

Web App Development with ASP.NET: A Deeper Look

29

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

—Lord Sandwich

Objectives

In this chapter you'll:

- Use the **Web Site Administration Tool** to modify web app configuration settings.
- Restrict access to pages to authenticated users.
- Create a uniform look-and-feel for a website using master pages.
- Use ASP.NET Ajax to improve the user interactivity of your web apps.

29.1	Introduction	29.3	ASP.NET Ajax
29.2	Case Study: Password-Protected Books Database App	29.3.1	Traditional Web Apps
29.2.1	Examining the ASP.NET Web Forms Application Template	29.3.2	Ajax Web Apps
29.2.2	Test-Driving the Completed App	29.3.3	Testing an ASP.NET Ajax App
29.2.3	Configuring the Website	29.3.4	The ASP.NET Ajax Control Toolkit
29.2.4	Modifying the Home and About Pages	29.3.5	Using Controls from the Ajax Control Toolkit
29.2.5	Creating a Content Page That Only Authenticated Users Can Access	29.3.6	ToolkitScriptManager
29.2.6	Linking from the <code>Default.aspx</code> Page to the <code>Books.aspx</code> Page	29.3.7	Grouping Information in Tabs Using the <code>TabContainer</code> Control
29.2.7	Modifying the Master Page (<code>Site.master</code>)	29.3.8	Partial-Page Updates Using the <code>UpdatePanel</code> Control
29.2.8	Customizing the Password-Protected <code>Books.aspx</code> Page	29.3.9	Adding Ajax Functionality to ASP.NET Validation Controls Using Ajax Extenders
		29.3.10	Changing the Display Property of the Validation Controls
		29.3.11	Running the App
		29.4	Wrap-Up

Self-Review Exercises | Answers to Self-Review Exercises | Exercises

29.1 Introduction

In Chapter 23, we introduced ASP.NET and web app development. In this chapter, we introduce several additional ASP.NET web-app development topics, including:

- master pages to maintain a uniform look-and-feel across a web app's pages
- creating a password-protected website with registration and login capabilities
- using the **Web Site Administration Tool** to specify which parts of a website are password protected
- using ASP.NET Ajax to quickly and easily improve the user experience for your web apps, giving them responsiveness comparable to that of desktop apps.

As in Chapter 23, we used Visual Studio Express 2012 for Web to build this chapter's apps.

29.2 Case Study: Password-Protected Books Database App

This case study presents a web app in which a user logs into a password-protected website to view a list of publications by a selected author. The app consists of several ASPX files. For this app, we'll use the **ASP.NET Web Forms Application** template, which is one of several *starter kits* for ASP.NET-based website development. The templates use Microsoft's recommended practices for organizing a website and separating the website's style (that is, its look and feel) from its content. The default site has three primary pages—**Home**, **About** and **Contact**—and is pre-configured with *login* and *registration* capabilities. The template also provides so-called master pages for both desktop and mobile web browsers—these provide a common look-and-feel for all the pages in the website.

We begin by examining some features of the default **ASP.NET Web Forms Application** that the IDE generates when you choose this template. Next, we test drive the completed

website to demonstrate the changes we made. Then, we provide step-by-step instructions to guide you through building the website.

There are many additional features in the default website that we do not use in this app. To learn more about the IDE's ASP.NET templates visit:

bit.ly/ASPNETtemplates

29.2.1 Examining the ASP.NET Web Forms Application Template

To test the default website, you'll begin by creating it. In Chapter 23, you created a new website by selecting **FILE > New Web Site....** To be able to take advantage of future new ASP.NET features, Microsoft recommends that you use **FILE > New Project...** instead:

1. Select **FILE > New Project...** to display the **New Project** dialog.
2. In the dialog's left column, ensure that **Visual C#** is expanded in the **Templates** node, then select **Web** to display the list of ASP.NET templates.
3. In the dialog's middle column, select **ASP.NET Web Forms Application**.
4. Choose a location for your website, name it Bug2Bug and click **OK** to create it.

Fig. 29.1 shows the website's contents in the **Solution Explorer**. We expanded the **Account** folder to show you the pages for registration, login and account management.

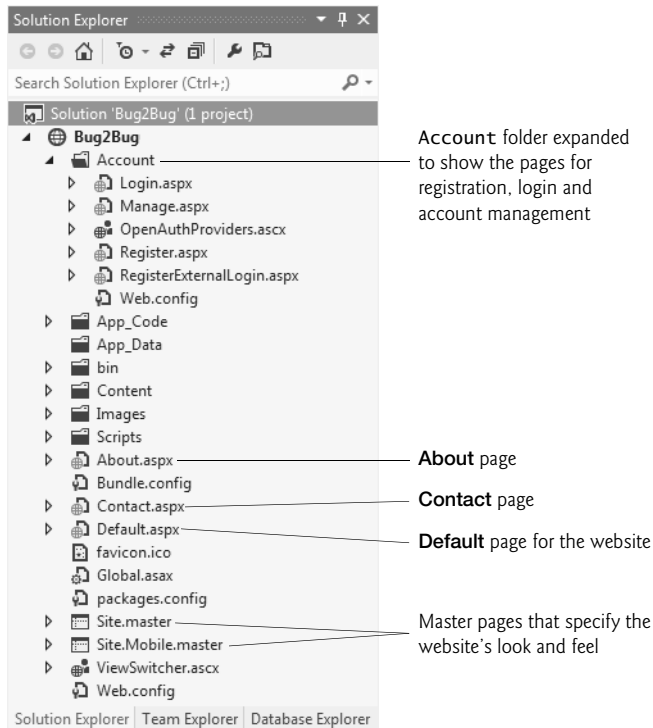


Fig. 29.1 | ASP.NET Web Forms Application in the Solution Explorer.

Executing the Website

You can now execute the website. Select the `Default.aspx` page in the **Solution Explorer**, then type *Ctrl + F5* to display the default page shown in Fig. 29.2.

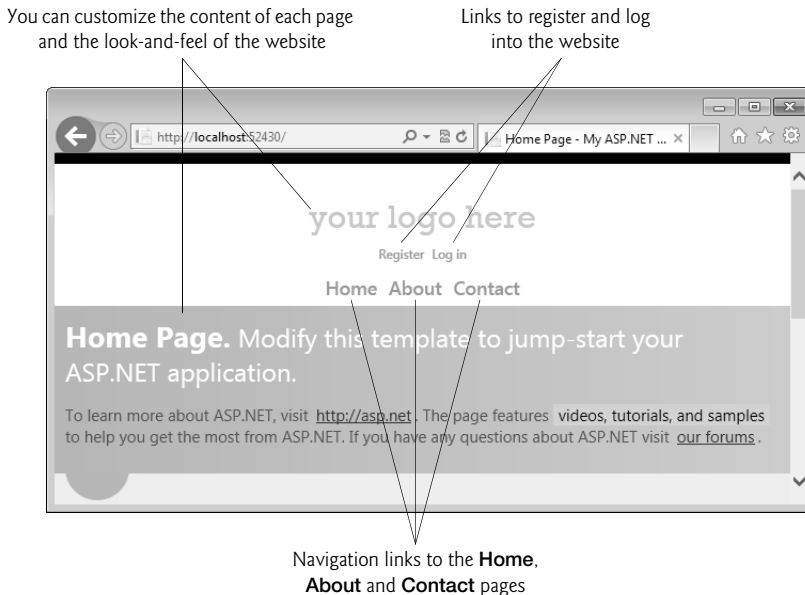


Fig. 29.2 | Default **Home** page of a website created with the **ASP.NET Web Forms Application** template.

Navigation and Pages

The default website contains **Home**, **About** and **Contact** pages—so-called **content pages**—that you'll customize in subsequent sections. The **Home**, **About** and **Contact** links near the top of the page allow you to navigate to each of the site's pages. In Section 29.2.7, you'll add another link to the navigation bar to allow users to browse book information. If you make your browser window wider, the page dynamically adjusts to the new width and repositions the **Register**, **Log in**, **Home**, **About** and **Contact** links to the upper-right corner of the page, and the text **Your Logo Here** to the upper-left corner.

As you navigate between the pages, notice that each page has the same look and feel, with the text **Your Logo Here**, and the **Register**, **Log in**, **Home**, **About** and **Contact** links at the top of the page. This commonality is typical of professional websites. The default site uses a **master page** and cascading style sheets (CSS) to achieve this. A master page defines common GUI elements that are displayed by each page in a set of content pages. Just as C# classes can inherit instance variables and methods from existing classes, content pages can inherit elements from master pages—this is a form of visual inheritance.

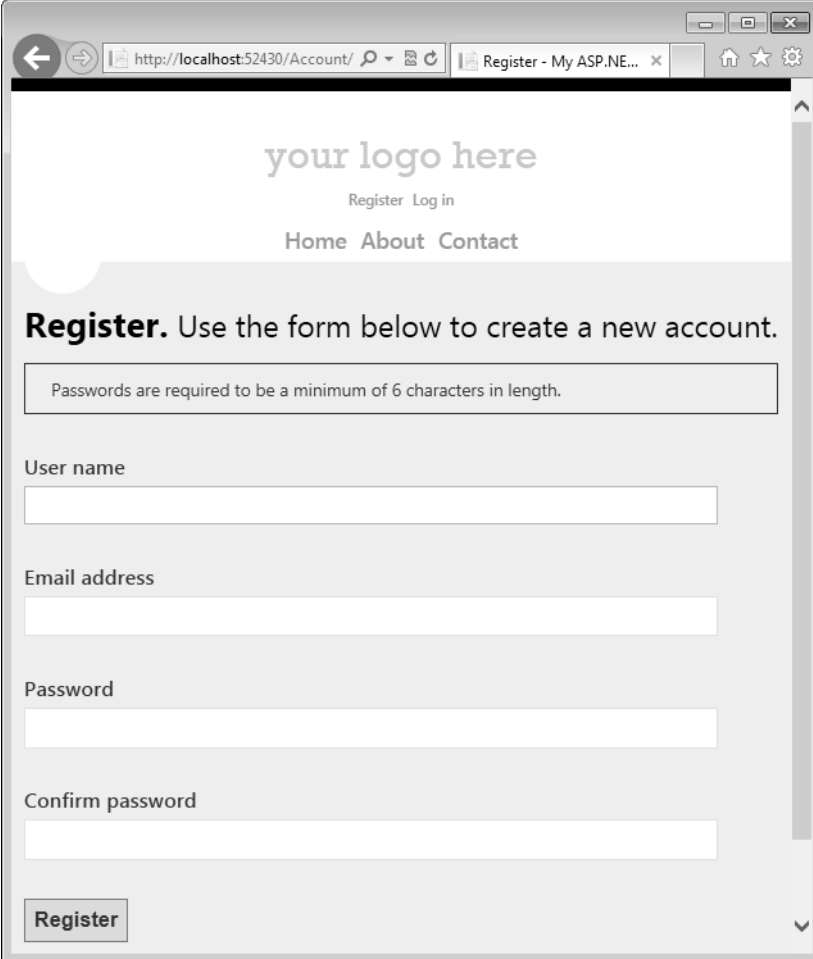
Login and Registration Support

Websites commonly provide “membership capabilities” that allow users to register at a website and log in. The default **ASP.NET Web Forms Application** is pre-configured to sup-

port registration and login capabilities. The default website has a pre-configured registration database that stores usernames and passwords. In addition, ASP.NET websites now provide capabilities for logging in with OAuth and OpenID services like Facebook, Twitter and Google. You can learn more about this at

bit.ly/ASPNETOAuthOpenID

At the top of each page are **Register** and **Log in** links. You can log in only if you've previously registered. If you are already registered with the site, you can click the **Log In** link then log in with your username and password. Since this is the first time the site has been executed, you'll need to register to be able to log in. Click the **Register** link to display the **Register** page (Fig. 29.3).



The screenshot shows a web browser window with the address bar displaying `http://localhost:52430/Account/`. The page title is "Register - My ASP.NET...". The main content area features a header with the text "your logo here" and links for "Register" and "Log in". Below the header is a navigation bar with links for "Home", "About", and "Contact". The main heading is "Register. Use the form below to create a new account." followed by a note: "Passwords are required to be a minimum of 6 characters in length." The form contains four input fields: "User name", "Email address", "Password", and "Confirm password". A "Register" button is located at the bottom left of the form.

Fig. 29.3 | Registration page.

For the purpose of this case study, we created an account with the username `testuser1` and the password `testuser1`. When you're logged into the site, the **Log in** link changes to a **Log off** link and the **Register** link is replaced with

Hello, *YourUserName*

where *YourUserName* is a link to a page where you can manage your account (e.g., to change your password). You do not need to be registered or logged into the default website to view the **Home**, **About** and **Contact** pages, but you will need to log into the final version of the Bug2Bug website to view the **Books** page.

29.2.2 Test-Driving the Completed App

This example uses *authentication* to protect a page so that only registered users who are logged into the website can access the page. Such users are known as the site's members. Authentication is a crucial tool for a site that allows only members to view protected or premium content. In this app, website visitors must log in before they're allowed to view the publications in the Books database.

Let's open the completed Bug2Bug website and execute it so that you can see the authentication functionality in action. Perform the following steps:

1. Close the app you created in Section 29.2.1—you'll reopen this website so that you can customize it in Section 29.2.3.
2. Open the `Bug2Bug.sln` file located in the Bug2Bug folder with this chapter's examples.

The website appears as shown in Fig. 29.4. We modified the site's master page so that the top of the page displays an image. Also, the navigation bar contains a link for the **Books** page that you'll create later in this case study.



Fig. 29.4 | Home page for the completed Bug2Bug website.

Try to visit the **Books** page by clicking the **Books** link in the navigation bar. Because this page is password protected in the Bug2Bug website, the website automatically redirects you to the **Login** page instead—you cannot view the **Books** page without logging in first. If you’ve not yet registered at the completed Bug2Bug website, click the **Register** link to create a new account. If you have registered, log in now.

If you are logging in, when you click the **Log in** Button on the **Log In** page, the website attempts to validate your username and password by comparing them with the usernames and passwords that are stored in a database on the server—this database is created for you with the **ASP.NET Web Forms Application** template. If there’s a match, you are **authenticated** (that is, your identity is confirmed) and you’re redirected to the **Books** page (Fig. 29.5). If you’re registering for the first time, the server ensures that you’ve filled out the registration form properly, then logs you in and redirects you to the **Books** page.



Fig. 29.5 | Books.aspx displaying a drop-down list for selecting an author.

The **Books** page 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, the page displays only the drop-down list until you make a selection. Links appear at the bottom of the table that allow you to access additional pages of data—we configured the table to display only four rows of data at a time. When the user chooses an author, a post-back occurs, and the page is updated to display information about books written by the selected author (Fig. 29.6).

Logging Out of the Website

When you’re logged in, the **Log In** link is replaced in each page with the message “Hello, *username*” where *username* is your log in name, and a **Log off** link. When you click **Log off**, the website redirects you to the **Home** page (Fig. 29.4).



Fig. 29.6 | Books.aspx displaying books by Paul Deitel.

29.2.3 Configuring the Website

Now that you're familiar with how this app behaves, you'll modify the default website you created in Section 29.2.1. Thanks to the default website's rich functionality, the only C# code you'll write for this example is in the code-behind file for the new Books page that you'll add to the site. The **ASP.NET Web Forms Application** template 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 password-protected books database app.

Step 1: Opening the Website

Open the default website that you created in Section 29.2.1.

Step 2: Setting Up Website Folders

For this website, you'll create a new folder for the password-protected page. Such parts of your website are typically placed in one or more separate folders. As you'll see shortly, you can control access to specific folders in a website.

You can choose any name you like for the new folder—we chose **ProtectedContent** for the folder that will contain the password-protected **Books** page. To create the folder,

right click the project name in the **Solution Explorer**, select **Add > New Folder** and type the name ProtectedContent.

Step 3: Importing the Website Header Image

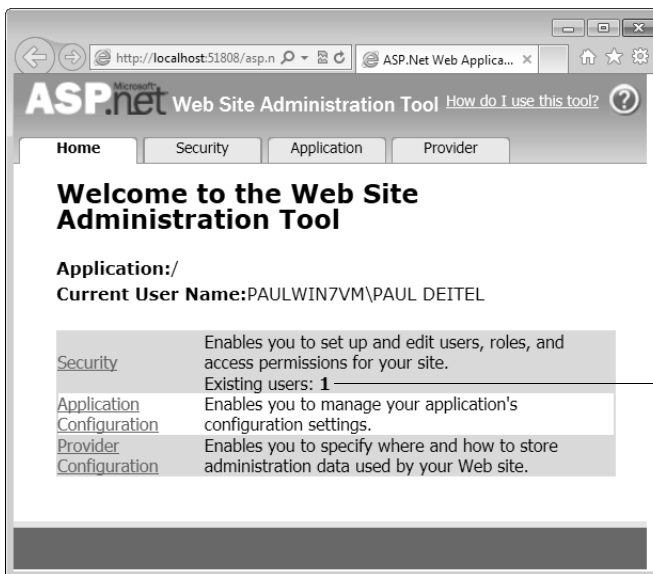
At the top of each page in the final Bug2Bug website, you saw a header image. Next, you'll add that image to the project's Images folder:

1. In Windows Explorer, locate the folder containing this chapter's examples.
2. Drag the image bug2bug.png from the images folder in Windows Explorer into the Images folder in the **Solution Explorer** to copy the image into the website.

Step 4: Opening the Web Site Administration Tool

In this app, we want to ensure that only authenticated users are allowed to access Books.aspx (which you'll create in Section 29.2.5) to view the information in the database. Previously, we created all of our ASPX pages in a website's root directory. 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 users won't be able to view any pages of the website except the login and registration pages. To restrict access to the **Books** page, we place it in a folder other than the project's root folder, then restrict access to that folder. You'll now configure the website to allow only authenticated users (that is, users who have logged in) to view the pages in the ProtectedContent folder. Perform the following steps:

1. Select **PROJECT > ASP.NET Configuration** to open the **Web Site Administration Tool** in a web browser (Fig. 29.7). This tool allows you to configure various options that determine how your site behaves.



This will say **0** if you have not yet created an account to test the website

Fig. 29.7 | Web Site Administration Tool for configuring a web app.

- Click either the **Security** link or the **Security** tab to open a web page in which you can set security options (Fig. 29.8), such as the type of authentication the app should use. By default, website users are authenticated by entering username and password information in a web form.

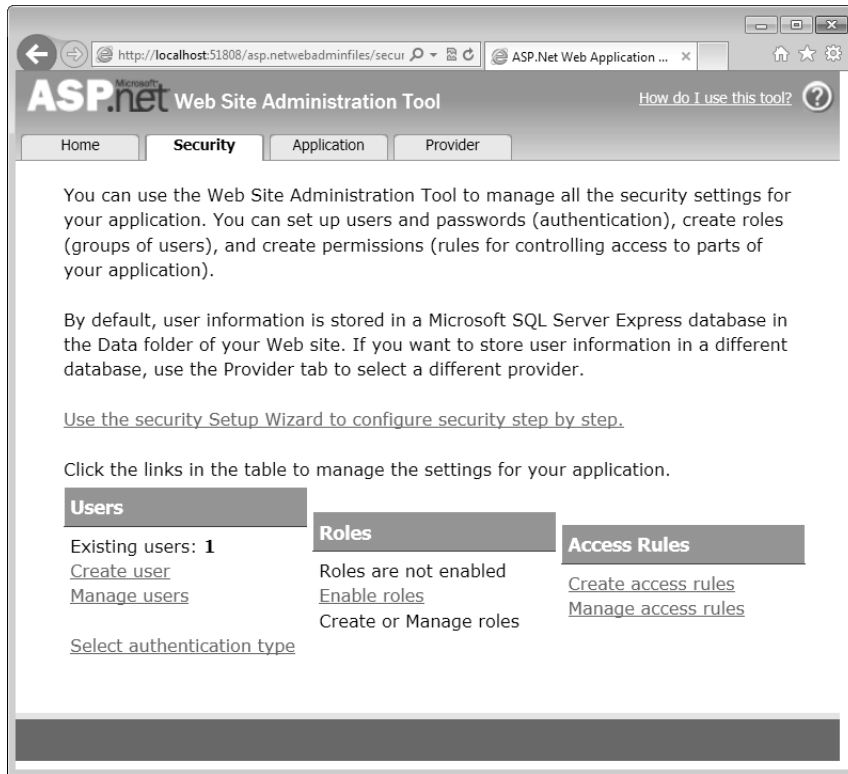


Fig. 29.8 | Security page of the Web Site Administration Tool.

Step 5: Configuring the Website's Security Settings

Next, you'll configure the ProtectedContent folder to grant access only to authenticated users—anyone who attempts to access pages in this folder without first logging in will be redirected to the Login page. Perform the following steps:

- Click the **Create access rules** link in the **Access Rules** column of the **Web Site Administration Tool** (Fig. 29.8) to view the **Add New Access Rule** page (Fig. 29.9). This page is used to create an **access rule**—a rule that grants or denies access to a particular directory for a specific user or group of users.
- Select the ProtectedContent 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 have not been authenticated.

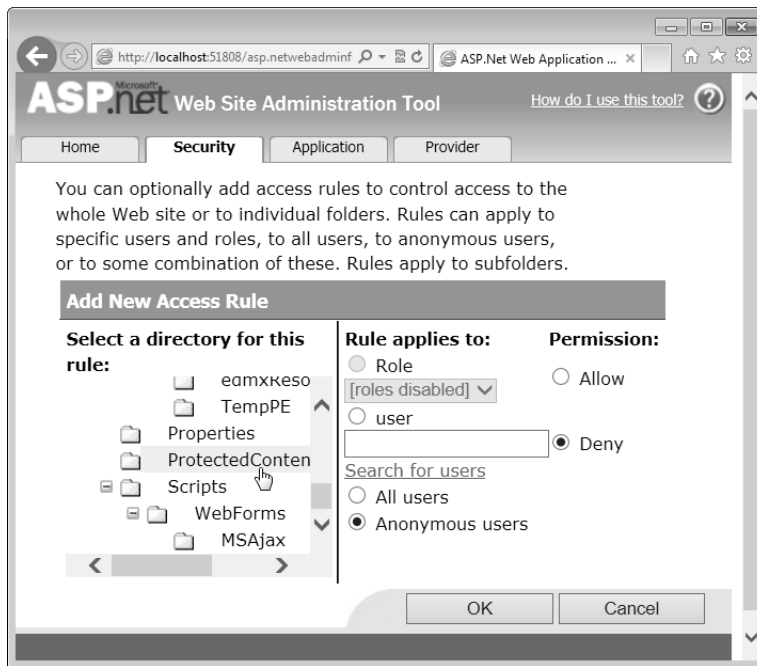


Fig. 29.9 | Add New Access Rule page used to configure directory access.

4. Finally, select **Deny** (the default) in the **Permission** column to prevent unauthenticated users from accessing pages in this directory, then click **OK** and close the browser window.


By default, unauthenticated (anonymous) users who attempt to load a page in the `ProtectedContent` directory are redirected to the `Login.aspx` page so that they can identify themselves. Because we did not set up any access rules for the `Bug2Bug` root directory, anonymous users may still access pages there.

29.2.4 Modifying the Home and About Pages

We modified the **Home** (`Default.aspx`) and **About** (`About.aspx`) pages to replace the default content. For this example, we did not modify the default content in the **Contact** page but you should edit the page if you intend to make your website publicly accessible.

Modifying the Home Page

For the **Home** page, perform the following steps:

1. Double click `Default.aspx` in the **Solution Explorer** to open it, then switch to **Design** view (Fig. 29.10). As you move the cursor over the page, you'll notice that sometimes the cursor displays as  to indicate that you cannot edit the part of the page behind the cursor. Any part of a content page that is defined in a master page can be edited only in the master page.

This cursor indicates a part of a content page that cannot be edited because it's inherited from a master page

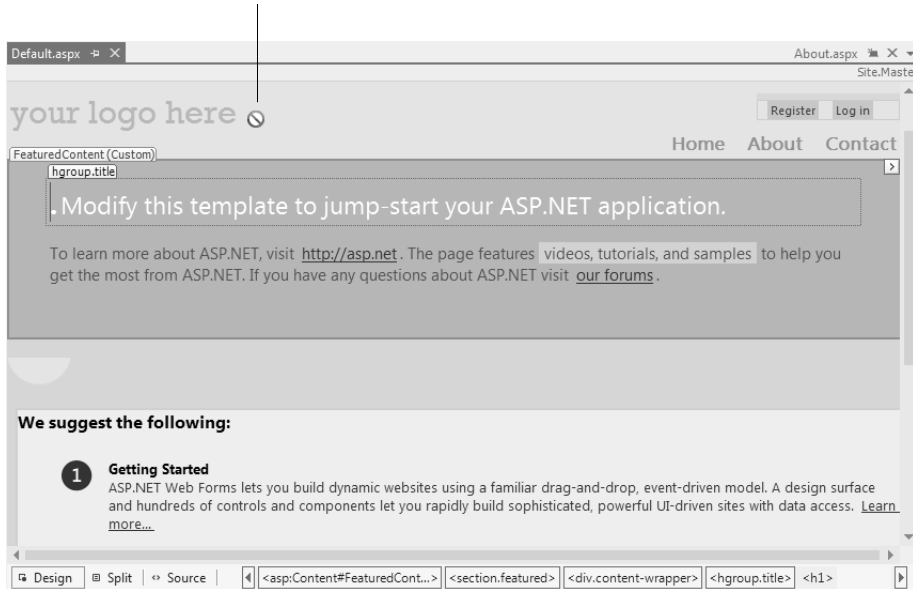


Fig. 29.10 | Default.aspx page in Design view.

2. There are two predefined content areas that you can change in Default.aspx—FeaturedContent (light blue background just below the page header) and MainContent (light gray background). For this example, we deleted the content in MainContent.
3. In the FeaturedContent section, change the text "Modify this template to jump-start your ASP.NET application." to "Welcome to Our Password-Protected Book Information Site."
4. Select the text of the other paragraph in the FeaturedContent section and replace it with "To learn more about our books, click here or click the Books tab in the navigation bar above. You must be logged in to view the Books page." In a later step, you'll link the words "click here" to the **Books** page.
5. Save and close the Default.aspx page.

Modifying the About Page

For the **About** page, perform the following steps:

1. Open About.aspx and switch to **Design** view. There are two predefined content areas that you can change in About.aspx—named MainContent and aside (located at the right side of the page). For this example, we deleted the content in aside.
2. In the MainContent section of the page, change the text "Your app description page." to "Bug2Bug password-protected book information example."

3. Replace the text of the next three paragraphs with, "This database-driven example demonstrates how to use the ASP.NET Web Forms Application template and how to password protect portions of your site."
4. Save and close the About.aspx page.

29.2.5 Creating a Content Page That Only Authenticated Users Can Access

We now create the Books.aspx file in the ProtectedContent 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 create a new account, both of which will authenticate the user, then redirect back to Books.aspx. To create the page, perform the following steps:

1. Right click the ProtectedContent folder in the **Solution Explorer** and select **Add > New Item....** In the resulting dialog's **Web** category, select **Web Form Using Master Page**, specify the file name Books.aspx and click **Add**.
2. In the **Select a Master Page** dialog, select Site.master and click **OK**. The IDE creates the file and opens it.
3. Switch to **Design** view, then select **DOCUMENT** from the ComboBox in the **Properties** window.
4. Change the **Title** property to Books, then save and close the page.

You'll customize this page and create its functionality shortly.

29.2.6 Linking from the Default.aspx Page to the Books.aspx Page

Next, you'll add a hyperlink from the text "click here" in the Default.aspx page to the Books.aspx page. To do so, perform the following steps:

1. Open the Default.aspx page and switch to **Design** view.
2. Select the text "click here".
3. Select **FORMAT > Convert to Hyperlink** to display the **Hyperlink** dialog. You can enter a URL here, or you can link to another page within the website.
4. Click the **Browse...** Button to display the **Select Project Item** dialog, which allows you to select another page in the website.
5. In the left column, select the ProtectedContent directory.
6. In the right column, select Books.aspx, then click **OK** to dismiss the **Select Project Item** dialog and click **OK** again to dismiss the **Hyperlink** dialog.

When browsing the website, users can now click the **click here** link in the Default.aspx page to browse to the Books.aspx page. If a user is not logged in, clicking this link will redirect the user to the Login page.

29.2.7 Modifying the Master Page (Site.master)

Next, you'll modify the website's master page, which defines the common elements we want to appear on each page. A master page is like a base class in an 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 the areas designated by the master page's placeholders.

For example, it's common for every page in a website to include navigation links for navigating to other pages on the site. If a 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 in one file and have it appear on all the content pages. If the navigation links change, you need to modify only the master page—any content pages that use it are updated the next time the page is requested.

In the final version of this website, we modified the master page to include the Bug2Bug logo in the header at the top of every page.

Inserting an Image in the Header

To display the logo, you'll replace the text "your logo here" with an image in the header of the master page. Each content page based on this master page will include the logo. Perform the following steps to add the image:

1. Open `Site.master` and switch to **Design** view.
2. Delete the text "your logo here" at the top of the page.
3. In the **Toolbox**, double click **Image** to add an Image control where the text used to be.
4. Edit the Image control's `ImageUrl` property to point to the `bug2bug.png` image in the `Images` folder.

Adding a Books Link to the Navigation Links

Next, you'll add a link to the **Books** page. This will appear with the other links at the top of the each page. The easiest way to do this is to edit the master page's markup directly. Perform the following steps:

1. Switch to **Source** view to edit the master page's markup.
2. Locate the following line of code:

```
<li><a runat="server" href="~/About">About</a></li>
```

This represents the link to the site's **About** page. Copy this line of code and paste the copy *above* the existing line.

3. In the new copy, replace `~/About` with `~/ProtectedContent/Books` and `About` with `Books`. This creates a link to the `Books.aspx` page in the website's `ProtectedContent` folder. The new line of code should now appear as follows:

```
<li><a runat="server" href="~/ProtectedContent/Books">Books
</a></li>
```

4. Save and close the `Site.master` to complete the changes to the master page.

29.2.8 Customizing the Password-Protected Books.aspx Page

You're now ready to customize the `Books.aspx` page to display the book information for a particular author. The `Books.aspx` 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.

Creating the Entity Data Model for the Books Database

To work with the Books database through LINQ, first you need to generate the Entity Data Model classes based on the Books database that you used in Chapter 22—`Books.mdf` is provided in the `databases` directory of this chapter's examples folder. Perform the following steps:

1. Right click the project name in the Solution Explorer and select **Add > Add New Item...** to display the **Add New Item** dialog.
2. Select the **Data** category in the left column, then **ADO.NET Entity Data Model** in the middle column. Change the **Name** to `BooksModel.edmx` and click **Add**.
3. In the **Entity Data Model Wizard's Choose Model Contents** step, select **Generate from Database** and click **Next >**.
4. In the **Choose Your Data Connection** step, click **New Connection...**
5. In the **Connection Properties** dialog, click **Browse...** then select the `Books.mdf` file (located in the `databases` folder with this chapter's examples) and click **Open**. Click **OK** to dismiss the **Connection Properties** dialog and click **Next >** in the **Entity Data Model Wizard**.
6. A dialog appears indicating that the data file (`Books.mdf`) is not in the project. Click **Yes** to copy it to the project.
7. In the **Entity Data Model Wizard** dialog's **Choose Your Database Objects and Settings** step, check the **Tables** node so that all three database tables (`AuthorISBN`, `Authors` and `Titles`) will be part of the Entity Data Model. By default, the IDE names the model `BooksModel`. Ensure that **Pluralize or singularize generated object names** is checked, keep the other default settings and click **Finish**. The IDE displays the `BooksModel` in the editor. Save the Entity Data Model then select **BUILD > Build Solution** to ensure that the model's new classes are compiled.

Adding a DropDownList to Display the Authors' First and Last Names

Now that you've created the model, you'll add controls to `Books.aspx` that will display the data on the web page. First, you'll add a `DropDownList` for selecting an author. Normally, only one database column's value can be displayed in a `DropDownList`. For that reason, you'll configure a custom query to display each author's full name.

1. Open `Books.aspx` in **Design** mode, then add the text `Select author:` and a `DropDownList` control named `authorsDropDownList` in the page's `MainContent` area. The `DropDownList` initially displays the text `Unbound`. In this app, you'll programmatically bind the results of a LINQ to Entities query to the `authorsDropDownList`.
2. Set the `authorsDropDownList` 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'll see shortly, this enables us to populate the `GridView` with new data based on the user's selection.

3. Double click the `authorsDropDownList` control to create its `SelectedIndexChanged` event handler, which is called when the user makes a new selection in the `DropDownList`.

Creating a GridView to Display the Selected Author's Books

Next, you'll add a `GridView` to `Books.aspx` for displaying the book information for the author selected in the `authorsDropDownList`.

1. Insert the cursor to the right of the `authorsDropDownList` then press *Enter* to create a new paragraph.
2. Add a `GridView` named `titlesGridView` to the new paragraph.
3. In the `GridView`'s smart tasks menu, click **Auto Format....** In the dialog that appears, select **Professional** and click **OK** to give the `GridView` a nicer look and feel.

Code-Behind File for the Books Page

Figure 29.11 shows the code for the completed code-behind file. Line 12 defines the `DbContext` object that's used in the LINQ queries that are bound to the `Books` page's controls.

```

1  // Fig. 29.11: Books.aspx.cs
2  // Code-behind file for the password-protected Books page.
3  using System;
4  using System.Data.Entity;
5  using System.Linq;
6
7  namespace Bug2Bug.ProtectedContent
8  {
9      public partial class Books : System.Web.UI.Page
10     {
11         // Entity Framework DbContext
12         BooksEntities dbcontext = new BooksEntities();
13
14         // Load event handler for Books page
15         protected void Page_Load(object sender, EventArgs e)
16         {
17             // if this is the first time the page is loading
18             if (!IsPostBack)
19             {
20                 dbcontext.Authors.Load(); // load Authors table into memory
21
22                 // LINQ query that populates authorsDropDownList
23                 var authorsQuery =
24                     from author in dbcontext.Authors.Local
25                     orderby author.LastName, author.FirstName
26                     select new
27                     {
28                         Name = author.LastName + ", " + author.FirstName,
29                         author.AuthorID
30                     };
31

```

Fig. 29.11 | Code-behind file for the password-protected **Books** page. (Part 1 of 2.)

```

32         // specify the field used as the selected value
33         authorsDropDownList.DataValueField = "AuthorID";
34
35         // specify the field displayed in the DropDownList
36         authorsDropDownList.DataTextField = "Name";
37
38         // set authorsQuery as the authorsDropDownList's data source
39         authorsDropDownList.DataSource = authorsQuery;
40
41         authorsDropDownList.DataBind(); // displays query results
42     } // end if
43 } // end method Page_Load
44
45 // display selected author's books
46 protected void authorsDropDownList_SelectedIndexChanged(
47     object sender, EventArgs e)
48 {
49     dbcontext.Authors.Load(); // load Authors table into memory
50
51     // use LINQ to get Author object for the selected author
52     Author selectedAuthor =
53         (from author in dbcontext.Authors.Local
54          where author.AuthorID ==
55               Convert.ToInt32(authorsDropDownList.SelectedValue)
56          select author).First();
57
58     // query to get books for the selected author
59     var titlesQuery =
60         from book in selectedAuthor.Titles
61         orderby book.Title1
62         select book;
63
64     // set titlesQuery as the titlesGridView's data source
65     titlesGridView.DataSource = titlesQuery;
66     titlesGridView.DataBind(); // displays query results
67 } // end method authorsDropDownList_SelectedIndexChanged
68 } // end class Books
69 } // end namespace Bug2Bug.ProtectedContent

```

Fig. 29.11 | Code-behind file for the password-protected **Books** page. (Part 2 of 2.)

Page_Load Event Handler

When the `Page_Load` event handler (lines 15–43) executes, it first determines whether the `Load` event was due to a *postback*. If not, line 20 loads the database's `Authors` table so that we can query it to populate the `authorsDropDownList`. Lines 23–30 define a LINQ query that combines the author names so that they're can be displayed in the format *LastName, FirstName* in the `DropDownList`. The query's results are anonymous objects that each contain a `Name` and an `AuthorID` property. Line 33 indicates that the value of the `authorsDropDownList`'s selected item will be the author's `AuthorID`, and line 36 indicates that the the `authorsDropDownList` will display the authors `Name`. Line 39 sets `authorsDropDownList`'s data source to the results of the LINQ query. Calling `DataBind` on the `authorsDropDownList` loads the query results.

authorsDropDownList_SelectedIndexChanged Event Handler

When the user selects an author from the authorsDropDownList, the authorsDropDownList_SelectedIndexChanged event handler (lines 46–67) loads the author’s books into the titlesGridView. First, line 49 loads the database’s Authors table so that we can get the selected author’s Author object (lines 52–56). There can be only one match for the selected author’s AuthorID, so we use LINQ extension method *First* to get the Author object from the LINQ query result. The titlesQuery (lines 59–62) sorts the author’s books by title. Line 65 sets the query as titlesGridView’s data source, then line 66 calls *DataBind* to populate the GridView with the author’s books.

29.3 ASP.NET Ajax

In this section, you learn the difference between a traditional web app and an **Ajax (Asynchronous JavaScript and XML) web app**. You also learn how to use ASP.NET Ajax to quickly and easily improve the user experience for your web apps. To demonstrate ASP.NET Ajax capabilities, you enhance the validation example of Section 23.6 by displaying the submitted form information without reloading the entire page. The only modifications to this web app appear in the Validation.aspx file. You use Ajax-enabled controls to add this feature.

29.3.1 Traditional Web Apps

Figure 29.12 presents the typical interactions between the client and the server in a traditional web app, such as one that uses a user registration form. The user first fills in the form’s fields, then submits the form (Fig. 29.12, *Step 1*). The browser generates a request to the server, which receives the request and processes it (*Step 2*). The server generates and sends a response containing the exact page that the browser renders (*Step 3*), which causes the browser to load the new page (*Step 4*) and temporarily makes the browser window blank. The client *waits* for the server to respond and *reloads the entire page* with the data

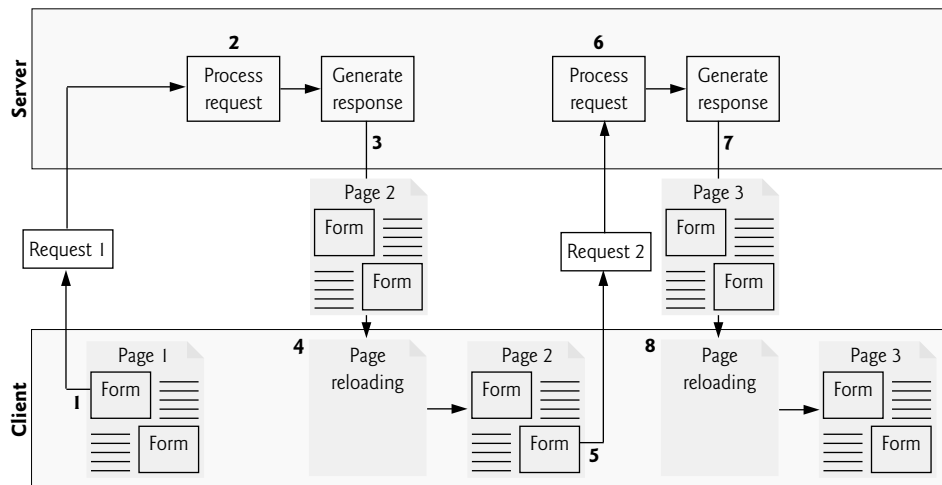


Fig. 29.12 | Traditional web app reloading the page for every user interaction.

from the response (*Step 4*). While such a **synchronous request** is being processed on the server, the user cannot interact with the web page for an indeterminate period of time. If the user interacts with and submits another form, the process begins again (*Steps 5–8*).

This model was designed for a web of hypertext documents—what some people called the “brochure web.” As the web evolved into a full-scale apps platform, the model shown in Fig. 29.12 yielded “choppy” user experiences. Every full-page refresh required users to reload the full page. Users began to demand a more responsive model.

29.3.2 Ajax Web Apps

Ajax web apps add a layer between the client and the server to manage communication between the two (Fig. 29.13). When the user interacts with the page, the client requests information from the server (*Step 1*). The request is intercepted by the ASP.NET Ajax controls and sent to the server as an **asynchronous request** (*Step 2*)—the user can continue interacting with the app in the client browser while the server processes the request. Other user interactions could result in additional requests to the server (*Steps 3 and 4*). Once the server responds to the original request (*Step 5*), the ASP.NET Ajax control that issued the request calls a client-side function to process the data returned by the server. This function—known as a **callback function**—uses **partial-page updates** (*Step 6*) to display the data in the existing web page *without reloading the entire page*. At the same time, the server may be responding to the second request (*Step 7*) and the client browser may be starting another partial-page update (*Step 8*). The callback function updates only a designated part of the page. Such partial-page updates help make web apps more responsive, making them feel more like desktop apps. The web app does not load a new page while the user interacts with it. In the following section, you use ASP.NET Ajax controls to enhance the `Validation.aspx` page.

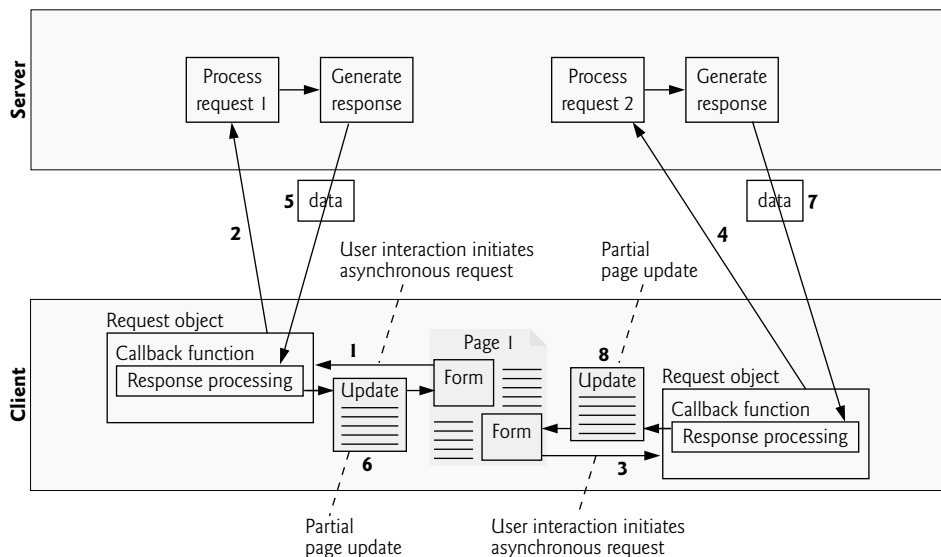


Fig. 29.13 | Ajax-enabled web app interacting with the server asynchronously.

29.3.3 Testing an ASP.NET Ajax App

To demonstrate ASP.NET Ajax capabilities, you'll enhance the **Validation** app from Section 23.6 by adding ASP.NET Ajax controls. There are no C# code modifications to this app—all of the changes occur in the .aspx file.

Testing the App in Your Default Web Browser

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

1. Open the **Validation** app's project by double clicking its `Validation.sln` file (located in the `Validation` folder with this chapter's examples).
2. Select `Validation.aspx` in the **Solution Explorer**, then type `Ctrl + F5` to execute the web app in your default web browser.

Figure 29.14 shows a sample execution of the enhanced app. In Fig. 29.14(a), we show the contact form split into two tabs via the `TabContainer` Ajax control. You can switch between the tabs by clicking the title of each tab. Fig. 29.14(b) shows a `ValidatorCalloutExtender` control, which displays a validation error message in a callout that points to the control in which the validation error occurred, rather than as text in the page. Fig. 29.14(c) shows the updated page with the data the user submitted to the server.

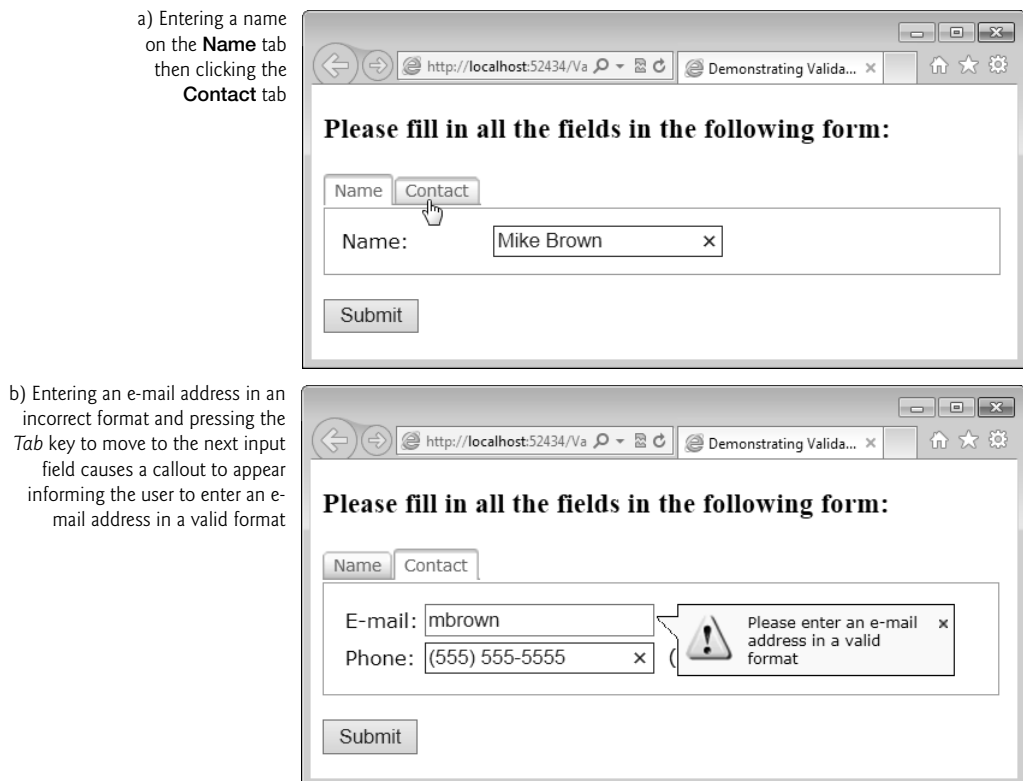


Fig. 29.14 | Validation app enhanced by ASP.NET Ajax. (Part 1 of 2.)

c) After filling out the form properly and clicking **Submit**, the submitted data is displayed at the bottom of the page with a partial page update

Please fill in all the fields in the following form:

Name **Contact**

E-mail:

Phone:

Submit

Thank you for your submission
We received the following information:
Name: Mike Brown
E-mail: mbrown@bug2bug.com
Phone: (555) 555-5555

Fig. 29.14 | Validation app enhanced by ASP.NET Ajax. (Part 2 of 2.)

29.3.4 The ASP.NET Ajax Control Toolkit

There's a tab of basic **AJAX Extensions** controls in the **Toolbox**. Microsoft also provides the **ASP.NET Ajax Control Toolkit** that contains many more Ajax-enabled, rich GUI controls. In this section, you'll open the original **Validation** example from Chapter 23, use Visual Studio's *NuGet package manager* to add the Ajax Control Toolkit to the project, then incorporate the toolkit's controls into your **Toolbox**. In subsequent sections, you'll then use the toolkit's **TabContainer** control to enhance the **Validation** example with Ajax capabilities. To learn more about the toolkit's many other extenders and controls, visit:

www.asp.net/ajaxlibrary/ajaxcontroltoolkitsamplesite

Using NuGet to Download the Ajax Control Toolkit and Add It to the Project

To download the Ajax Control Toolkit and add it to the project:

1. Open the original **Validation** example, which is located in the folder named `CopyOfValidationExample` provided with this chapter's examples. We provided a solution (`.sln`) file so that you can double click it to open the project in Visual Studio Express 2012 for Web (or your full version of Visual Studio).
2. Right click the solution name at the top of the **Solution Explorer** window and select **Manage NuGet Packages for Solution**.
3. In the dialog that appears, locate `AjaxControlToolkit`—you can find this quickly by selecting the **Online** tab at the left side of the dialog, then typing `ajax` in the dialog's **Search Online** field.
4. Select `AjaxControlToolkit`, then click **Install**. In the **Select Projects** dialog, ensure that the **Validation** project is selected and click **OK**.

This will download the toolkit and place its files into the `Bin` folder within your project. When the installation is complete, close the **Manage NuGet Packages** window.

Adding the ASP.NET Ajax Controls to the Toolbox

Next, you'll add the toolkit's controls to the **Toolbox**, so you can drag and drop them onto your Web Forms. To do so, perform the following steps:

1. Right click inside the **Toolbox** and choose **Add Tab**, then type `AjaxControlToolkit` in the new tab.
2. Right click in the new **AjaxControlToolkit** tab and select **Choose Items...** to open the **Choose Toolbox Items** dialog.
3. Click **Browse...** then locate the `Bin` folder within the `Validation` project.
4. Select the file `AjaxControlToolkit.dll` then click **Open**. All of the toolkit's controls will automatically be selected in the `.NET Framework Components` tab.
5. Click **OK** to close dialog. The toolkit's controls now appear in the **Toolbox's Ajax-ControlToolkit** section.
6. If the control names are not in alphabetical order, you can sort them alphabetically, by right clicking in the list of `Ajax Control Toolkit` controls and selecting **Sort Items Alphabetically**.

29.3.5 Using Controls from the Ajax Control Toolkit

You'll now enhance the `Validation` app by adding ASP.NET Ajax controls. The key control in every ASP.NET Ajax-enabled app is the **ScriptManager** (in the **Toolbox's AJAX Extensions** tab), which manages the JavaScript client-side code (called scripts) that enable asynchronous Ajax functionality. A benefit of using ASP.NET Ajax is that you do not need to know JavaScript to be able to use these scripts. The **ScriptManager** is meant for use with the controls in the **Toolbox's AJAX Extensions** tab. There can be only one **ScriptManager** per page.

29.3.6 ToolkitScriptManager

The Ajax Control Toolkit comes with an enhanced **ScriptManager** called the **ToolkitScriptManager**, which manages the scripts for the Ajax Control Toolkit's controls. This one should be used in any page with controls from the ASP.NET Ajax Toolkit. Drag a **ToolkitScriptManager** from the **AjaxControlToolkit** tab in the **Toolbox** to the top of the page—a script manager must appear *before* any controls that use the scripts it manages.



Common Programming Error 29.1

*More than one **ScriptManager** and/or **ToolkitScriptManager** control on a Web Form causes the app to throw an `InvalidOperationException` when the page is initialized.*

29.3.7 Grouping Information in Tabs Using the TabContainer Control

The **TabContainer** control enables you to group information into tabs that are displayed only if they're selected. The information in an unselected tab won't be displayed until the user selects that tab. To demonstrate a **TabContainer** control, let's split the form into two tabs—one in which the user can enter the name and one in which the user can enter the e-mail address and phone number. Perform the following steps:

1. In **Design** view, click to the right of the text **Please fill in all the fields in the following form:** and press *Enter* to create a new paragraph.
2. Drag a **TabContainer** control from the **AjaxControlToolkit** tab in the **Toolbox** into the new paragraph. This creates a container for hosting tabs. Set the **TabContainer**'s **Width** property to 450px.
3. To add a tab, open the **TabContainer Tasks** smart-tag menu and select **Add Tab Panel**. This adds a **TabPanel** object—representing a tab—to the **TabContainer**. Do this again to add a second tab. In **Design** view, you can navigate between tabs by clicking the tab headers. You can drag-and-drop elements into each tab as you would anywhere else on the page.
4. Next, change each **TabPanel**'s text label by clicking the tab, then selecting the text and typing the new text. Change **TabPanel1** to **Name** and **TabPanel2** to **Contact**.
5. Select the **Name** tab, then click in its body and insert a one row and two column table. Take the text and controls that are currently in the **Name:** row of the original table and move them to the table in the **Name** tab.
6. Switch to the **Contact** tab, click in its body, then insert a two-row-by-two-column table. Take the text and controls that are currently in the **E-mail:** and **Phone:** rows of the original table and move them to the table in the **Contact** tab.
7. Delete the original table that is currently below the **TabContainer**.

29.3.8 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. You'll now use a partial-page update to display the user's information that is submitted to the server. Perform the following steps:

1. Insert an **UpdatePanel** control (located in the **Toolbox**'s **AJAX Extensions** tab) before the **Submit** Button.
2. Partial page updates require the control(s) to update and the control that triggers the update to be placed in the **UpdatePanel**. For this reason, drag the paragraph elements containing the **submitButton** and **outputLabel** into the **UpdatePanel**.

Now, 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.

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

Several controls in the Ajax Control Toolkit are **extenders**—components that enhance the functionality of regular ASP.NET controls. In this example, we use **ValidatorCallout-Extender** controls that enhance the ASP.NET validation controls by displaying error messages in small yellow callouts next to the input fields, rather than as text in the page.

You can create a **ValidatorCalloutExtender** by opening any validator control's smart-tag menu and clicking **Add Extender...** In the **Extender Wizard** dialog (Fig. 29.15), choose **ValidatorCalloutExtender** from the list of available extenders. The extender's ID is chosen

based on the ID of the validation control you're extending, but you can rename it if you like. Click **OK** to create the extender. Do this for each of the validation controls in this example.

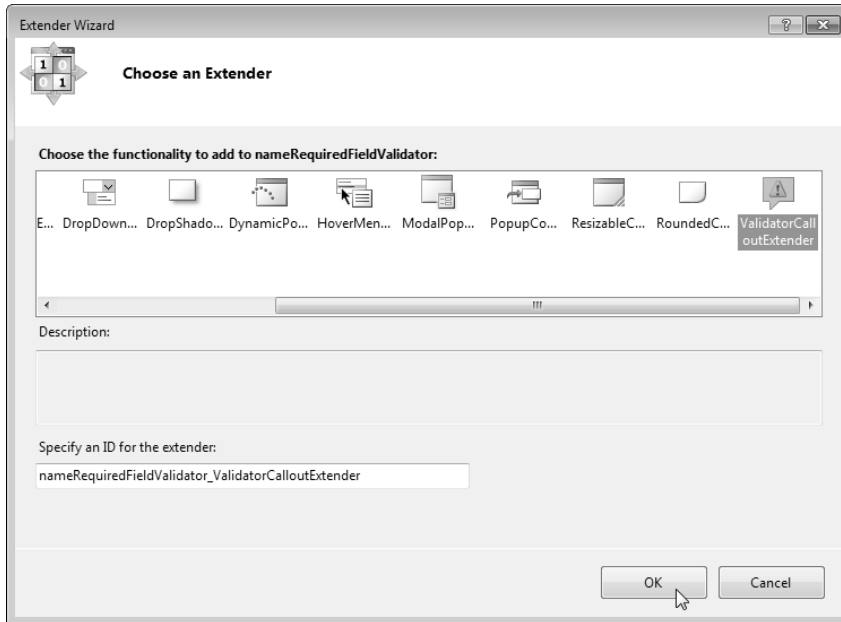


Fig. 29.15 | Creating a control extender using the **Extender Wizard**.

29.3.10 Changing the Display Property of the Validation Controls

The `ValidatorCalloutExtenders` 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, set each validation control's `Display` property to `None`.

29.3.11 Running the App

When you run this app, the `TabContainer` will display whichever tab was last displayed in the ASPX page's **Design** view. Ensure that the **Name** tab is displayed, then select `Validation.aspx` in the **Solution Explorer** and type `Ctrl + F5` to execute the app.

29.4 Wrap-Up

In this chapter, we presented a case study in which we built a password-protected web app that requires users to log in before accessing information from the Books database. You used the **Web Site Administration Tool** to configure the app to prevent anonymous users from accessing the book information. We used the **ASP.NET Web Forms Application** template, which provides preconfigured login and registration capabilities for a website. You also modified a master page that defined the website's uniform look-and-feel.

Finally, you learned the difference between a traditional web app and an Ajax web app. We introduced ASP.NET Ajax and the Ajax Control Toolkit. You learned how to

build an Ajax-enabled web app by using a `ScriptManager` and the Ajax-enabled controls of the Ajax Extensions package and the Ajax Control Toolkit.

In the next chapter, we introduce web services, which allow methods on one machine to call methods on other machines via common data formats and protocols, such as XML and HTTP. You will learn how web services promote software reusability and interoperability across multiple computers on a network such as the Internet.

Self-Review Exercises

- 29.1** State whether each of the following is *true* or *false*. If *false*, explain why.
- An access rule grants or denies access to a particular directory for a specific user or group of users.
 - When using controls from the Ajax Control Toolkit, you must include the `ScriptManager` control at the top of the ASPX page.
 - A master page is like a base class in an inheritance hierarchy, and content pages are like derived classes.
 - Ajax web apps make synchronous requests and wait for responses.
- 29.2** Fill in the blanks in each of the following statements:
- A(n) _____ defines common GUI elements that are inherited by _____.
 - The main difference between a traditional web app and an Ajax web app is that the latter supports _____ requests.
 - The _____ is a starter kit for a small multi-page website that uses recommended practices for organizing a website and separating the website's style from its content.
 - The _____ allows you to configure various options that determine how your app behaves.
 - Setting a `DropDownList`'s _____ property to `True` indicates that a postback occurs each time the user selects an item in the `DropDownList`.
 - Several controls in the Ajax Control Toolkit are _____—components that enhance the functionality of regular ASP.NET controls.

Answers to Self-Review Exercises

- 29.1** a) True. b) False. The `ToolkitScriptManager` control must be used for controls from the Ajax Control Toolkit. The `ScriptManager` control can be used only for the controls in the **Toolbox's AJAX Extensions** tab. c) True. d) False. That is what traditional web apps do. AJAX web apps can make asynchronous requests and do not need to wait for responses.
- 29.2** a) master page, content pages. b) asynchronous. c) **ASP.NET Web Forms Application** template. d) **Web Site Administration Tool**. e) `AutoPostBack`. f) extenders.

Exercises

- 29.3** (*Guestbook App Modification*) Add Ajax functionality to the Guestbook app in Exercise 23.5. Use control extenders to display error callouts when one of the user input fields is invalid.
- 29.4** (*Guestbook App Modification*) Modify the Guestbook app in Exercise 29.3 to use a `UpdatePanel` so only the `GridView` updates when the user submits the form. Because only the `UpdatePanel` will be updated, you cannot clear the user input fields in the `Submit` button's `Click` event, so you can remove this functionality.
- 29.5** (*Session Tracking Modification*) Use the **ASP.NET Web Forms Application** template that you learned about in this chapter to reimplement the session tracking example in Section 23.7.