form causes the application to throw an `InvalidOperationException` when the page is initialized.

Partial Page Updates Using the `UpdatePanel` Control

The **`UpdatePanel` control** eliminates full-page refreshes by isolating a section of a page for a partial-page update. To implement a partial-page update, drag the `UpdatePanel` control from the **Ajax Extensions** tab in the **Toolbox** to your form. Then, drag into the `UpdatePanel` the control to update and the control that triggers the update. For this example, drag the `outputLabel` and `submitButton` elements into the `UpdatePanel`. The components that are managed by the `UpdatePanel` are placed in the `ContentTemplate` element (lines 95–101) of the `UpdatePanel` (lines 94–102). When the user clicks the **Submit** button, the `UpdatePanel` intercepts the request and makes an asynchronous request to the server instead. Then the response is inserted in the `outputLabel` element, and the `UpdatePanel` reloads the label to display the new text without refreshing the entire page.

Adding Ajax Functionality to ASP.NET Validation Controls Using Ajax Extenders

Several controls in the Ajax Control Toolkit are **extenders**—components that enhance regular ASP.NET controls. Lines 36–37, 51–52, 61–63, 78–79 and 88–90 define **`ValidatorCalloutExtender` controls** that display error messages in small yellow callouts next to the input fields. Line 37 sets the `targetControlID` property, which indicates the validator control from which the `ValidatorCalloutExtender` should obtain the error message to display. The `ValidatorCalloutExtender`s display error messages with a nicer look and feel, so we no longer need the validator controls to display these messages on their own. For this reason, line 33 sets the `Display` property of the first validator to `None`. The remaining control extenders and validator controls are configured similarly.

Additional ASP.NET Information

The Ajax Control Toolkit contains many other extenders and independent controls. You can check them out using the sample website included with the toolkit. The live version of the sample website can be found at `www.asp.net/ajax/control-toolkit/live/`. For more information on ASP.NET Ajax, check out our ASP.NET Ajax Resource Center at `www.deitel.com/aspdotnetajax`.

25.8 Web Resources

`www.deitel.com/aspdotnet/`

The Deitel ASP.NET Resource Center focuses on the vast amount of free ASP.NET content available online, plus some for-sale items. Start your search here for tools, downloads, text and video tutorials, webcasts, podcasts, wikis, documentation, reference manuals, conferences, FAQs,

books, e-books, sample chapters, articles, newsgroups, forums, downloads from CNET's download.com, jobs and contract opportunities, and more that will help you develop ASP.NET-based applications. Keep track of ASP.NET blogs for the latest news and developments, or sign up for RSS feeds to be notified promptly of each new development. Also, download free open-source ASP.NET projects.

Summary

Section 25.1 Introduction

- Microsoft's ASP.NET technology is used for web application development.
- Web-based applications create web content for web-browser clients. This web content includes XHTML, client-side scripting, images and binary data.
- A Web Form file generates a web page that is sent to the client browser. Web Form files have the filename extension .aspx and contain a web page's GUI. You customize Web Forms by adding web controls.
- Every ASPX file created in Visual Studio has a corresponding class written in a .NET language. The file that contains this class is called the code-behind file and provides the ASPX file's programmatic implementation.

Section 25.2 Creating and Running a Simple Web Form Example

- An ASP.NET Web Form typically consists of an ASPX file and a code-behind file.
- Visual Web Developer generates markup when you change a Web Form's properties and when you add text or controls to a Web Form.

Section 25.2.1 Examining an ASPX File

- ASP.NET comments begin with `<%--` and terminate with `--%>`.
- A `Page` directive (delimited by `<%@` and `%>`) specifies information needed by ASP.NET to process an ASPX file. The `CodeFile` attribute of the `Page` directive indicates the name of the corresponding code-behind file. The `Language` attribute specifies the .NET language used in this file.
- When a control's `runat` attribute is set to `"server"`, the control is processed by ASP.NET on the server, generating an XHTML equivalent.
- The `asp:` tag prefix in a control declaration indicates that a control is an ASP.NET web control.
- Each web control maps to a corresponding XHTML element (or group of elements)—when processing a web control on the server, ASP.NET generates XHTML markup that will be sent to the client to represent that control in a web browser.

Section 25.2.2 Examining a Code-Behind File

- The code-behind file is a partial class.
- Namespace `System.Web.UI` contains classes for the creation of web applications and controls.
- Class `Page` defines a standard web page, providing events and objects necessary for creating Web-based applications. All web pages directly or indirectly inherit from class `Page`.
- Class `Control` is the base class that provides common functionality for all web controls.
- Method `Page_Init` handles the `Init` event, which indicates that a page is initialized and ready to execute application-specific initialization code.

Section 25.2.3 Relationship Between an ASPX File and a Code-Behind File

- When a client requests an ASPX file, ASP.NET combines two partial classes—the one defined in the code-behind file and the one that ASP.NET generates based on the markup in the ASPX file that defines the page's GUI.
- ASP.NET compiles the combined partial classes and creates a class that represents the page. An instance of this class creates the XHTML that is sent to the client.
- Namespace `System.Web.UI.WebControls` contains web controls (derived from class `WebControl`) for designing a page's user interface.

Section 25.2.4 How the Code in an ASP.NET Web Page Executes

- When an instance of a page is created, the `PreInit` event occurs first, invoking method `Page_PreInit`. The `Init` event occurs next, invoking method `Page_Init`. Then the `Load` event occurs, invoking method `Page_Load`.
- After `Page_Load` finishes executing, the page processes any events raised by the page's controls.
- When a Web Form object completes the response to the user, an `Unload` event occurs. Event handler `Page_Unload` is inherited from class `Page` and contains any code that releases resources.

Section 25.2.5 Examining the XHTML Generated by an ASP.NET Application

- A form is a mechanism for collecting user information and sending it to the web server.
- XHTML forms can contain visual and nonvisual components.
- Nonvisual components in an XHTML form, called hidden inputs, store any data that the document author specifies.

Section 25.2.6 Building an ASP.NET Web Application

- The name `localhost` indicates that the server resides on the local computer. If the web server were located on a different computer, `localhost` would be replaced with the appropriate IP address or hostname.
- **DOCUMENT** is the name used to represent a Web Form in the **Properties** window.
- The Web Forms Designer's **Source** mode allows you to view the markup that represents the user interface of a page. The **Design** mode allows you to view the page as it will look and modify it by dragging and dropping controls from the **Toolbox** onto the Web Form.
- Controls and other elements are placed sequentially on a Web Form, much as text and images are placed in a document using word-processing software like Microsoft Word. The positions of controls and other elements flow based on the width of the page by default.
- An alternate type of layout is known as absolute positioning, in which controls are located exactly where they are dropped on the Web Form.
- Visual Web Developer is a WYSIWYG (What You See Is What You Get) editor—whenever you make a change to a Web Form in **Design** mode, the IDE creates the markup (visible in **Source** mode) necessary to achieve the desired visual effects seen in **Design** mode.
- `Web.config` is a file that stores configuration settings for an ASP.NET web application.

Section 25.3 Web Controls

- The **Standard** section of the **Toolbox** in Visual Web Developer contains several web controls.

Section 25.3.1 Text and Graphics Controls

- The **Insert Table** command from the **Layout** menu in **Design** mode allows you to add an XHTML table to a Web Form.

- An `Image` control inserts an image into a web page. The `ImageUrl` property specifies the file location of the image to display.
- A `TextBox` control allows the you to obtain text from the user and display text to the user.
- A `DropDownList` control provides a list of options to the user. Each item in the drop-down list is defined by a `ListItem` element.
- Visual Web Developer displays smart tag menus for many ASP.NET controls to facilitate performing common tasks. A smart tag menu is opened by clicking the small arrowhead that appears in the upper-right corner of the control in **Design** mode.
- A `HyperLink` control adds a hyperlink to a web page. The `NavigateUrl` property of this control specifies the resource that is requested when a user clicks the hyperlink.
- A `RadioButtonList` control provides a series of radio buttons for the user.

Section 25.3.2 AdRotator Control

- ASP.NET provides the `AdRotator` web control for displaying advertisements (or any other images). Using data from an XML file, the `AdRotator` control randomly selects an image to display and generates a hyperlink to the web page associated with that image.
- An `XmlDataSource` references an XML file containing data that will be used in an ASP.NET application. The **AdRotator Tasks** smart tag menu allows you to create a new `XmlDataSource` that retrieves advertisement data from an XML file.
- The advertisement file used for an `AdRotator` control contains `Ad` elements, each of which provides information about a different advertisement.
- Element `ImageUrl` in an advertisement file specifies the path (location) of the advertisement's image, and element `NavigateUrl` specifies the URL that loads when a user clicks the advertisement.
- The `AlternateText` element contains text that displays in place of the image when the browser cannot locate or render the image for some reason, or to assist the visually impaired.
- Element `Impressions` specifies how often an image appears, relative to the other images.

Section 25.3.3 Validation Controls

- A validation control (or validator) determines whether the data in another web control is in the proper format. Validators provide a mechanism for validating user input on the client.
- When the XHTML for a page is created, a validator is converted into ECMAScript, scripting language that enhances the functionality and appearance of Web pages.
- The `Visible` property of a control indicates whether the control appears in the client's browser.
- A `RequiredFieldValidator` ensures that a control receives user input before a form is submitted.
- A validator's `ControlToValidate` property indicates which control will be validated.
- A validator's `ErrorMessage` property contains text to be displayed if the validation fails.
- A `RegularExpressionValidator` matches a web control's content against a regular expression. The regular expression that validates the input is assigned to property `ValidationExpression`.
- Web programmers using ASP.NET often design their web pages so that the current page reloads when the user submits the form. This event is known as a postback.
- A `Page`'s `IsPostBack` property determines whether the page is being loaded due to a postback.
- When data is posted to the web server, the XHTML form's data is accessible to the web application through properties of the ASP.NET controls.
- The `EnableViewState` attribute determines whether a web control's state persists (i.e., is retained) when a postback occurs.

Section 25.4 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.
- The request/response system on which the web operates is facilitated by HTTP.
- HTTP is a stateless protocol.
- 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. To help the server distinguish among clients, each client must identify itself to the server.
- Tracking individual clients is known as session tracking.

Section 25.4.1 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.
- A cookie object is of type `HttpCookie`. Properties `Name` and `Value` of class `HttpCookie` can be used to retrieve the key and value in a key–value pair (both strings) in a cookie.
- Cookies are sent and received as a collection of type `HttpCookieCollection`. An application on a server can write cookies to a client using the `Response` object's `Cookies` property. Cookies can be accessed programmatically using the `Request` object's `Cookies` property. Cookies can be read by an application only if they were created in the domain in which the application is running.
- 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 includes any cookies the server wants to store on the client computer.
- 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.
- Clients can disable cookies. If they do, they may not be able to use certain web applications.

Section 25.4.2 Session Tracking with `HttpSessionState`

- Session-tracking capabilities are provided by FCL class `HttpSessionState`. Every Web Form includes an `HttpSessionState` object, which is accessible through property `Session` of class `Page`.
- When the web page is requested, an `HttpSessionState` object is created and assigned to the `Page`'s `Session` property. 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. Recall that clients may disable cookies in their web browsers to ensure that their privacy is protected. Such clients will experience difficulty using web applications that depend on `HttpSessionState` objects to maintain state information, unless `HttpSessionState` is configured to use URL rewriting.
- The `Page`'s `Session` property is often referred to as the `Session` object.
- The `Session` object's key–value pairs are often referred to as session items.
- Session items are placed into an `HttpSessionState` object by calling method `Add`.
- `HttpSessionState` objects can store any type of object (not just `string`s) as attribute values. This provides increased flexibility in maintaining client state information.

- Property `SessionID` contains the unique session ID. The first time a client connects to the web server, a unique session ID is created for that client. When the client makes additional requests, the client's session ID is compared with the session IDs stored in the web server's memory to retrieve the `HttpSessionState` object for that client.
- Property `Timeout` specifies the maximum amount of time that an `HttpSessionState` object can be inactive before it is discarded.
- Property `Count` provides the number of session items contained in a `Session` object.
- Indexing the `Session` object with a key name retrieves the corresponding value.
- Property `Keys` of class `HttpSessionState` returns a collection containing all the session's keys.

Section 25.5 Case Study: Connecting to a Database in ASP.NET

- A `GridView` ASP.NET data control displays data on a Web Form in a tabular format.

Section 25.5.1 Building a Web Form That Displays Data from a Database

- A `GridView`'s colors can be set using the **Auto Format...** link in the **GridView Tasks** smart tag menu.
- A SQL Server 2005 Express database used by an ASP.NET website should be located in the project's `App_Data` folder.
- A `SqlDataSource` control allows a web application to interact with a database.
- When a `SqlDataSource` is configured to perform `INSERT` SQL operations against the database table from which it gathers data, you must specify the values to insert either programmatically or through other controls on the Web Form.
- The **Command and Parameter Editor**, accessed by clicking the ellipsis next to a `SqlDataSource`'s `InsertQuery` property, allows you to specify that parameter values come from controls.
- Each column in a `GridView` is represented as a `BoundField`.
- `SqlDataSource` property `ConnectionString` indicates the connection through which the `SqlDataSource` control interacts with the database.
- An ASP.NET expression, delimited by `<%$` and `%>`, can be used to access a connection string stored in an application's `Web.config` configuration file.

Section 25.5.2 Modifying the Code-Behind File for the Guestbook Application

- A `SqlDataSource`'s `InsertParameters` collection contains an item corresponding to each parameter in the `SqlDataSource`'s `INSERT` command.
- `SqlDataSource` method `Insert` executes the control's `INSERT` command against the database.
- `GridView` method `DataBind` refreshes the information displayed in the `GridView`.

Section 25.6.1 Examining the Completed Secure Books Database Application

- Forms authentication is a technique that protects a page so that only users known to the website can access it. Such users are known as the site's members.
- ASP.NET login controls help create secure applications using authentication. These controls are found in the **Login** section of the **Toolbox**.
- When a user's identity is confirmed, the user is said to have been authenticated.
- A master page defines common GUI elements that are inherited by each page in a set of content pages. Just as Visual Basic classes can inherit instance variables and methods from existing classes, content pages inherit elements from master pages—this is known as visual inheritance.

Section 25.6.2 Creating the Secure Books Database Application

- ASP.NET hides the details of authenticating users, displaying appropriate success or error messages and redirecting the user to the correct page based on the authentication results.
- The **Web Site Administration Tool** allows you to configure an application's security settings, add site users and create access rules that determine who is allowed to access the site.
- By default, anonymous users who attempt to load a page in a directory to which they are denied access are redirected to a page named `Login.aspx` so that they can identify themselves.
- In an ASP.NET application, a page's configuration settings are determined by the current directory's `Web.config` file. The settings in this file take precedence over the settings in the root directory's `Web.config` file.
- A master page contains placeholders for custom content created in a content page, which visually inherits the master page's content, then adds content in place of the placeholders.
- Master pages have the filename extension `.master` and, like Web Forms, can optionally use a code-behind file to define additional functionality.
- A `Master` directive in an ASPX file specifies that the file defines a master page.
- A `ContentPlaceHolder` control serves as a placeholder for page-specific content defined by a content page using a `Content` control. The `Content` control will appear in place of the master page's `ContentPlaceHolder` when the content page is requested.
- A `CreateUserWizard` control provides a registration form that site visitors can use to create a user account. ASP.NET handles the details of creating a SQL Server database to store the user names, passwords and other account information of the application's users.
- A `Login` control encapsulates the details of logging a user into a web application (i.e., authenticating a user by comparing the provided user name and password with those of an account in the ASP.NET-created membership database). If the user is authenticated, the browser is redirected to the page specified by the `Login` control's `DestinationPageUrl` property. If the user is not authenticated, the `Login` control displays an error message.
- ASP.NET writes to the client an encrypted cookie containing data about an authenticated user.
- Encrypted data is data translated into a code that only the sender and receiver can understand.
- A `LoginName` control displays the current authenticated user name on a Web Form.
- A `LoginStatus` control renders on a web page in one of two ways—by default, if the user is not authenticated (the **Logged Out** view), the control displays a hyperlink with the text `Login`; if the user is authenticated (the **Logged In** view), the control displays a hyperlink with the text `Logout`. The `LogoutText` determines the text of the link in the **Logged In** view.
- An `ObjectDataSource` control encapsulates a business object that provides access to a data source. A business object (e.g., a `TableAdapter`) represents the middle tier of an application and mediates interactions between the bottom tier and the top tier.
- The `AS` SQL keyword allows you to generate a column in a query result—called an alias—that contains the result of a SQL expression.
- A `DropDownList`'s `AutoPostBack` property indicates whether a postback occurs each time the user selects an item.
- When you **Enable Sorting** for a `GridView`, the column headings in the `GridView` turn into hyperlinks that allow users to sort the data it displays.
- When you **Enable Paging** for a `GridView`, the `GridView` divides its data among multiple pages. The user can click the numbered links at the bottom of the `GridView` control to display a different page of data. `GridView`'s `PageSize` property determines the number of entries per page.

Section 25.7 ASP.NET Ajax

- ASP.NET Ajax is an extension of ASP.NET that provides a fast and simple way to create Ajax-enabled applications.
- The ASP.NET Ajax Control Toolkit contains rich controls that implement Ajax functionality.
- The key part of every ASP.NET Ajax-enabled application is the `ScriptManager` control, which manages the client-side scripts that enable asynchronous functionality.
- The `ToolkitScriptManager` bundles all the scripts associated with ASP. NET Ajax Toolkit controls to optimize the application's performance.
- The `UpdatePanel` control eliminates full-page refreshes by isolating a section of a page for a partial-page update.
- The components that an `UpdatePanel` reloads are placed in the `ContentTemplate` element.
- Several controls in the Ajax Control Toolkit are extenders—components that enhance regular ASP.NET controllers.

Terminology

<%-- --%> ASP.NET comment delimiters
<%$ %> ASP.NET expression delimiters
<%@ %> ASP.NET directive delimiters
absolute positioning
access rule in ASP.NET
action attribute of XHTML element `form`
`Ad` XML element in an `AdRotator` advertisement file
`Add` method of class `Hashtable`
`Add` method of class `HttpSessionState`
`AdRotator` ASP.NET web control
`Advertisements` XML element in an `AdRotator` advertisement file
Ajax Control Toolkit
Ajax Control Toolkit `ToolkitScriptManager` control
Ajax Control Toolkit `ValidatorCalloutExtender` control
alias in SQL
`AlternateText` element in an `AdRotator` advertisement file
`AS` SQL keyword
`asp:` tag prefix
ASP.NET 2.0
ASP.NET Ajax
ASP.NET Ajax extender
ASP.NET Ajax `ScriptManager` control
ASP.NET Ajax `UpdatePanel` control
ASP.NET comment
ASP.NET expression
ASP.NET login control
ASP.NET server control
ASP.NET Web Site in Visual Web Developer
ASPX file
`.aspx` filename extension
authenticating a user
`authentication` element in `Web.config`
`authorization` element in `Web.config`
`AutoEventWireup` attribute of ASP.NET page
`AutoPostBack` property of a `DropDownList`
bottom tier
`BoundField` ASP.NET element
Build Page command in Visual Web Developer
Build Site command in Visual Web Developer
business logic
business object
business rule
`Button` ASP.NET Web control
client tier
code-behind file
`CodeFile` attribute in a `Page` directive
`ConnectionString` property of a `SqlDataSource`
`Content` ASP.NET control
content page in ASP.NET

Page class
Page directive in ASP.NET
Page_Init event handler
Page_Load event handler
Page_PreInit event handler
Page_Unload event handler
PageSize property of a GridView
Parameter ASP.NET element
personalization
postback event of an ASP.NET page
PreInit event of an ASP.NET web page
presentation logic
RadioButtonList ASP.NET web control
RegularExpressionValidator ASP.NET validation control
relative positioning
rendering XHTML in a web browser
Request object in ASP.NET
RequiredFieldValidator ASP.NET validation control
runat ASP.NET attribute
script element in ASP.NET
SelectCommand property of a SqlDataSource
server
server control
session item
Session property of class Page
session tracking
SessionID property of class HttpSessionState
smart tag menu in Visual Web Developer
Source mode in Visual Web Developer
span XHTML element
SqlDataSource ASP.NET data control
System.Web.UI namespace
System.Web.UI.WebControls namespace
Target property of a HyperLink control
TextBox ASP.NET web control
tier in a multitier application
Timeout property of class HttpSessionState
Title property of a Page directive
Title property of a Web Form
title XHTML element
top tier
unique session ID of an ASP.NET client
Unload event of an ASP.NET page
UpdateCommand property of a SqlDataSource
validation control
ValidationExpression property of a RegularExpressionValidator control
validator
Value property of class HttpCookie
View in Browser command in Visual Web Developer
__VIEWSTATE hidden input
virtual directory
Visible property of an ASP.NET Web control
visual inheritance
web application development
web control
Web Form
Web Site Administration Tool
Web.config ASP.NET configuration file
WebControl class
WYSIWYG (What You See Is What You Get) editor
XHTML markup
XHTML tag
XmlDataSource ASP.NET data control

Self-Review Exercises

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

a) Web Form filenames end in .aspx.
b) App.config is a file that stores configuration settings for an ASP.NET web application.
c) A maximum of one validation control can be placed on a Web Form.
d) If no expiration date is set for a cookie, that cookie will be destroyed at the end of the browser session.

e) A `LoginStatus` control displays the current authenticated user name on a Web Form.
f) ASP.NET directives are delimited by `<%@` and `%>`.
g) An `AdRotator` control always displays all ads with equal frequency.
h) Each web control maps to exactly one corresponding XHTML element.
i) A `SqlDataSource` control allows a web application to interact with a database.

25.2 Fill in the blanks in each of the following statements:
a) Web applications contain three basic tiers: _____, _____, and _____.
b) A control which ensures that the data in another control is in the correct format is called a(n) _____.
c) A(n) _____ occurs when a page requests itself.
d) Every ASP.NET page inherits from class _____.
e) When a page loads, the _____ event occurs first, followed by the _____ event.
f) The _____ file contains the functionality for an ASP.NET page.
g) A(n) _____ control provides a registration form that site visitors can use to create a user account.
h) A(n) _____ defines common GUI elements that are inherited by each page in a set of _____.
i) In a multitier application, the _____ tier controls interactions between the application's clients and the application's data.

Exercises

25.3 *(`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. When the page reloads, it should reflect the specified changes to the properties of the `Label` displaying the time.

25.4 *(`WebControls` Modification)* Provide the following functionality for the example in Section 25.3.1: When users click **Register**, store their information in the `Users` table of the `Registration.mdf` database (provided in the chapter's examples directory). On postback, thank the user for providing the information.

25.5 Modify the `WebTime` example to asynchronously update the label every second. To do so, use the `UpdatePanel` and `Timer` ASP.NET Ajax controls. The `Timer` control refreshes the `UpdatePanel` each time its `Tick` even occurs. The `Interval` property of the `Timer` control determines how often the `UpdatePanel` should be refreshed.