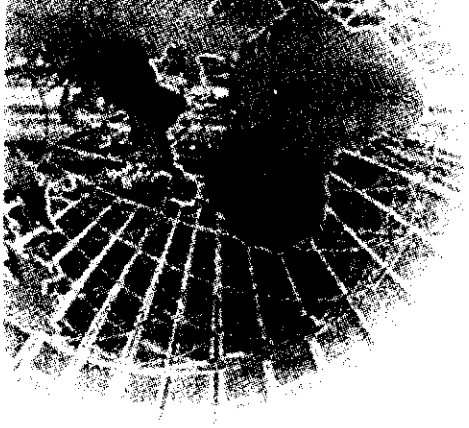


22

Introducing Login Controls

You have already been familiarized with the process of creating websites of your choice. However, only creating a website is not enough if it is not secure from unwanted users. Such users can access and steal vital information of other users, such as credit card numbers and email-ids. Considering these factors, it is necessary to secure the websites from malicious users. To implement security, we must be able to track the users who visit the website and allow only valid users to access the website resources. For tracking users, we need to collect the required information (such as name, e-mail id, and contact no.) including username and password; users are required to furnish username and password to authenticate them as valid registered users. To fulfill these tasks, we need to create a user interface for authenticating the user, and displaying the desired page based on the roles or rights given to the user. However, creating such forms or user interfaces with the help of standard ASP.NET server controls is quite tedious and time-consuming.

Microsoft realized the problem and developed a new series of server controls, called login controls. Using the login controls, you can ensure that only authenticated users are allowed to access your website, without writing a single line of code. The login controls have built-in functionality (called Membership service) for all the tasks related to authentication and authorization of users. Login controls were introduced with ASP.NET 2.0 and are also available with later versions of ASP.NET, which are ASP.NET 3.0 and 3.5.

In this chapter, we first provide a brief overview of the Membership service, which is an essential part of login controls. We then explore the login controls, which provide a rich user interface to implement authentication and authorization of users logging on to a website.

NOTE

Sometimes, in general, login is also called Sign In and creating a user is also called Sign Up. Therefore, do not get confused if you find these terms being used interchangeably in this chapter as well as in this book.

Introducing the Membership Service

The Membership service is one of the important features of ASP.NET that helps you validate and store user credentials. In order to authenticate users, the `login controls` make use of the Membership service, in which each user is provided with a unique `username` and `password`. There is a default Membership provider that is configured automatically, while using `login controls` in a Web application. In short, ASP.NET Membership service and login controls work simultaneously to achieve the goal of user registration and authentication.

The ASP.NET Membership service helps you implement following functionalities:

- Helps create new users and passwords.
- Stores the membership information, such as username, password, e-mail address and other supporting data, in data stores such as Microsoft SQL Server Database, Oracle Database, or XML file.
- Authenticates the visitors of the website.
- Helps manage passwords by creating, changing, and resetting them. It also includes a password-reset system that takes a user-supplied question and response to reset the password of a user.
- Exposes a unique identification system for authenticated users, which helps you in role-management and personalizing websites.

Now let's discuss all the login controls provided by ASP.NET. To use login controls in your website you can simply drag-and-drop the control on the Web form.. The login controls are available under the Login tab in the Toolbox of Visual Studio 2008 IDE. You can find the following controls under the Login tab:

- Login
- LoginView
- LoginStatus
- LoginName
- PasswordRecovery
- CreateUserWizard
- ChangePassword

The **Login** Control

The `Login` control provides a user interface that facilitates in authenticating users of a website on the basis of username and password. The `Login` control can also be used for designing the Home page of a website. By using the `Login` control on a Web page of any website, you can restrict access to other Web pages of the website to authenticated users only. It can be done by redirecting only authenticated users to other Web pages of the website.

The `Login` control is provided by the `Login` class. The class hierarchy of the `Login` class is as follows:

```
System.Object
  System.Web.UI.Control
    System.Web.UI.WebControls.WebControl
      System.Web.UI.WebControls.CompositeControl
        System.Web.UI.WebControls.Login
```

The `Login` class contains various methods, properties, and events used to work with the `Login` control. These members enable you to customize `Login` control, such as displaying customized messages and creating links to change password or recovering a forgotten password. The `Login` control contains built-in authentication logic; however, you can create your own logic by handling the `Authenticate` event. Noteworthy methods of the `Login` class are listed in Table 22.1:

Table 22.1: Noteworthy Methods of the Login Class

Method name	Description
`CreateChildControls`	Creates individual controls of `Login` control and associate event handlers with their events
`OnAuthenticate`	Raises the `Authenticate` event to authenticate users
`OnLoggedIn`	Raises the `LoggedIn` event when users log into a website after successful authentication
`OnLoggingIn`	Raises the `LoggingIn` event after users enter login information but before the authentication completes
`OnLoginError`	Raises the `LoginError` event when users fail to log in

Noteworthy properties of the `Login` class are listed in Table 22.2:

Table 22.2: Noteworthy Properties of the Login Class

Property	Description
`BorderPadding`	Retrieves or specifies the padding of borders of `Login` control
`CheckBoxStyle`	Retrieves the reference of the `Style` object to define the display settings of the Remember Me check box
`CreateUserIconUrl`	Retrieves the location of an image to display the link for new users for registration
`CreateUserText`	Retrieves or specifies the text of a link to a registration page for new users
`CreateUserUrl`	Retrieves or specifies the URL of the new user registration page
`DestinationPageUrl`	Retrieves or specifies the URL of the page displayed to the user when a login attempt is successful
`DisplayRememberMe`	Retrieves or specifies the value that indicates whether to display the Remember Me check box or not
`FailureAction`	Retrieves or specifies the action that occurs when a login attempt fails
`FailureText`	Retrieves or specifies the text displayed when a login attempt fails

Table 22.2: Noteworthy Properties of the Login Class

Property	Description
FailureTextStyle	Retrieves the reference of style properties to define the look of the failure text
HelpPageIconUrl	Retrieves the location of an image to display the link to the login help page
HelpPageText	Retrieves or specifies the text of a link to the login help page
HelpPageUrl	Retrieves or specifies the URL of the login help page
HyperLinkStyle	Retrieves a reference to a collection of properties that define the look of hyperlinks in Login control
InstructionText	Retrieves or specifies the login instruction text for user
InstructionTextStyle	Retrieves a reference of the TableItemStyle object to define the style of the instruction text
LayoutTemplate	Retrieves or specifies the template to display Login control
LoginButtonImageUrl	Retrieves or specifies the URL of an image to use as the Login button
LoginButtonText	Retrieves or specifies the text for the Login control's login button
Orientation	Retrieves or specifies the value to specify the position of elements of Login control on the page
Password	Retrieves the password entered by the user
PasswordLabelText	Retrieves or specifies the text of the label for the Password text box
PasswordRecoveryText	Retrieves or specifies the text of a link of the password recovery page
PasswordRecoveryUrl	Retrieves or specifies the URL of the password recovery page
PasswordRequiredErrorMessage	Retrieves or specifies the error message that is displayed when the password field is left blank, this error message is also displayed in ValidationSummary control
RememberMeText	Retrieves or specifies the text of the label for the Remember Me check box
RememberMeSet	Retrieves or specifies a value indicating whether to send a persistent authentication cookie to the user's browser
TitleText	Retrieves or specifies the title of Login control
UserName	Retrieves the username entered by user
UserNameLabelText	Retrieves or specifies the text of the label for the User Name text box

Noteworthy events of the Login class are listed in Table 22.3:

Table 22.3: Noteworthy Events of the Login class

Event	Description
Authenticate	Initiated when a user is authenticated
LoggedIn	Initiated when users log into the website and have been authenticated
LoggingIn	Initiated when a user submits login information, before authentication takes place
LoginError	Initiated when a login error is detected

login controls are made up of a combination of standard .NET controls, which are as follows:

- **Username Label and TextBox**—Gathers the string, which is used to authenticate a user during the logging process.
- **Password Label and TextBox**—Gathers the password as specified by the user. The `Textbox` control is always encrypted to hide the user password from getting displayed in the plain text.
- **LogIn button**—Submits the user's request for authentication.
- **Remember Me check box**—Enables users to store their credentials as a persistent cookie on the user's machine by selecting the displayed CheckBox control. It helps the user to enter the credentials (username and password) the next time the user wants to log on.
- **Title and Instruction**—Displays a text usually to guide users while accessing `login controls`. It basically provides information about the controls displayed below it.
- **Validators**—Display the required field validators for validation of the user credentials.

Figure 22.1 displays login controls at design time:

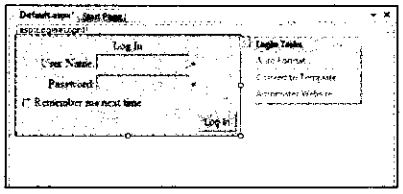


Figure 22.1: Displaying Login Controls

After discussing the `login controls`, let's move on to discuss the `LoginView` control.

The **LoginView** Control

The **`LoginView`** control is a Web server control, which is used to display two different views of a Web page of any website, depending on whether the user has logged on to a Web page as a registered user or a visitor. The `LoginView` control provides a way of altering the look of the page or showing different content to different groups of users. This control has the built-in functionality to gather the current user's status and roles. If the user is authenticated, the control displays the appropriate information to the user with the help of its three view templates, which are:

- **AnonymousTemplate**—Displayed when the user is not logged in
- **LoggedInTemplate**—Displayed when the user is logged in
- **RoleGroups**—Displayed when the user who has logged in, is a member of a specific role with defined role-group templates

The `LoginView` control is provided by the `LoginView` class. The class hierarchy of the `LoginView` class is as follows:

```
System.Object
  System.Web.UI.Control
        System.Web.UI.WebControls.LoginView
```

The `LoginView` class contains various methods, properties, and events used to work with the `LoginView` control. Noteworthy methods of the `LoginView` class are listed in Table 22.4:

Table 22.4: Noteworthy Methods of the LoginView Class

Method	Description
`DataBind`	Binds a data source to the `LoginView` control and all its child controls
`Focus`	Specifies input focus to the `LoginView` control
`OnViewChanged`	Raises the `ViewChanged` event after the `LoginView` control changes its view
`OnViewChanging`	Raises the `ViewChanging` event before the `LoginView` control changes its view

Noteworthy properties of the `LoginView` class are listed in Table 22.5:

Table 22.5: Noteworthy Properties of the LoginView Class

Property	Description
`AnonymousTemplate`	Retrieves or specifies the template to display to users who are not logged into the website
`Controls`	Retrieves the `ControlCollection` object that contains the child controls for the `LoginView` control
`EnableTheming`	Retrieves or specifies a value indicating whether themes can be applied to the `LoginView` control
`LoggedInTemplate`	Retrieves or specifies the template to display to website users who are logged into the website but are not members of one of the role groups specified in the `RoleGroups` property
`RoleGroups`	Retrieves a collection of role groups that associates content templates with particular roles
`SkinID`	Retrieves or specifies the skin to apply to the `LoginView` control

Noteworthy events of the `LoginView` class are listed in Table 22.6:

Table 22.6: Noteworthy Events of the LoginView Class

Event	Description
`ViewChanged`	Initiated after the view is changed
`ViewChanging`	Initiated before when the view is in process to change

The `LoginView` control displays the AnonymousTemplate view by default; however, we can even design a custom layout for the `LoginView` control as per the requirement. We can do so by adding any server control (such as `Label` and `TextBox`) to the empty region displayed on the `LoginView` control. You can simply drag-and-drop the `LoginView` control from the toolbox to add it on a Web page. Figure 22.2 displays the `LoginView` control at design time:

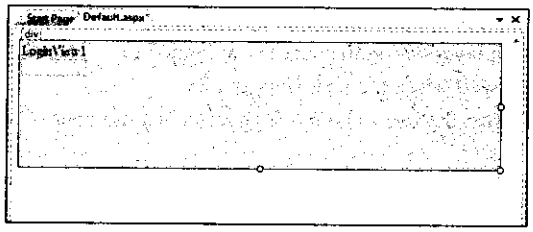


Figure 22.2: Displaying the LoginView Control

After discussing the `LoginView` control, let's move on to discuss the `LoginStatus` control.

The **LoginStatus** Control

The `LoginStatus` control specifies whether or not a particular user has logged on to the website. The text displayed on this control changes according to the login status of the user. When the user is not logged in, it displays the Login text as a hyperlink that facilitates the user in navigating to the login page. For this, the `LoginStatus` control use the authentication section of the `web.config` file and retrieves the URL of the login page. The `LoginStatus` control displays the Logout text as hyperlink when user is logged on to the website. As soon as, the user clicks the Logout link, the `LoginStatus` control deletes the authentication information. The `LoginStatus` control provides the following two views:

- **Logged Out**—Displayed when the user is not logged in. In this view, it provides a link to the login page.
- **Logged In**—Displayed when the user is logged in. In this view, it provides a link to logout of the website.

The `LoginStatus` control is provided by the `LoginStatus` class. The class hierarchy of the `LoginStatus` class is as follows:

```
System.Object
  System.Web.UI.Control
    System.Web.UI.WebControls.WebControl
      System.Web.UI.WebControls.CompositeControl
        System.Web.UI.WebControls.LoginStatus
```

The `LoginStatus` class contains various methods, properties, and events used to work with the `LoginStatus` control. Noteworthy methods of the `LoginStatus` class are listed in Table 22.7:

Table 22.7: Noteworthy Methods of the LoginStatus Class

Method	Description
`CreateChildControls`	Creates the child controls to make the `LoginStatus` control
`OnLoggedOut`	Raises the `LoggedOut` event when the user clicks the logout link
`OnLoggingOut`	Raises the `OnLoggingIn` event when the user clicks the logout link on the `LoginStatus` control

Noteworthy properties of the `LoginStatus` class are listed in Table 22.8:

Table 22.8: Noteworthy Properties of LoginStatus Class

Property	Description
`LoginImageUrl`	Retrieves or specifies the URL of the image used for the login link
`LoginText`	Retrieves or specifies the text used for the login link
`LogoutAction`	Retrieves or specifies a value to determine the action, when a user logs out of a website with the `LoginStatus` control
`LogoutImageUrl`	Retrieves or specifies the URL of the image used for the logout button
`LogoutPageUrl`	Retrieves or specifies the URL of the logout page
`LogoutText`	Retrieves or specifies the text used for the logout link

Noteworthy events of the `LoginStatus` class are listed in Table 22.9:

Table 22.9: Noteworthy Events of the LoginStatus Class

Event	Description
`LoggingOut`	Initiated after the user has sent a logout request to the server by clicking the logout link or button of the Login control
`LoggedOut`	Initiated by the `LoginStatus` class when the user logout process completes

You can simply drag-and-drop the `LoginStatus` control from the toolbox to add it to a Web page. Figure 22.3 displays the `LoginStatus` control at design time:.

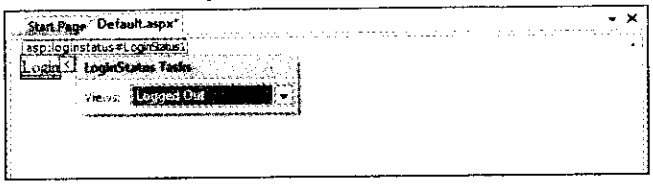


Figure 22.3: Displaying the LoginStatus Control

After discussing the `LoginStatus` control, let's move on to the `LoginName` control.

The LoginName Control

The LoginName control is responsible for displaying the name of the presently authenticated users. It uses Page.User.Identity.Name to return the value for the user's name. If there is no user logged in, then this control is not displayed on the page. This control is provided by the LoginName class. The class hierarchy of the LoginName class is as follows:

```
System.Object
  System.Web.UI.Control
    System.Web.UI.WebControls.WebControl
      System.Web.UI.WebControls.LoginName
```

The LoginName class does not contain any method, property or event that is not inherited from any other class, such as the Control class. The only non-inherited property it contains is FormatString that is used to display format item string in the LoginName control. You can simply drag-and-drop this control from the toolbox to add it on a Web page. Figure 22.4 displays the LoginName control at design time:

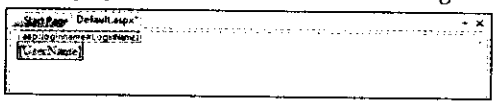


Figure 22.4: Displaying the LoginName Control

After discussing the LoginName control, let's move on to discuss the PasswordRecovery control.

The PasswordRecovery Control

The PasswordRecovery control is used to recover or reset the forgotten password of a user. This control does not display the password on the browser, but sends the password as an e-mail message to the e-mail address specified at the time of registration. This control also uses the Membership service to create or reset the password. The PasswordRecovery control has the following three views:

- **UserName**—In this view, the user is asked to enter its username for its password to be recovered.
- **Question**—In this view, the user is asked to enter the answer for its security question.
- **Success**—In this view, a message is displayed to the user, which specifies that the retrieved password has been sent to the user.

To retrieve or reset passwords, you must set some properties of the ASP.NET Membership service. The PasswordRecovery control uses the following configuration of the Membership service settings for retrieving passwords:

- requiresQuestionAndAnswer—If this property is set to True, the user is required to answer a security question, which was specified at the time of creating the user account through ASP.NET configuration services.
- passwordFormat—If this property is set to 'Hashed', the password is in encrypted format. Therefore, this control will not be able to retrieve the password, but just reset the password.
- enablePasswordRetrieval—If this property is set to True, only then the Membership service allows any Web control to retrieve the password.
- enablePasswordReset—If this property is set to True, only then users are allow to reset their passwords.

The PasswordRecovery control is provided by the PasswordRecovery class. The class hierarchy of the PasswordRecovery class is as follows:

```
System.Object
  System.Web.UI.Control
    System.Web.UI.WebControls.WebControl
      System.Web.UI.WebControls.CompositeControl
        System.Web.UI.WebControls.PasswordRecovery
```

The PasswordRecovery class contains various methods, properties and events used to work with the PasswordRecovery control. Noteworthy methods of the PasswordRecovery class are listed in Table 22.10:

Table 22.10: Noteworthy Methods of the PasswordRecovery Class

Method	Description
`OnAnswerLookupError`	Raises the `AnswerLookupError` event when users' answers do not match with the answer stored in the website database
`OnSendingMail`	Raises the `OnSendingMail` event when the user is verified and the recovered password is sent to the user
`OnSendingMailError`	Raises the `OnsendingMailError` event when an e-mail message cannot be sent to the user
`OnUserLookupError`	Raises the `OnUserLookupError` event when the entered username does not match with any username stored in the website database
`OnVerifyingAnswer`	Raises the `OnVerifyingAnswer` event before the users' answers are sent to the Membership service provider to recover the forgotten password
`OnVerifyingUser`	Raises the `OnVerifyingUser` event after submitting the username but before sending it to the Membership service provider for verification

Noteworthy properties of the `PasswordRecovery` class are listed in Table 22.11:

Table 22.11: Noteworthy Properties of the PasswordRecovery Class

Property	Description
`Answer`	Retrieves the answer provided by the user to confirm the password recovery by the valid user
`AnswerLabelText`	Retrieves or specifies the label text for the password confirmation answer text box
`AnswerRequiredErrorMessage`	Specifies a custom error message for the `PasswordRecovery` control, in case the user leaves the answer field blank
`BorderPadding`	Retrieves or specifies the padding width inside the borders of the `PasswordRecovery` control
`FailureTextStyle`	Retrieves a reference to a collection of properties to define the look of the error text
`GeneralFailureText`	Retrieves or specifies the error message to display when there is a problem with the Membership service provider for the `PasswordRecovery` control
`HelpPageIconUrl`	Retrieves the location of an image to display the link to the password recovery help page
`HelpPageText`	Retrieves or specifies the text of a link to the password recovery help page
`HelpPageUrl`	Retrieves or specifies the URL of the password recovery help page
`HyperLinkStyle`	Retrieves a reference to a collection of properties that define the look of hyperlinks in the `PasswordRecovery` control
`InstructionTextStyle`	Retrieves a reference of the `style` object to define the look of the explanatory text in the `PasswordRecovery` control
`LabelStyle`	Retrieves a collection of the `style` object to define the look of the `TextBox` control's labels in the `PasswordRecovery` control
`MailDefinition`	Retrieves the characteristics of new e-mail messages sent to the user to recover the password
`MembershipProvider`	Used to get the reference of the Membership service provider, which is used to gather user's identity
`Question`	Retrieves the password recovery question, which was provided by the user at the time of registration

Table 22.11: Noteworthy Properties of the PasswordRecovery Class

Property	Description
QuestionFailureText	Specifies a custom message for the PasswordRecovery control to be displayed when the user provides an incorrect answer
QuestionTemplate	Retrieves or specifies the template used to set the look of Question view of the PasswordRecovery control
QuestionTitleText	Retrieves or specifies the title of the PasswordRecovery control when the control is in the Question view
SubmitButtonImageUrl	Retrieves or specifies the image path that is used as the Button control
SubmitButtonStyle	Retrieves the reference of style properties that are used to define the look of the Button control
SubmitButtonText	Retrieves or specifies the text to be displayed on the Button control
SubmitButtonType	Retrieves or specifies the type of Button control that is used to render the PasswordRecovery control
SuccessPageUrl	Retrieves or specifies the URL of the page to be displayed if the forgotten password is sent to the user's e-mail id successfully
SuccessTemplate	Retrieves or specifies the template for the PasswordRecovery control when it is in the Success view
SuccessText	Specifies a custom message for the PasswordRecovery control to be displayed in case the task of password recovery is accomplished successfully
SuccessTextStyle	Retrieves or specifies the style in which the Success text is to be displayed
UserName	Retrieves or specifies the text to be displayed in the User Name text box
UserNameFailureText	Retrieves or specifies the text when a user enters an invalid username
UserNameInstructionText	Retrieves or specifies the text in the PasswordRecovery control when it is in the UserName view
UserNameLabelText	Retrieves or specifies the label for the User Name text box
UserNameRequiredErrorMessage	Retrieves or specifies the error message that is displayed when a user does not enter any value in the User Name text box
UserNameTemplate	Retrieves or specifies the template used to display the UserName view of the PasswordRecovery control
UserNameTitleText	Retrieves or specifies the title of the PasswordRecovery control when it is in the UserName view
ValidatorTextStyle	Retrieves or specifies the style properties that are used to define the look of the error message of the Validator control

Noteworthy events of the PasswordRecovery class are listed in Table 22.12:

Table 22.12: Noteworthy Events of the PasswordRecovery Class

Event	Description
AnswerLookupError	Initiated when the user's answer to the question is found incorrect
SendingMail	Initiated when the server is sending an e-mail containing the password after the user's answer is found correct
SendMailError	Initiated when the SMTP Mail system throws an error while attempting to send an e-mail message

Table 22.12: Noteworthy Events of the PasswordRecovery Class	
Event	**Description**
`UserLookupError`	Initiated when the Membership provider cannot find the username entered by the user
`VerifyingAnswer`	Initiated when the user has submitted the answer to the password recovery confirmation question
`VerifyingUser`	Initiated when the validation of username by the Membership provider is under process

You can simply drag-and-drop this control from the toolbox to add it to a Web page. Figure 22.5 displays the `PasswordRecovery` control at design time:

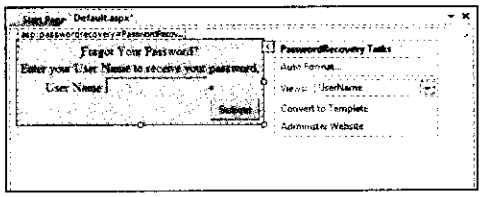


Figure 22.5: Displaying the PasswordRecovery Control

After discussing the `PasswordRecovery` control, let's move on to discuss the `CreateUserWizard` control.

The **CreateUserWizard** Control

The `CreateUserWizard` control uses the Membership service to create a new user in the Membership data store. This control is an extended version of the already existing `Wizard` control. Being an extended control, the `CreateUserWizard` control can be customized by using templates and style properties. The `CreateUserWizard` control is provided by the `CreateUserWizard` class. The class hierarchy of the `CreateUserWizard` class is as follows:

```
System.Object
    System.Web.UI.Control
        System.Web.UI.WebControls.WebControl
            System.Web.UI.WebControls.CompositeControl
                System.Web.UI.WebControls.Wizard
                    System.Web.UI.WebControls.CreateUserWizard
```

The `CreateUserWizard` class contains various methods, properties, and events. However, the methods of the `CreateUserWizard` class are mostly inherited and the other non-inherited methods are not of much use. Noteworthy properties of the `CreateUserWizard` class are listed in Table 22.13:

Table 22.13: Noteworthy Properties of the CreateUserWizard Class	
Property	**Description**
`ActiveStepIndex`	Retrieves or specifies the step that is currently displayed to the user. This is an overridden property.
`Answer`	Retrieves or specifies the end user's answer to the password recovery confirmation question.
`AnswerLabelText`	Retrieves or specifies the text of the label of the password confirmation answer text box.
`AnswerRequiredErrorMessage`	Retrieves or specifies the error message shown when a user leaves the password confirmation question unanswered.
`AutoGeneratePassword`	Retrieves or specifies a value indicating whether or not to automatically generate a password for the new user account.

Table 22.13: Noteworthy Properties of the CreateUserWizard Class

Property	Description
CompleteStep	Defines the final step of the process of creating the user.
CompleteSuccessText	Retrieves or specifies the text displayed when a website user account is created successfully.
CompleteSuccessTextStyle	Retrieves a reference to a collection of properties that define the look of the text displayed when a website user account is created successfully.
ConfirmPassword	Retrieves the second password entered by the user.
ConfirmPasswordCompareErrorMessage	Retrieves or specifies the error message to be displayed when the passwords in the password text box and the confirm password do not match.
ConfirmPasswordLabelText	Retrieves or specifies the text of the label for the second password text box.
ConfirmPasswordRequiredErrorMessage	Retrieves or specifies the error message displayed when the user leaves the confirm password text box empty.
ContinueButtonImageUrl	Retrieves or specifies the URL of an image to display the continue button. The continue button is displayed as the final step of the process of creating user account.
ContinueButtonStyle	Retrieves a reference to a collection of properties that define the look of the continue button.
ContinueButtonText	Retrieves or specifies the text caption displayed on the Continue button.
ContinueButtonType	Retrieves or specifies the type of button rendered as the Continue button.
ContinueDestinationPageUrl	Retrieves or specifies the URL of the page that the user will see after clicking the Continue button on the success page.
CreateUserButtonImageUrl	Retrieves or specifies the URL of an image displayed for the Create User button.
CreateUserButtonStyle	Retrieves a reference to a collection of properties that define the look of the Create User button.
CreateUserButtonText	Retrieves or specifies the text caption displayed on the Create User button.
CreateUserButtonType	Retrieves or specifies the type of button rendered as the Create User button.
CreateUserStep	Retrieves a reference to the template, which will be used as a step in the process of creating the user account.
DisableCreatedUser	Retrieves or specifies a value indicating whether the new user should be allowed to log on to the website.
DisplaySideBar	Retrieves or specifies a value indicating whether to display the sidebar area of the control. This is an overridden property.
DuplicateEmailErrorMessage	Retrieves or specifies the error message displayed when the user enters an e-mail address that is already in use in the Membership service provider.
DuplicateUserNameErrorMessage	Retrieves or specifies the error message displayed when the user enters a username that is already in use in the Membership service provider.
EditProfileIconUrl	Retrieves or specifies the URL of an image to display next to the link to the user profile editing page.
EditProfileText	Retrieves or specifies the text caption for the link to the user profile editing page.
EditProfileUrl	Retrieves or specifies the URL of the user profile editing page.
Email	Retrieves or specifies the e-mail address entered by the user.
EmailLabelText	Retrieves or specifies the text of the label for the e-mail text box.

Table 22.13: Noteworthy Properties of the CreateUserWizard Class

Property	Description
EmailRegularExpression	Retrieves or specifies a regular expression used to validate the e-mail address.
EmailRegularExpressionErrorMessage	Retrieves or specifies the error message displayed when the entered e-mail address does not pass the site's criteria for e-mail addresses.
EmailRequiredErrorMessage	Retrieves or specifies the error message shown to the user when an e-mail address is not entered in the e-mail text box.
ErrorMessageStyle	Retrieves a reference to a collection of style properties that define the look of the error messages.
HelpPageIconUrl	Retrieves or specifies the URL of an image to display next to the link to the help page.
HelpPageText	Retrieves or specifies the text for the link to the help page.
HelpPageUrl	Retrieves or specifies the URL of the help page.
HyperLinkStyle	Retrieves or specifies a collection of properties that define the look of hyperlinks.
InstructionText	Retrieves or specifies help text to create a new user account.
InstructionTextStyle	Retrieves a reference to a collection of properties that define the look of the instruction text.
InvalidAnswerErrorMessage	Retrieves or specifies the message displayed when the password retrieval answer is not valid.
InvalidEmailErrorMessage	Retrieves or specifies the message displayed when the entered e-mail address is not valid.
InvalidPasswordErrorMessage	Retrieves or specifies the message displayed when the password entered is not valid.
InvalidQuestionErrorMessage	Retrieves or specifies the message displayed when the password retrieval question is not valid.
LabelStyle	Retrieves a reference to a collection of properties that define the look of labels.
LoginCreatedUser	Retrieves or specifies a value indicating whether the new user can login, after creating the user account.
MailDefinition	Retrieves a reference to a collection of properties that define the characteristics of the e-mail message sent to new users.
MembershipProvider	Retrieves or specifies the Membership service provider called to create user accounts.
Password	Retrieves the password entered by the user.
PasswordHintStyle	Retrieves a reference to a collection of properties that define the look of the text that describes password requirements.
PasswordHintText	Retrieves or specifies the text that describes password requirements.
PasswordLabelText	Retrieves or specifies the text of the label for the password text box.
PasswordRegularExpression	Retrieves or specifies a regular expression used to validate the provided password.
PasswordRegularExpressionErrorMessage	Retrieves or specifies the error message shown when the password entered does not conform to the specific pattern.
PasswordRequiredErrorMessage	Retrieves or specifies the text of the error message shown when the user does not enter a password.

Table 22.13: Noteworthy Properties of the CreateUserWizard Class	
Property	Description
Question	Retrieves or specifies the password recovery confirmation question entered by the user.
QuestionLabelText	Retrieves or specifies the text of the label for the question text box.
QuestionRequiredErrorMessage	Retrieves or specifies the error message that is displayed when the user does not enter a password confirmation question.
RequireEmail	Retrieves or specifies a value indicating whether an e-mail address is required for the website user.
SkipLinkText	Retrieves or specifies a value that is used to provide alternate text that notifies screen readers to skip the sidebar area's content. This is an overridden property
TextBoxStyle	Retrieves a reference of style properties that define the look of TextBox controls.
TitleTextStyle	Retrieves a reference of style properties that define the look of titles.
UnknownErrorMessage	Retrieves or specifies the error message displayed when an error returned by the Membership provider is not defined.
UserName	Retrieves or specifies the username entered by the user.
UserNameLabelText	Retrieves or specifies the text of the label for the User Name text box.
UserNameRequiredErrorMessage	Retrieves or specifies the error message displayed when the User Name text box is left blank.
ValidatorTextStyle	Retrieves a reference to the style object that allows you to set the look of the validation error messages.
WizardSteps	Retrieves a reference to a collection containing all the `WizardStepBase` objects defined for the control. This is an overridden property.

Noteworthy events of the `CreateUserWizard` class are listed in Table 22.14:

Table 22.14: Noteworthy Events of the CreateUserWizard Class	
Event	Description
ContinueButtonClick	Initiated when the user clicks the Continue button. The Continue button is displayed as the final step of the user account creation process.
CreatedUser	Initiated after the Membership service provider has created the new website user account.
CreateUserError	Initiated when the Membership service provider throws error while creating a user account.
CreatingUser	Initiated before the Membership service provider is called to create the user account.
SendingMail	Initiated before sending a confirmation e-mail on successful creation of a user.
SendMailError	Initiated when there is an SMTP error while sending an e-mail to the new user.

The `CreateUserWizard` control has a default collection of steps to gather user information. However, you can also add additional steps to the Wizard in order to collect additional user information. The following are some of the important components used to create steps of the `CreateUserWizard` control:

- `CreateUserWizardStep`—This is a pre-defined step that contains the user interface and the logic for creating a new user.

- `CompleteWizardStep`—This is a pre-defined step that displays the information whenever a user account is created successfully.
- Collection of `WizardSteps`—This is a collection of all the user-defined steps that are displayed one by one during the process of creating a user. Though, there can be multiple steps, only one step is displayed at a given time.
- `Navigation Buttons`—These buttons are present in the navigation area below each `WizardStep`. The navigation buttons allow users to go to the next and the previous steps in the `CreateUserWizard` control.
- `SideBar`—This is an optional component. It is responsible for listing all the `WizardSteps` steps and, therefore, enabling users to randomly access any of the `WizardSteps` steps.
- `Header`—This is also an optional component. It provides users with the information regarding each step. It is present at the top of the wizard.

You can simply drag-and-drop this control from the toolbox to add it on a Web page. Figure 22.6 displays the `CreateUserWizard` control at design time:

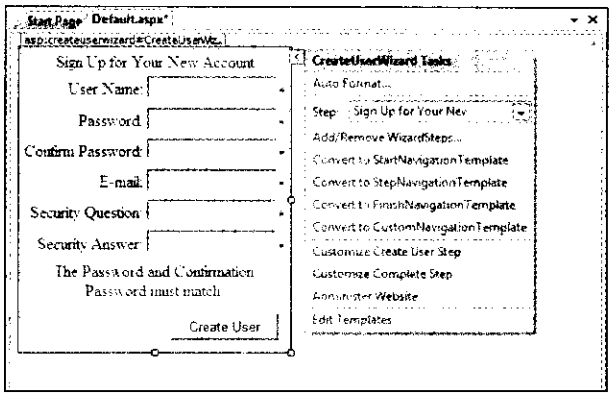


Figure 22.6: Displaying the CreateUserWizard Control

After discussing the `CreateUserWizard` control, let's move on to discuss the `ChangePassword` control.

The **ChangePassword** Control

The `ChangePassword` control allows the users to change their password. This control prompts users to provide the current password first and then set the new password. If the old password is not correct, the new password cannot be set. This control also facilitates in sending e-mails to users about the new password. The `ChangePassword` control is provided by the `ChangePassword` class. The class hierarchy of the `ChangePassword` class is as follows:

```
System.Object
    System.Web.UI.Control
        System.Web.UI.WebControls.WebControl
            System.Web.UI.WebControls.CompositeControl
                System.Web.UI.WebControls.ChangePassword
```

The `ChangePassword` class contains various methods, properties, and events. However, the methods of the `ChangePassword` class are mostly inherited and the other non-inherited methods are not very useful. Noteworthy properties of the `ChangePassword` class are listed in Table 22.15:

Table 22.15: Noteworthy Properties of the ChangePassword Class

Property	Description
`BorderPadding`	Retrieves or specifies the amount of padding, in pixels, inside the border and the designated area for the `ChangePassword` control.
`CancelButtonImageUrl`	Retrieves or specifies the URL of an image to display with the Cancel button, if the

Table 22.15: Noteworthy Properties of the ChangePassword Class

Property	Description
	Cancel button is configured by the `CancelButtonType` property to be an image button.
`CancelButtonStyle`	Retrieves a reference of style properties to define the look of the Cancel button on the `ChangePassword` control.
`CancelButtonText`	Retrieves or specifies the text displayed on the Cancel button.
`CancelButtonType`	Retrieves or specifies the type of button to use for the Cancel button when customizing the `ChangePassword` control.
`CancelDestinationPageUrl`	Retrieves or specifies the URL of the page that the user is shown after clicking the Cancel button in the `ChangePassword` control.
`ChangePasswordButtonImageUrl`	Retrieves or specifies the URL of an image displayed next to the Change Password button on the `ChangePassword` control if the Change Password button is configured through `ChangePasswordButtonType` property to be an image button.
`ChangePasswordButtonStyle`	Retrieves a reference to a collection of style properties to define the look of the Change Password button.
`ChangePasswordButtonText`	Retrieves or specifies the text displayed on the Change Password button.
`ChangePasswordButtonType`	Retrieves or specifies the type of button to use when rendering the Change Password button.
`ChangePasswordFailureText`	Retrieves or specifies the message that is shown when the user's password is not changed.
`ChangePasswordTemplate`	Retrieves or specifies the `ITemplate` object used to display the Change Password view of the `ChangePassword` control.
`ChangePasswordTemplate Container`	Retrieves the container that a `ChangePassword` control uses to create an instance of the `ChangePasswordTemplate` template. This provides programmatic access to child controls.
`ChangePasswordTitleText`	Retrieves or specifies the text displayed at the top of the `ChangePassword` control in the Change Password view.
`ConfirmNewPassword`	Retrieves the value of Confirm new password text box.
`ConfirmNewPasswordLabelText`	Retrieves or specifies the label text for the `ConfirmNewPassword` text box.
`ConfirmPasswordCompareError Message`	Retrieves or specifies the message that is displayed when the new password and the duplicate password are not identical.
`ConfirmPasswordRequiredError Message`	Retrieves or specifies the error message that is displayed when the Confirm New Password text box is left blank.
`ContinueButtonImageUrl`	Retrieves or specifies the URL of an image to use for the Continue button on the Success view of the `ChangePassword` control if the Continue button is configured by the `ContinueButtonType` property to be an image button.
`ContinueButtonStyle`	Retrieves a reference of style properties to define the look of Continue button on the Success view of the `ChangePassword` control.
`ContinueButtonText`	Retrieves or specifies the text of Continue button in the Success view of the `ChangePassword` control.
`ContinueButtonType`	Retrieves or specifies the type of button to use when rendering the Continue button for the `ChangePassword` control.
`ContinueDestinationPageUrl`	Retrieves or specifies the URL of the page that the user will see after clicking the

Table 22.15: Noteworthy Properties of the ChangePassword Class

	Continue button on the Success view.
`CreateUserIconUrl`	Retrieves or specifies the URL of an image to display next to the link of the Web page that contains a `CreateUserWizard` control for the website.
`CreateUserText`	Retrieves or specifies the text of the link to the Web page that contains a `CreateUserWizard` control for the website.
`CreateUserUrl`	Retrieves or specifies the URL of the Web page that contains a `CreateUserWizard` control for the website.
`CurrentPassword`	Retrieves the current password of the user.
`DisplayUserName`	Retrieves or specifies a value indicating whether the `ChangePassword` control should display the `UserName` control and label.
`EditProfileIconUrl`	Retrieves or specifies the URL of an image to display next to the link to the user profile editing page.
`EditProfileText`	Retrieves or specifies the text of the link to the user profile editing page.
`EditProfileUrl`	Retrieves or specifies the URL of the user profile editing page.
`FailureTextStyle`	Retrieves a reference of style properties to define the look of error messages on the `ChangePassword` control.
`HelpPageIconUrl`	Retrieves or specifies the URL of an image to display next to the Change Password help page.
`HelpPageText`	Retrieves or specifies the link text to the Change Password help page.
`HelpPageUrl`	Retrieves or specifies the URL of the Change Password help page for the website.
`HyperLinkStyle`	Retrieves a reference of style properties that define the look of hyperlinks on the `ChangePassword` control.
`InstructionText`	Retrieves or specifies informational text that appears on the `ChangePassword` control between the `ChangePasswordTitleText` and the input boxes.
`InstructionTextStyle`	Retrieves a reference of style properties that define the look of the instructional text on the `ChangePassword` control.
`LabelStyle`	Retrieves a reference of style objects that define the look of text box labels on the `ChangePassword` control.
`MailDefinition`	Retrieves a reference to a collection of properties that define the e-mail message that is sent to the users after they have changed their password.
`MembershipProvider`	Retrieves or specifies the Membership provider that is used to manage member information.
`NewPassword`	Retrieves the new password entered by the user.
`NewPasswordLabelText`	Retrieves or specifies the label text for the New Password text box.
`NewPasswordRegularExpression`	Retrieves or specifies the regular expression that is used to validate the password provided by users.
`NewPasswordRegularExpression ErrorMessage`	Retrieves or specifies the error message that is displayed when the password entered does not match with the regular expression defined in the `NewPasswordRegularExpression` property.
`NewPasswordRequiredError Message`	Retrieves or specifies the error message that is displayed when the user leaves the New Password text box empty.
`PasswordHintStyle`	Retrieves a reference of style properties that define the appearance of

Table 22.15: Noteworthy Properties of the ChangePassword Class	
	informational text that appears on the `ChangePassword` control.
`PasswordHintText`	Retrieves or specifies the informational text about the requirements for creating a password for the website.
`PasswordLabelText`	Retrieves or specifies the label text for the Current Password text box.
`PasswordRecoveryIconUrl`	Retrieves or specifies the URL of an image to display next to a link of the Web page that contains the `PasswordRecovery` control.
`PasswordRecoveryText`	Retrieves or specifies the text of the link of the Web page that contains the `PasswordRecovery` control.
`PasswordRecoveryUrl`	Retrieves or specifies the URL of the Web page that contains the `PasswordRecovery` control.
`PasswordRequiredErrorMessage`	Retrieves or specifies the error message that is displayed when users leave the Current Password text box empty.
`SuccessPageUrl`	Retrieves or specifies the URL of the page that is shown to users after successfully changing the password.
`SuccessTemplate`	Retrieves or specifies the `ITemplate` object that is used to display the Success and Change Password views of the `ChangePassword` control.
`SuccessTemplateContainer`	Retrieves the container that a `ChangePassword` control used to create an instance of the `SuccessTemplate` template. This provides programmatic access to child controls.
`SuccessText`	Retrieves or specifies the text that is displayed on the Success view between the `SuccessTitleText` and the Continue button.
`SuccessTextStyle`	Retrieves style properties to define the look of text on the Success view.
`SuccessTitleText`	Retrieves or specifies the title of the Success view.
`TextBoxStyle`	Retrieves a reference of style properties that define the look of TextBox controls on the `ChangePassword` control.
`TitleTextStyle`	Retrieves a reference of style properties that define the look of titles on the `ChangePassword` control.
`UserName`	Retrieves or specifies the username for which the password is to be changed.
`UserNameLabelText`	Retrieves or specifies the label for the User Name text box.
`UserNameRequiredErrorMessage`	Retrieves or specifies the error message that is displayed when the user leaves the User Name text box empty.
`ValidatorTextStyle`	Retrieves a reference of style properties that define the look of error messages that are associated with any input validation used by the `ChangePassword` control.

Noteworthy events of the `ChangePassword` class are listed in Table 22.16:

Table 22.16: Noteworthy Events of the ChangePassword Class	
`CancelButtonClick`	Initiated when the user clicks the Cancel button to cancel the changing of a password
`ChangedPassword`	Initiated when the password is changed for a user account
`ChangePasswordError`	Initiated when there is an error in changing the password for the user account
`ChangingPassword`	Initiated before the password for a user account is changed

Table 22.16: Noteworthy Events of the ChangePassword Class	
[redacted]	
ContinueButtonClick	Initiated when the user clicks the Continue button
SendingMail	Initiated before the user is sent an e-mail confirmation that the password has been changed
SendMailError	Initiated when there is an SMTP error sending an e-mail message to the user

You can simply drag-and-drop the `ChangePassword` control from the toolbox to add it on a Web page. Figure 22.7 displays the `ChangePassword` control at design time:

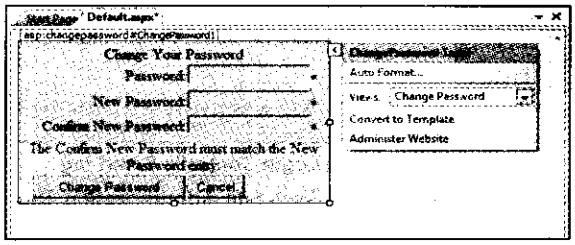


Figure 22.7: Displaying the ChangePassword Control

Now that you have become familiar with the login controls and their respective methods, properties, and events, let's move ahead and learn to implement some of these controls in an application.

Implementing Authentication in ASP.NET Using Login Controls

As we have already discussed, the login controls are a new set of controls that allow you to easily create a well-functioning login page for a website. We have created a simple website to depict the power of `login controls`. In this website, we have added two Web forms, which are login.aspx and welcome.aspx and deleted the Default.aspx page. The welcome.aspx is accessible when a user successfully logs on through the login.aspx page. You can find this code of `LoginControlExampleAppVB` application in the Code\ASP.NET\Chapter 22\`LoginControlExampleAppVB` folder on the CD. Let's now perform the following steps to learn about the implementation of authentication logic to validate users before allowing them to log on to the website:

1. Place a `Login` control on your form and change the `Auto Format option` to `Professional` to make it look similar to Figure 22.8:

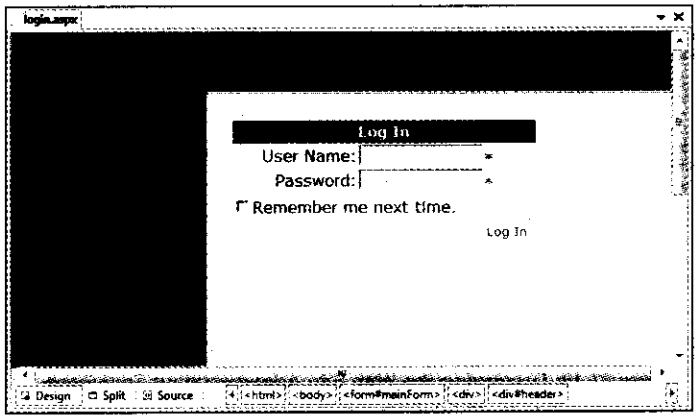


Figure 22.8: Showing a Login Control on a Web Form

2. Now, click the Show Smart Tag (▷) at the top right corner of the `Login` control to display its smart tag and select the Administer Website option, as shown in Figure 22.9:

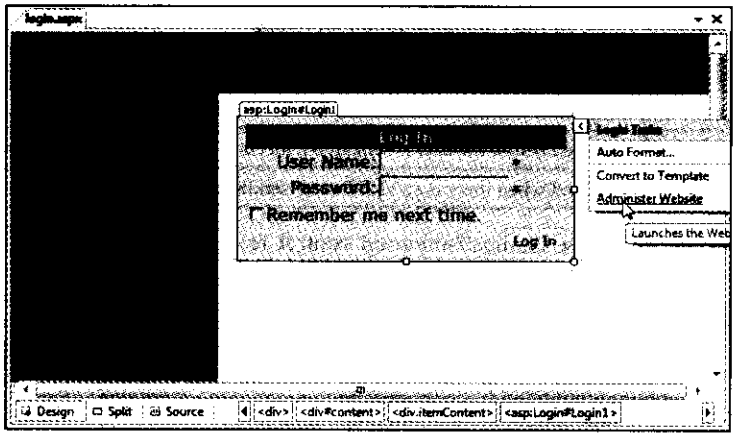


Figure 22.9: Using the Login Control's Smart tag

When you click the Show Smart Tag, the ASP.NET Web Site Administration Tool page opens. Now, there are several ways of adding user information in the database to authenticate and configure users and assign appropriate permissions, but we will select the wizard to do so, because it provides a step-by-step approach to authenticate and configure users.

3. Click the Security tab and then click the `Use the security Setup Wizard to configure security step by step` link, as shown in Figure 22.10:

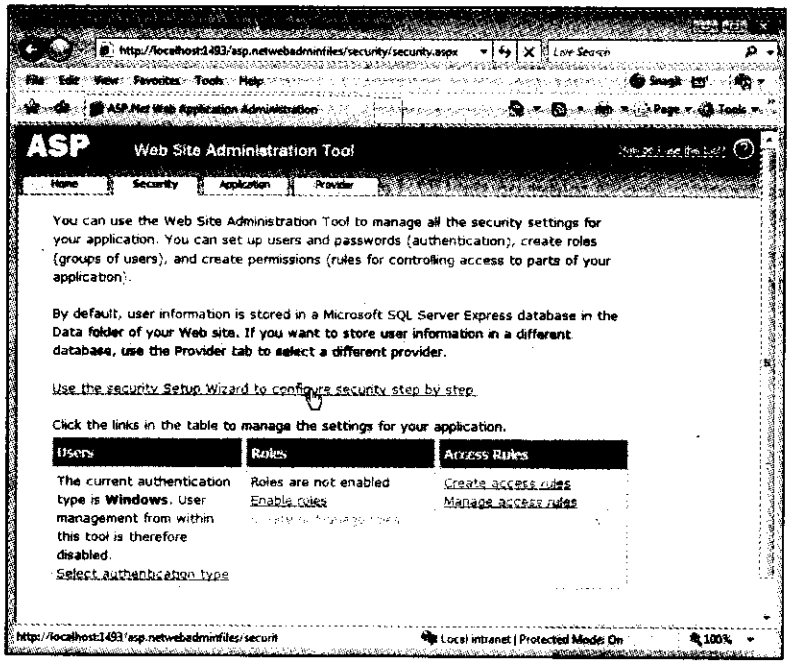


Figure 22.10: Displaying the Web Site Administration Tool Page

4. The Welcome page appears; click the Continue button.
5. We want our website to authenticate users who are trying to access it over the Internet. For this, select the `From the internet` radio button, as shown in Figure 22.11:

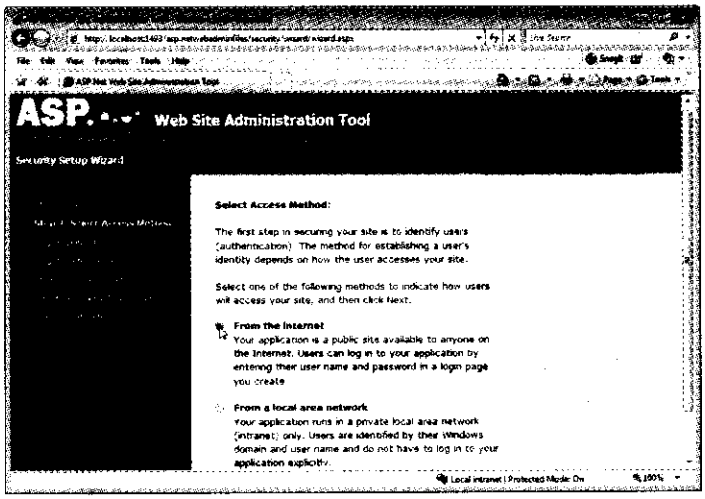


Figure 22.11: Selecting the Access Method of the Users

The next step is to register users with their respective credentials, including username and password. This information is asked for at the time of accessing the website. If the user enters the wrong username or password, an error message will be shown, and if the correct username and password are entered, then the user will be allowed to access the website.

6. Add any number of users you want. However, remember that the password of each user must have one numeric character, one special character, and the rest of the string must not be the username.
7. After entering all the details, click the `Create User` button. Figure 22.12 shows how to enter the user information:

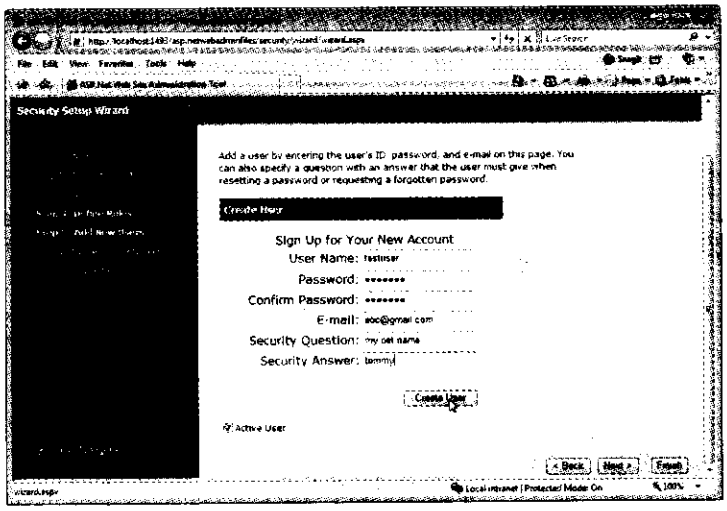


Figure 22.12: Adding Users for Authentication

Now, we define roles for the users we just created and then assign permissions to each role. You can assign permission to individual users as well.

8. Allow access to all the users by selecting the `All Users` and `Allow` radio buttons, and click the `Add This Rule` button, as shown in Figure 22.13:

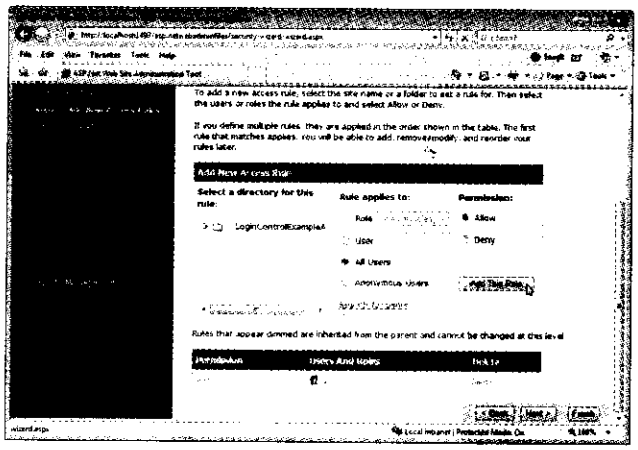


Figure 22.13: Specifying Permission for the Users

TIP

All the information regarding users and permissions that you create using the ASP.NET Web Site Administration Tool is stored in the database file called ASPNETDB.MDF, which you can find in the App_Data folder present under the application folder.

9. Now, click the Next button to complete the process of creating user and assigning permissions.

Now, let's discuss a hypothetical case here, where we are allowed to access a page, say, `welcome.aspx`, only if the authentication is successful, we added in the previous steps. This page (`welcome.aspx`) is indicative of the resources that you want to protect from unauthorized users. If you are working on IIS 7 instead of the built-in Web server, you have to disable the `Anonymous Authentication` mode and enable the `Windows Authentication` mode. For this, select IIS manager and then select `Authentication` from the IIS group. Select the Anonymous Authentication mode and click the Disable option from the Actions navigation bar. Enable the Forms Authentication and Windows Authentication modes in the same manner, as shown in Figure 22.14:

NOTE

When you enable the Windows Authentication mode after enabling the Forms Authentication mode, the IIS manger will show an error that the Forms Authentication and Windows Authentication modes cannot be enabled simultaneously. However, you should proceed further as Login control work for the Forms Authentication mode (Figure 22.14)

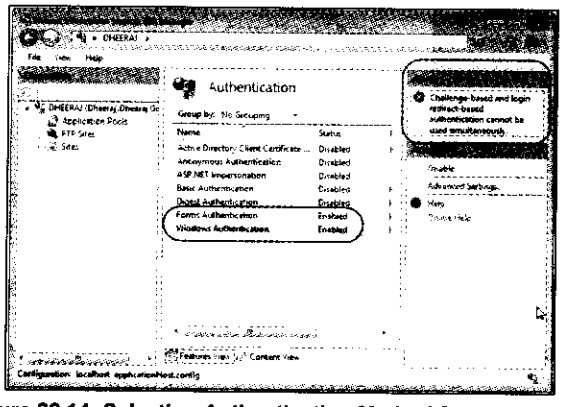


Figure 22.14: Selecting Authentication Method for the Website

10. Click the Close button to close the Internet Information Services (IIS) Manager window.

The next step is to create the page that can be accessed only by the authenticated users.

11. Go to the `login.aspx` page and set the `DestinationPageUrl` property of the `Login1` control to `welcome.aspx` page. You can find the code for the `login.aspx` page in the Listing 22.1:

Listing 22.1: Showing the complete code for the `login.aspx` page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="login.aspx.vb"
  Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>Login Control Example</title>
  <link href="Stylesheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="mainForm" runat="server">
  <div>
        <div id="header">
        </div>
        <div id="sidebar">
              <div id="nav">
                     
              </div>
        </div>
        <div id="content">
              <div class="itemContent">
              <br />
              <asp:Login ID="Login1" runat="server" BackColor="#F7F6F3"
              BorderColor="#E6E2D8" BorderPadding="4"
              BorderStyle="Solid" BorderWidth="1px" Font-Names="Verdana" Font-
              Size="Medium"
              ForeColor="#333333" Height="125px" Width="314px"
              DestinationPageUrl="~/welcome.aspx">
              <TitleTextStyle BackColor="#5D7B9D" Font-Bold="True" Font-Size="0.9em"
              ForeColor="White" />
              <InstructionTextStyle Font-Italic="True" ForeColor="Black" />
              <TextBoxStyle Font-Size="0.8em" />
              <LoginButtonStyle BackColor="#FFFBFF" BorderColor="#CCCCCC"
              BorderStyle="Solid" BorderWidth="1px"
              Font-Names="Verdana" Font-Size="0.8em" ForeColor="#284775" />
              </asp:Login>
               
              <br />
              </div>
              <div id="footer">
                    <p class="left">
                    All content copyright &copy; Knpcet Publications Inc.
              </p>
              </div>
        </div>
  </div>
  </form>
</body>
</html>
```

12. Now, navigate to the `welcome.aspx` page, select the `LoginStatus` control, and set the `LogoutAction` property to `RedirectToLoginPage` and the `LogoutPageUrl` property to the `login.aspx` page. You can find the code for the `welcome.aspx` page in the Listing 22.2:

Listing 22.2: Showing the complete code for the `welcome.aspx` page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="welcome.aspx.vb"
  Inherits="welcome" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>LoginControl Example</title>
    <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <form id="mainForm" runat="server">
    <div>
        <div id="header">

        </div>
        <div id="sidebar">
            <div id="nav">
                 
            </div>
        </div>
        <div id="content">
            <div class="itemContent">
            <br />
            <asp:Label ID="Label1" runat="server" Text="Welcome"></asp:Label> 
            <asp:LoginName ID="LoginName1" runat="server" FormatString="{0}!" />
            <br />
            <br />
            <asp:LoginStatus ID="LoginStatus1" runat="server"
            LogoutAction="RedirectToLoginPage"
            LogoutPageUrl="~/login.aspx" />
            <br />
            </div>
            <div id="footer">
                <p class="left">
                All content copyright &copy; Kogent Solutions Inc.</p>
            </div>
        </div>
    </div>
    </form>
</body>
</html>
```

13. Now, right-click the `login.aspx` in Solution Explorer and select `Set As Start Page` option from the context menu.
14. Finally, press the F5 key to run the application. The output is shown in Figure 22.15:

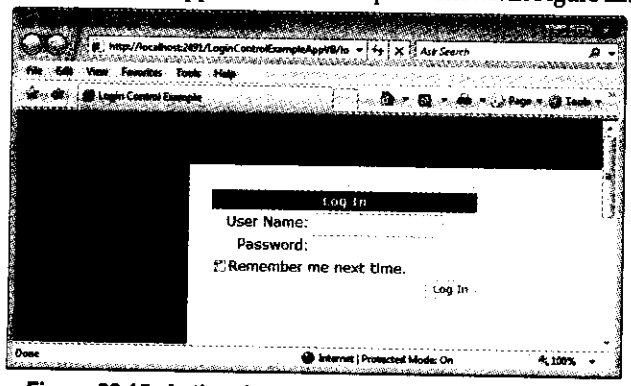


Figure 22.15: Authenticating Users Using Login Controls

15. Enter the username and password that you have specified while creating the user account and click the Log In button to log on; in our case, the username is testuser and password is abc@123.
16. After successful authentication, a page named `welcome.aspx` will appear, as shown in Figure 22.16:

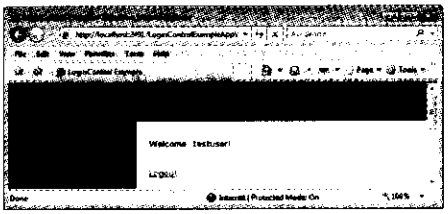


Figure 22.16: Showing a Successfully Authenticated User

However, if the login information is incorrect, then an error message will be displayed, as shown in Figure 22.17:

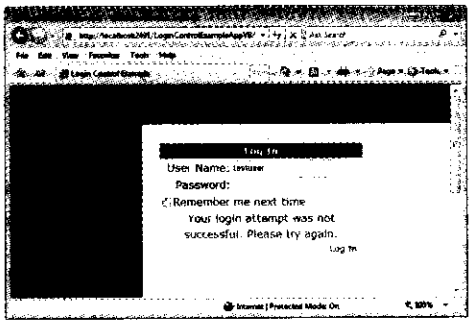


Figure 22.17: Displaying a Failed Authentication

Summary

In this chapter, you have learned about the Membership service provider and ASP.NET login controls, which are used together for authentication and authorization of website users and also enable you to create a secure website. The login controls you learned are Login, LoginView, LoginStatus, LoginName, PasswordRecovery, CreateUserWizard, and ChangePassword. In addition, you have also learned to implement these login controls in an application. In the next chapter, you will learn about master pages and themes.

Quick Revise

Q1. Why login Controls are used in an ASP.NET website?

Ans: Login controls are used to provide a robust login solution for websites without writing a single line of code. These controls allow you to implement security in a website, with the help of authentication and authorization. The login controls are first Introduced with Visual Studio 2005 IDE and are also part of Visual Studio 2008 IDE.

Q2. What are the various login controls you can find in Visual Studio 2008 IDE?

Ans: You can find the following login controls in the Visual Studio 2008 IDE, under the Login tab of the Toolbox:

- Login
- LoginName
- LoginStatus
- LoginView
- PasswordRecovery
- CreateUserWizard
- ChangePassword

Q3. What do you understand by authentication and authorization?

Ans: The process of recognizing the identity of a user is known as authentication. The authorization is performed with the help of username and password. Once the user is logged on to the website, the user is provided access to various resources, such as databases and printers, to the authenticated users. The process of

proving access to various website resources is known as authorization. We can also say, that authentication and authorization are used to secure your website and its resources from anonymous users.

Q4. What do you understand by user accounts and how it is created?

Ans: The user can login to the website with the help of username and password, this username and password is provided by registering user with the website. The process of registering user is called creating a user account. A user account generally stores username, password, e-mail address and other necessary information about the user. The Visual Studio 2008 IDE provides a powerful tool called ASP.NET Website Administration Tool, for user accounts of an ASP.NET website.

Q5. What is the Login control? How it is helpful to restrict access of unauthorized users?

Ans: The Login control is a Web Server control that provides a user interface for authenticating users of a website. This control accepts a user id and a password, which are used to authenticate the user of the website. Generally, the Login control is used in designing the Home page for a website. By using the Login control on a Web page of any website, you can restrict access to other Web pages of the website for unauthenticated users. It can be done by redirecting only authenticated users to any of those Web pages of the website.

Q6. How can you redirect an authenticated user to another page of the website using the Login control?

Ans: At time the user visits the website, the user is asked to enter username and password to authenticate user. After successful authentication the user must be redirected to the Web page from where the user can continue its interaction with the website. For this, you can use the `DestinationPageUrl` property of the Login control and specify the address of the Web page (to which the user should be redirected) as value of this property.

Q7. Why the LoginName control is used in a ASP.NET website?

Ans: The LoginName control is a Web Server control that displays the user id of the currently logged in user. The user id is displayed only for the authenticated users. You can edit the look and feel of the user id by changing the font and other formatting properties of this control. Moreover you can also the `ApplyStyleSheetSkin` method to apply a predefined style (skin) to format the display of user id.

Q8. Give a brief description of the LoginStatus control.

Ans: The LoginStatus control is a Web Server control that displays text on itself specifying whether or not the user of the website is currently logged in. The text displayed on this control changes according to the logged in status of the user. The LoginStatus control has the following two views:

- **Logged Out**—It is displayed when the user is not logged in. In this view, the text Login is displayed on the LoginStatus control. You can specify your own text by using the LoginText property.
- **Logged In**—It is displayed when the user is logged in. In this view, the text Logout is displayed on the LoginStatus control. You can specify your own text by using the LogoutText property.

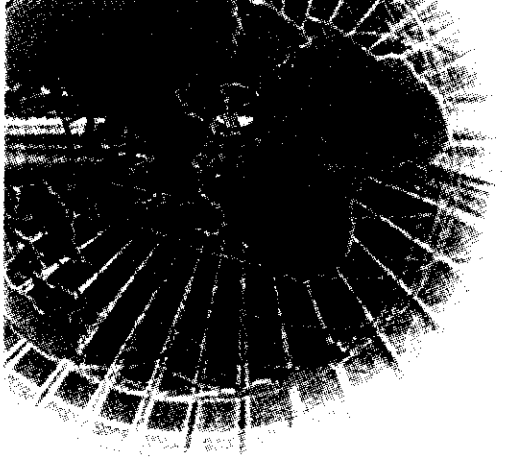
Q9. How the PasswordRecovery control is used to recover the forgotten password?

Ans: The PasswordRecovery control is a Web server control that provides a user interface for recovering or resetting a forgotten password. Passwords recovered by using the PasswordRecovery control are sent to the e-mail addresses of the concerned users, specified at the time of creating the user account. The PasswordRecovery control has the following three views:

- **UserName**—In this view, the user is asked to enter the user name to recover the password.
- **Question**—In this view, the security question is displayed and user is asked to enter the answer for its security question.
- **Success**—In this view, a message is displayed to the user that specifies that the retrieved password has been sent to the user.

Q10. What settings you have to do for retrieving the password using the PasswordRecovery control through an e-mail?

Ans: The PasswordRecovery control through an e-mail only when you have configured the virtual SMTP e-mail server through ASP.NET Web Site Administration Tool or by specifying required information in the web.config file. For this you have to specify Server name, Server Port, Authentication type, e-mail id of the sender and its corresponding password.



23

Inside Master Pages and Themes

A consistent look and feel is more of a necessity than a luxury for real-world Web applications. However, ensuring a consistent look and feel for the pages in Web applications has not always been easy. In ASP.NET 3.5, developers can incorporate consistent look and feel in their applications by using features such as master pages and themes.

The concept of a master page is all about incorporating visual inheritance in Web applications. In other words, you define the layout of a Web page once, and all the pages of Web applications inherit that layout. When you create a master page, what you actually do is create a template page, based on which all the other pages are derived. Therefore, all pages in the Web application inherit some common sections of that master page, which maintains consistency in the Web application.

In addition to master pages, there is another outstanding concept of ASP.NET 3.5—themes. A theme is a collection of settings for the controls and Web pages that defines their appearance. These themes are applied across all the pages in a Web application to maintain a consistent appearance.

In this chapter, we explore the need for master pages and themes. We then discuss the concepts of master pages and themes in detail.

Need for Master Pages and Themes

When you want to create Web pages that contain some common elements and customize the properties of different controls, such as `Buttons`, `TextBox`, and `Labels`, master pages and themes are needed. For example, you want to create a website for your company in which all the Web pages have a header stating the company name and all the buttons on the Web pages are of blue color. You can create this website by using master pages and themes.

Master page is a feature in ASP.NET 3.5 that helps define the overall layout of a Web application and reuse the defined layout in all the pages derived from the master page. A master page contains markups and controls that you can share across different Web pages of your website. This makes your website more manageable and also avoids the duplication of code. To understand the need of master pages, let's consider a scenario.

Suppose you want to include a navigation bar at a common place in all the Web pages of your website. If you are not using a master page, you will have to copy and paste the code for the common navigation bar on each page, which obviously is a tedious and time-consuming process. Also, if later you want to change the navigation bar slightly, then you will have to manually change the navigation bar code in all the pages of the Web application. Therefore, this does not seem to be a right way to bring consistency in your applications. However, if you are using master pages, you just need to include the navigation bar code in the master page, and all the Web pages will inherit it. It means no overhead of copying and pasting the code in different pages is required. Themes are needed when you want to keep the style and layout information files separate from other website files.

Understanding Master Pages

A master page defines the overall layout of a Web application and all the pages are derived from this master page. A master page can contain markups, controls, banners, navigation menus, and other elements that you want to include in all the pages of your website. The Web pages that inherit the properties defined in master pages are called content pages. The content pages display their own properties as well as the properties inherited from master pages. The following are the key functions of a master page:

- Defining common properties of a website, such as banners, navigation menus, and other elements that can be accessed by the content pages
- Allowing a single or multiple content pages to access single or multiple master pages
- Displaying the content of each content page in the content placeholder of the master pages

To create a master page for a website, identify the controls that you want to display on all the pages and then add these controls to the master page. You then create the `ContentPlaceHolder` control for the master page to place the content of the Web pages. When this website is executed, then the layout of the master page and the content in `ContentPlaceHolder` control are merged to display the output of the Web page.

NOTE

A master page can contain multiple ContentPlaceHolders.

There are two types of master pages – Simple Master Page and Nested Master Page.

Simple Master Page

A simple master page lets you simplify your Web site design by creating master pages and the content pages. You don't need to create the submaster pages. The submaster pages are needed when you use the nested master pages. A simple master page is a page that is combined with the content page to display the combined page. The concept of simple master and content pages can be better visualized with the help of Figure 23.1:

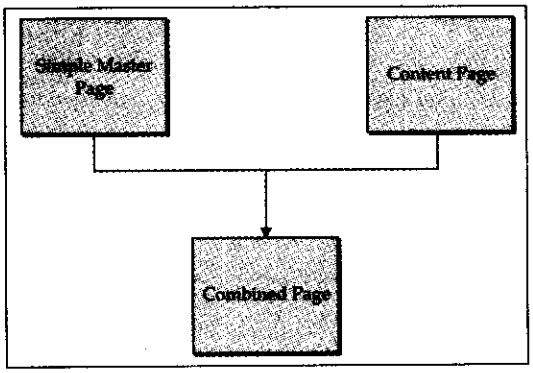


Figure 23.1: The block diagram of a combined page

As, you can see in the above figure, the master and content pages are combined together to generate a single page that inherits its properties from both the pages.

Creating Master Pages

As we have already mentioned, the need for creating master pages arises when you want to incorporate a common look and feel across all the pages of a website. By creating a master page, we actually create a template that is referred to by all the content pages of the Web application A master page has a `.master` extension, while a content page has the `.aspx` extension.

Master pages can be created in a simple text editor, such as a Notepad, or by using tools, such as Visual Web Developer (VWD) Express Edition, or Microsoft Visual Studio 2008. There are two types of master pages, Simple and Nested. In the case of simple master pages, when a user requests for the Web page, the content page is displayed with the properties inherited from a simple master page. To create simple master pages, you need to create a master page and then a content page that inherits the properties of the master page. However, nested master pages are slightly different. They are created with a master page, submaster page(s), and content page(s). The submaster pages inherit properties of the master pages and content pages inherit properties of the submaster pages. To create nested master pages, you need to create master pages, submaster pages, and content pages that inherit the properties of both master and submaster pages. Now, let's delve deeper into this.

Simple Master Pages

Now, let's put the concept of simple master page into practice. We create a Web application named `MyOrganizationVB` that creates a master page, and a content page that inherits the created master page. You can find the code of `MyOrganizationVB` application in the Code\ASP.NET\Chapter 23\`MyOrganizationVB` folder on the CD. To do so, follow these steps:

1. After creating the `MyOrganizationVB` Web application, you need to add a master page.
2. To add a master page, select the Master Page option from the Templates pane of the Add New Item dialog box and click the OK button. The `MasterPage.master` page opens on Visual Studio 2008.

3. Now, add the code, as shown in Listing 23.1, to the `MasterPage.master` page:

Listing 23.1: Code for the `MasterPage.master` Page

```
<%@ Master Language="VB" CodeFile="MasterPage.master.vb" Inherits="MasterPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head id="Head1" runat="server">
   <title>Master Page</title>
   <link href="StyleSheet.css" rel="stylesheet" type="text/css" />

</head>

<body>
   <form id="form1" runat="server">
   <div>
        <div id="header">

        </div>
        <div id="sidebar">
             <div id="nav">
                   
             </div>
        </div>
        <div id="content">
             <div class="itemContent">
             <strong><em>My Organization Web Site<br />
             <asp:Image ID="Image1" runat="server" ImageUrl="~/MyOrganization.bmp" />
             <br />
             </em></strong>
             <br />
             <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
             </asp:ContentPlaceHolder>
             <br />
             <br />
              <br />
             <div id="footer">
                  <p class="left">
                       All content copyright &copy; Kogent Solutions Inc.</p>
             </div>
        </div>
   </div>
   </form>
</body>
</html>
```

The preceding listing adds some text at the top and bottom of the master page and adds Image control through which we can display an image on the Web page. An important thing to note in the preceding listing is that there is a `ContentPlaceHolder` tag, which defines an area for the content pages to place their content.

You can place multiple `ContentPlaceHolder` tags in a master page. You must also ensure that you add the logo image to the directory structure of the `MyOrganizationVB` application. Figure 23.2 shows the master page in the Design view:

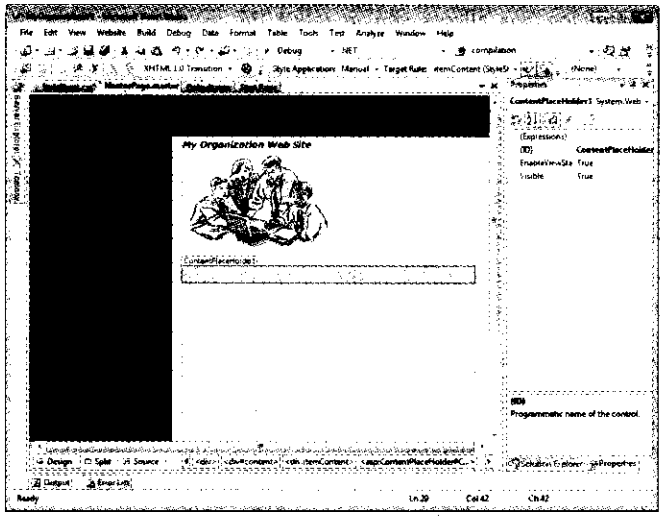


Figure 23.2: Design View of the Master Page

4. Now, you need to create a content page to inherit the properties of the master page. A content page is created in the same way as any other `.aspx` page is created, with a little difference in that you need to select the Select master page check box in the Add New Item dialog box while creating the content page, as shown in Figure 23.3:

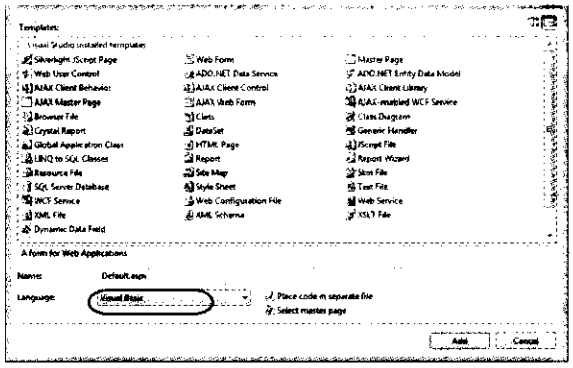


Figure 23.3: Showing the Add New Item Dialog Box

5. Click the Add button. The Select a Master Page dialog box appears, as shown in Figure 23.4:

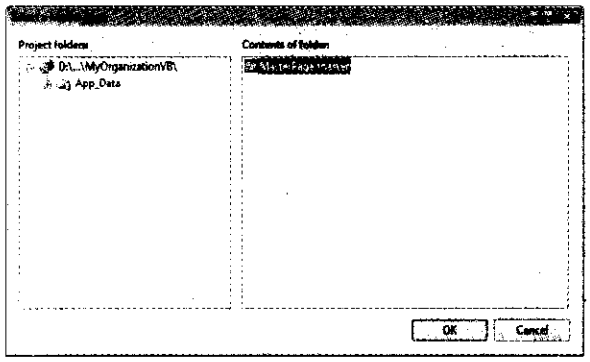


Figure 23.4: The Select a Master Page Dialog Box

The Select a Master Page dialog box allows you to select the master page from which your content page (`Default.aspx`) inherits its properties.

6. Select the `MasterPage.master` option and click on the OK button (Figure 23.4). The `Default.aspx` page appears in VISUAL STUDIO 2008.

NOTE

We have deleted the default page, Default.aspx, created automatically when you create a new Web application because it does not inherit the properties of the Master page.

7. Now, add the code, as shown in Listing 23.2, to the `Default.aspx` page:

Listing 23.2: Code for the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false"
  MasterPageFile="~/MasterPage.master" CodeFile="Default.aspx.vb"
  Inherits="_Default"  Title= "Content Page"%>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
    Runat="Server">
  Enter your name in the text box<br />
  <br />
  <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
  <br />
  <asp:Button ID="Button1" runat="server" Text="Click to Submit" />
  <br />
  <asp:Label ID="Label1" runat="server" Text="Label" Visible="False"
    Width="132px"></asp:Label>
</asp:Content>
```

8. Add the code, as shown in Listing 23.3, to the code-behind file of the `Default.aspx` page:

Listing 23.3: Code for the Code-Behind File of the `Default.aspx` Page

```
Partial Class _Default
  Inherits System.Web.UI.Page
  Protected Sub Button1_Click(ByVal sender As Object, ByVal e As
    System.EventArgs) Handles Button1.Click
      Label1.Text = "You have entered " & TextBox1.Text
      Label1.visible = True
  End Sub
End Class
```

9. Now, run the `MyOrganizationVB` Web application. The ASP.NET engine combines both the content and master pages to generate a single combined page. You can see the final output of the Web application in Figure 23.5:

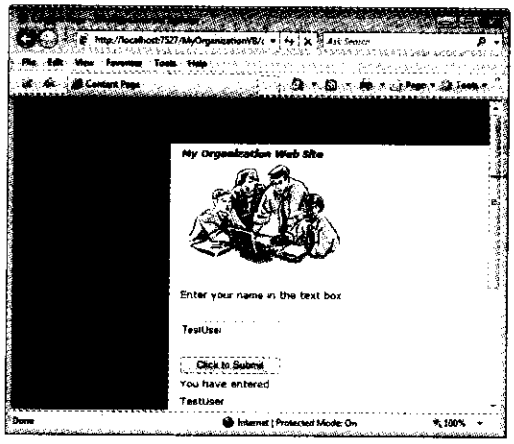


Figure 23.5: Output of the MyOrganizationVB Web Application

Nested Master Pages

The concept of nested master pages is very much the same as that of simple master pages. The only difference is that the nested master pages are used in websites that have several hierarchical levels. You can best visualize the concept of nested master pages with the help of Figure 23.6:

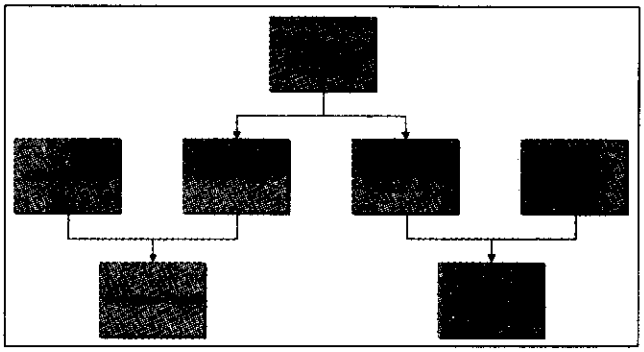


Figure 23.6: The Block Diagram of the Nested Master Page

In the preceding figure, an application can have a number of master pages depending on the level of hierarchy it incorporates.

Creating Nested Master Pages

To elaborate further on the concept of nested master pages, let's go back to the `MyOrganizationVB` application that we had created earlier. In that application, we had used a single master and a content page. Now, we will add another master page to it. You can find this modified application as `MyOrganizationNestedVB`. You can find the code of `MyOrganizationNestedVB` application in the `Code\ASP.NET\Chapter 23\MyOrganizationNestedVB` folder on the CD. To create a new submaster page that inherits the properties of the main master page, you need to select the Select master page check box in the Add New Item dialog box (shown earlier in Figure 23.3). We name this submaster page `hrd.master`. Now, add the code, as shown in Listing 23.4, to the `hrd.master` page:

Listing 23.4: Code for the `hrd.master` Page

```
<%@ Master Language="VB" MasterPageFile="~/MasterPage.master" AutoEventWireup="false"
    CodeFile="hrd.master.vb" Inherits="hrd" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
<asp:Label ID="Label1" runat="server" BackColor="Aqua" BorderColor="Black"
    BorderWidth="2px"
    BorderStyle="Solid" Font-Size="Large" Text="Human Resource Department"
    Visible="true">
    </asp:Label>
    <br/>
    <br/>
    <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
    </asp:ContentPlaceHolder>
</asp:Content>
```

You also need to make the following changes in the Page directive of the `Default.aspx` page, as shown in Listing 23.5:

Listing 23.5: Code for the `Default.aspx` Page

```
<%@ Page Language="VB" MasterPageFile="~/hrd.master" AutoEventWireup="false"
    CodeFile="Default.aspx.vb" Inherits="_Default" Title="Nested Master Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    Enter your name in the text box<br />
    <br />
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
    <br />
```

```
    <asp:Button ID="Button1" runat="server" Text="Click to Submit" />
    <br />
    <asp:Label ID="Label1" runat="server" Text="Label" Visible="False" ></asp:Label>
</asp:Content>
```

Now, let's build and execute the application and click the Click to Submit button to call the click event of the button. Figure 23.7 shows the output after clicking the button on the Web page:

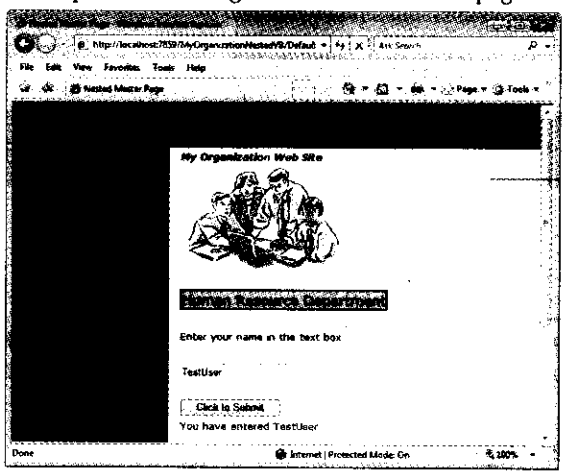


Figure 23.7: The MyOrganizationNestedVB Application at Work

In the preceding figure, the contents and layout of the `MasterPage`, `hrd`, and `Default` pages are combined together to generate a single page.

Configuring Master Pages

To associate a content page with a master page, you need to either specify the name of the master page in the content page by using the `<%@ Page%>` directive, or specify the name of the master file in the Web application configuration file. Using configuration files to associate a content page with a master page helps you to easily maintain your website, because the configuration files provide a single location to change the master page associated with multiple content pages.

For example, to use the configuration file to associate the `MasterPage.master` page with the `Default.aspx` content page, open the `web.config` file of your website, and then write the following code in the `<system.web>` section of the configuration file:

```
<pages masterPageFile="~/MasterPage.master" />
```

After adding the preceding code, you are not required to exclusively set the `MasterPageFile` attribute in the Page directive of the `Default.aspx` page. However, the following points must be considered while configuring master pages:

- Master page included in the content page by using the `<%@ Page%>` directive has higher precedence over the associated master pages by using the `web.config` file. If the master file is configured using both methods, the master page path mentioned in the content page overrides the master page defined in the `web.config` file.
- If a website contains multiple `web.config` files located in different subfolders, each can be used for configuring different master pages.

When you associate a master page with a content page, the content page automatically includes the Title attribute. You can change the title of the content page by modifying the `Title` attribute in the `<%@ Page%>` directive. If you want to use the Master Page title, you can remove the Title attribute from the content Page directive.

Modifying Content on the Master Page from the Content Page

You can update the content on the `master page` by finding the control that needs to be modified on the master page from the content page. To demonstrate how all this happens, let's put a Calendar control on the master page. We will name this new application as `CalendarUpdationVB`. You can find the code of `CalendarUpdationVB` application in the Code\ASP.NET\Chapter 23\`CalendarUpdationVB` folder on the CD. After creating the `CalendarUpdationVB` application, follow these steps to modify the content in the master page:

1. Add the code, as shown in Listing 23.6, to the `MasterPage.master` page:

Listing 23.6: Code for the `MasterPage.master` Page

```
<%@ Master Language="VB" CodeFile="MasterPage.master.vb" Inherits="MasterPage" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>Calendar Updation</title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="form1" runat="server">
  <div>
      <div id="header">

      </div>
      <div id="sidebar">
          <div id="nav">
              <br />
          <br />
          <br />
          <br />
          <asp:Calendar ID="Calendar1" runat="server" BackColor="White"
          BorderColor="Black" Font-Names="Verdana" Font-Size="9pt" ForeColor="Black"
          Height="250px" NextPrevFormat="ShortMonth"
          Width="330px" BorderStyle="Solid" CellSpacing="1">
          <SelectedDayStyle BackColor="#333399" ForeColor="White" />
          <TodayDayStyle BackColor="#999999" ForeColor="White" />
          <DayStyle BackColor="#CCCCCC" />
          <OtherMonthDayStyle ForeColor="#999999" />
          <NextPrevStyle Font-Size="8pt" ForeColor="White" Font-Bold="True" />
          <DayHeaderStyle Font-Bold="True" Font-Size="8pt" ForeColor="#333333"
          Height="8pt" />
          <TitleStyle BackColor="#333399" Font-Bold="True" Font-Size="12pt"
          ForeColor="White" Height="12pt" BorderStyle="Solid" />
          </asp:Calendar>
          <br />
          <br />
          <br />
          </div>
      </div>
      <div id="content">
          <div class="itemContent">
          <br />
           <br />
          <br />
           <br />
          <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
          </asp:ContentPlaceHolder>
          <div id="footer">
               <p class="left">
              All content copyright &copy; Kogent Solutions Inc.</p>
          </div>
          </div>
      </div>
```

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

2. Now, add the code, as shown in Listing 23.7, to the `Default.aspx` page:

Listing 23.7: Code for the `Default.aspx` Page

```
<%@ Page Language="VB" MasterPageFile="~/MasterPage.master" AutoEventWireup="false"
    CodeFile="Default.aspx.vb" Inherits="_Default" Title="Content Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
    Enter date(MM/DD/YY)in Textbox for updating Calendar control<br />
    <br />
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
     <asp:Label ID="Label1" runat="server" Text="Label"
    Visible="false"></asp:Label><br />
    <asp:Button ID="Button1" runat="server" Text="Update Calendar" />
</asp:Content>
```

3. Now, add the code, as shown in Listing 23.8, to the code-behind file of the `Default.aspx` page:

Listing 23.8: Code for the Code-Behind File

```
Partial Class _Default
    Inherits System.Web.UI.Page
    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Handles Button1.Click
        Try
            Dim cal As Calendar
            cal = CType(Master.FindControl("Calendar1"), Calendar)
            If Not cal Is Nothing Then
                cal.VisibleDate = TextBox1.Text
                cal.SelectedDate = TextBox1.Text
                Label1.Visible = False
            End If
            Catch
                Label1.Text = "Wrong Format Entered"
                Label1.Visible = True
        End Try
    End Sub
End Class
```

4. Now, let's run the application. Figure 23.8 shows the output:

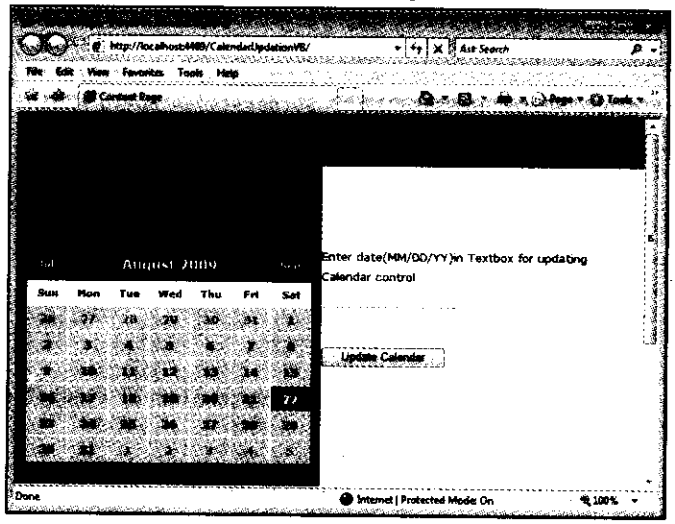


Figure 23.8: The CalendarUpdationVB Application at Work

5. In Figure 23.8, the Calendar control is showing 22 August 2009 as the selected date. Modify the selected date by entering `12/06/08` in the textbox and click the Update Calendar button to update the calendar with the new date, as shown in Figure 23.9:

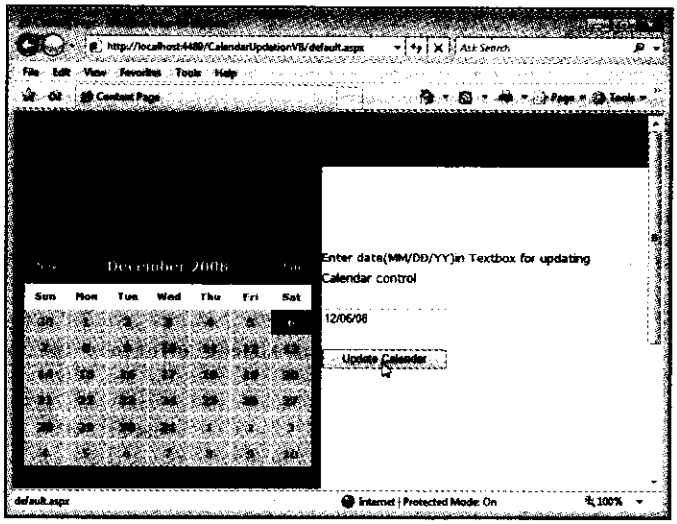


Figure 23.9: The Modified Date in the Calendar

After modifying the content of the master page, let's learn how to display the properties and methods by using a master page.

Displaying Properties and Methods Using a Master Page

You can display the properties and methods by using a master page to increase the accessibility over the content of a master page. Displaying the properties and methods by using a master page is also called exposing properties and methods. Displaying is a technique that you can use to change the public properties of a master page. To understand this concept in detail, let's consider a master page with a public header property exposed to the content page. Now, create a Web application, `DisplayingPropertiesVB`. You can find the code of `DisplayingPropertiesVB` application in the Code\ASP.NET\Chapter 23\`DisplayingPropertiesVB` folder on the CD and follow these steps to display properties and methods:

1. Add the code, as shown in Listing 23.9, to the master page named `MasterPage.master`:

Listing 23.9: Code for the `MasterPage.master` Page

```
<%@ Master Language="VB" CodeFile="MasterPage.master.vb" Inherits="MasterPage" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
  Private _header As String
  Public Property Header() As String
      Get
          Return _header
      End Get
      Set(ByVal value As String)
          _header = value
      End Set
  End Property
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Master Page</title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
  <asp:ContentPlaceHolder ID="head" runat="server">
```

```
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <div id="header">

        </div>
        <div id="sidebar">
            <div id="nav">
                 
            </div>
        </div>
        <div id="content">
            <div class="itemContent">
            <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
            </asp:ContentPlaceHolder>
            <div id="footer">
                <p class="left">
                All content copyright &copy; Kogent Solutions Inc.</p>
            </div>
            </div>
        </div>
    </div>
    </form>

</body>

</html>
```

In the preceding listing, the master page defines a property named header. The content page passes the value of the header to the master page and the new header is displayed.

2. Add the content page (`Default.aspx`) to the Web application, and write the code, as shown in Listing 23.10:

Listing 23.10: Code for the Content Page

```
<%@ Page Language="VB" MasterPageFile="~/MasterPage.master" AutoEventWireup="false"
    CodeFile="Default.aspx.vb" Inherits="_Default" Title="Untitled Page" %>

<%@ MasterType VirtualPath="~/MasterPage.master" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="Server">
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
  Welcome to the Custom page Header
  <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
</asp:Content>
```

3. Add the code, as shown in Listing 23.11, in the code-behind file of the `Default.aspx` page,:

Listing 23.11: Code for the Code-Behind File

```
Partial Class _Default
  Inherits System.Web.UI.Page
  Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
  Me.Load
      Master.Header = "Header"
      Master.Page.Title = "Content Page"
      Label1.Text = Master.Header
  End Sub
End Class
```

4. Now, run the Web application. The Internet Explorer window with the content page is displayed, as shown in Figure 23.10:

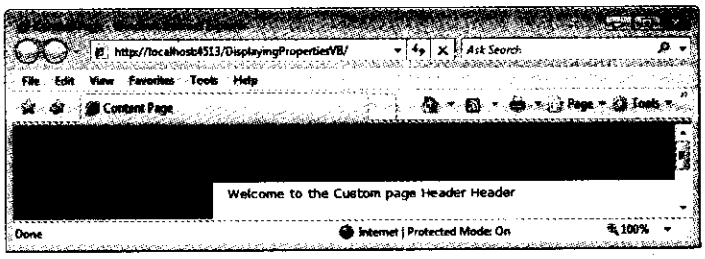


Figure 23.10: The Content Page Displaying the Header

The title of the content page that is defined in the content page is displayed, and the master page title is overridden.

Loading Pages Dynamically

Loading pages dynamically can be achieved by providing a link of master pages on the content page. Dynamic loading of the master pages takes place during the `PreInit` event of a Web page and the `PreInit` event is coded in the content page. To show how dynamic page loading works, let's create multiple master pages and a content page. When the website is executed, the content page inherits properties of one of the master page and also contains a link to inherit the properties of other master pages. Now, let's create a Web application `DynamicMasterVB`. You can find the code of `DynamicMasterVB` application in the `Code\ASP.NET\Chapter 23\DynamicMasterVB` folder on the CD and follow these steps to load the master page dynamically:

1. After creating the Web application, add a master page (named `Master1.master`) and write the code, as shown in Listing 23.12, in the `Master1.master` Page:

Listing 23.12: Code for the `Master1.master` Page

```
<%@ Master Language="VB" CodeFile="Master1.master.vb" Inherits="Master1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
  Protected Sub LinkButton1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
      Session("masterpage") = "Master2.master"
      Response.Redirect(Request.Url.ToString())
  End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Loading Pages Dynamically</title>
  <link href="Stylesheet.css" rel="stylesheet" type="text/css" />
  <asp:ContentPlaceHolder ID="head" runat="server">
  </asp:ContentPlaceHolder>
</head>
<body>
  <form id="form1" runat="server">
  <div>
      <div id="header">

      </div>
      <div id="sidebar">
          <div id="nav">
               
          </div>
      </div>
      <div id="content">
          <div class="itemContent">
          <h1>
          Master Page without Color</h1>
          <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
          </asp:ContentPlaceHolder>
```

```
                <asp:LinkButton ID="LinkButton1" runat="server"
                OnClick="LinkButton1_Click">Page With Background Color
                </asp:LinkButton> 
                <div id="footer">
                <p class="left">
                    All content copyright &copy; Kogent Solutions Inc.</p>
                </div>
                </div>
            </div>
        </div>
    </form>
</body>
</html>
```

2. Now, create another master page (named `Master2.master`) for dynamic page loading and write the code, as shown in Listing 23.13:

Listing 23.13: Code for the `Master2.master` Page

```
<%@ Master Language="VB" CodeFile="Master2.master.vb" Inherits="Master2" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
    Protected Sub LinkButton1_Click(ByVal sender As Object, ByVal e As EventArgs)
        Session("masterpage") = "Master1.master"
        Response.Redirect(Request.Url.ToString())
    End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Loading Pages Dynamically</title>
    <link rel="stylesheet" type="text/css" />
    <asp:ContentPlaceHolder ID="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body bgcolor="#99ccff">
    <form id="form1" runat="server">
    <div>
        <div id="header">
        </div>
        <div id="sidebar">
            <div id="nav">
                 
            </div>
        </div>
        <div id="content">
            <div class="itemContent">
            <h1>
            Master Page With Color
            </h1>
            <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
            </asp:ContentPlaceHolder>
            <asp:LinkButton ID="LinkButton1" runat="server"
            OnClick="LinkButton1_Click">Page Without Background Color
            </asp:LinkButton>
            <div id="footer">
            <p class="left">
                All content copyright &copy; Kogent Solutions Inc.</p>
            </div>
            </div>
        </div>
    </div>
    </form>
</body>
</html>
```

The following are the differences between the `Master1.master` and `Master2.master` pages:

- The `Master1.master` page does not contain a background color, whereas the `Master2.master` page contains a background color.
- The heading (h1) on the `Master1.master` page is Master Page Without Color, whereas the heading (h1) on the `Master2.master` page is Master Page With Color.
- The text and link properties of the `LinkButton` control are different in both the master pages.

To inherit the properties of master pages dynamically, you need a content page.

3. Create a content page (`Default.aspx`), and write the code, as shown in Listing 23.14, in it:

Listing 23.14: Code for the Content File `Default.aspx`

```
<%@ Page Language="VB" MasterPageFile="~/Master1.master" AutoEventWireup="false"
  CodeFile="Default.aspx.vb" Inherits="_Default" Title="Loading Pages Dynamically" %>
<script runat="server" language="vb">
  Sub Page_PreInit(ByVal sender As Object, ByVal e As EventArgs) _
      Handles Me.PreInit
      If Not Session("masterpage") Is Nothing Then
          Me.MasterPageFile = CType(Session("masterpage"), String)
      End If
  End Sub
  Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
  Label1.Text = "<b>Hello " & TextBox1.Text & "! Welcome to MyWebSite.</b>"
  End Sub
</script>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
  <b>Enter Your Name: </b>
  <br />
  <asp:TextBox ID="TextBox1" runat="server" />
  <br />
  <br />
  <asp:Button ID="Button1" runat="server" Text="Enter" OnClick="Button1_Click" />
  <br />
  <asp:Label ID="Label1" runat="server" />
</asp:Content>
```

4. Now, run the Web application. The content page inherits properties from the `Master1.master` page, as shown in Figure 23.11:

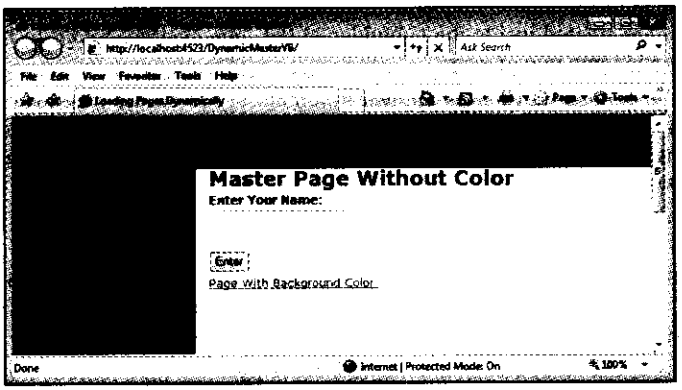


Figure 23.11: The Default.aspx Page with Master1.master Properties

5. Enter name in the textbox and click the Enter button and also click the link Page With Background Color. The content page inherits properties from the second master page `Master2.master`, as shown in Figure 23.12:

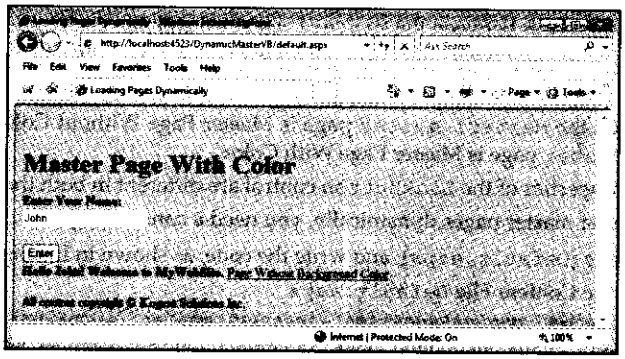


Figure 23.12: The Default.aspx Page with Master2.master Properties

You can click the link Page Without Background Color to display the content page again, which inherits properties from the first master page `Master1.master`.

Understanding ASP.NET AJAX Master Pages

For creating an AJAX-enabled Web page, it is mandatory to add the `ScriptManager` control on the Web page. The `ScriptManager` control is used to manage ASP.NET AJAX script libraries and script files, partial-page rendering, and client proxy class generation for Web and application services. (ASP.NET AJAX and its controls are discussed in Chapter 32: *Developing ASP.NET AJAX Applications*).

ASP.NET 3.5 provides master pages with built-in AJAX features. There is no need to explicitly add the `ScriptManager` control on the AJAX Master Page. In ASP.NET 3.5, you can use the AJAX Master Page template to create an AJAX-enabled Master Page, where the `ScriptManager` control is already added.

Another benefit of the AJAX Master Page is that the `web.config` file of the application is automatically configured; you do not need to configure the file. This saves the time spent in configuring the `web.config` file to use AJAX features in a master page.

Understanding Themes

As we have already stated, themes define the style properties of a Web page. Now, let's explore the elements of themes, such as skins, Cascading Style Sheets (CSS), and images. A skin is a file with the `.skin` extension that contains property settings for individual controls, such as `Button`, `TextBox`, or `Label`. You can either define skins for each control in different skin files or define skins for all the controls in a single file. The skin files with the `.skin` extension are created inside the subfolder of the `App_Themes` folder in the Solution Explorer of a website. Skins are divided into two groups:

- **Default**—It is the default skin that is applied automatically to all the controls of same type present on a Web page, when a theme is applied to the page. The `SkinID` attribute is used to determine whether the skin applied to a control on a Web page is a default skin or not. A skin is a default skin, if the skin does not have a `SkinID` attribute.
- **Named**—It is a custom skin that is applied to controls when a theme is applied to the page. The named skin has the `SkinID` property set and associates the `.skin` file with it.

In contrast to skins that apply the property settings to controls, CSS is a mechanism for adding fonts, colors, and styles to Web pages. CSS has the `.css` file extension and applies a style sheet as a part of theme. To use CSS, you need to store the CSS file inside the subfolder of the `App_Themes` folder.

In addition to skins and CSS, themes can also include other resource files, such as images, script files or sound files that can be used as style properties for the Web pages. These files are also placed inside the subfolders of the `App_Themes` folder. To access a resource file stored in a subfolder, you need to explicitly specify the path of that file in the code of the Web page. For example, the syntax to access the `h1.ext` file located in the theme subfolder of the `App_Themes` folder is:

```
<asp:Image runat="server" ImageUrl="themesubfolder/h1.ext" />
```

The resource files stored outside the theme folder can also be accessed by using the tilde (~) symbol in the code. For example, the syntax to access the `h1.ext` file located in the other1 folder located outside the `App_Themes` folder is:

```
<asp:Image runat="server" ImageUrl="~/other1/h1.ext" />
```

To sum up, we can say that themes incorporate all the elements, such as skins, stylesheets, and images to define a unanimous look and feel for the website. ASP.NET 3.5 contains two types of themes, page and global. ASP.NET 3.5 also introduced an attribute `StyleSheetTheme` to apply the themes on a Web page.

Now, let's discuss different types of themes.

Types of Themes in ASP.NET 3.5

Themes are of two types, page and global. A page theme contains the control skins, style sheets, graphic files, and other resources stored inside the subfolders of the `App_Themes` folder. A page theme is applied to a single page of the website. For different themes, separate subfolders are created in the `App_Themes` folder.

A global theme, on the other hand, is a theme that is applied to all the websites on a Web server and includes property settings, style sheets settings, and graphics. This theme allows you to maintain all the websites on the same Web server and to define the same style for all the Web pages of all the websites. Global themes are placed in the Themes folder and can be accessed by any website.

Creating Themes

Applying built-in themes to Web pages is quite easy, as you just need to specify the name of the theme with the theme attribute of the Page directive. However, you can also apply custom themes to Web pages. For this, you need to create a theme first. To demonstrate how to create themes, let's create a Web application `CreatingThemesVB`. You can find the code of `CreatingThemesVB` application in the Code\ASP.NET\Chapter 23\`CreatingThemesVB` folder on the CD.. Now, first, we need to add the `App_Themes` folder to our application and perform the following steps for it:

1. Right-click the website name on the Solution Explorer and select `Add ASP.NET Folder`→`Theme` from the shortcut menu to add the `App_Themes` folder. A subfolder named `Theme1` is automatically created inside the `App_Themes` folder.
2. Right-click the `Theme1` folder and select the Add New Item option from the context menu to display the Add New Item dialog box, as shown in Figure 23.13:

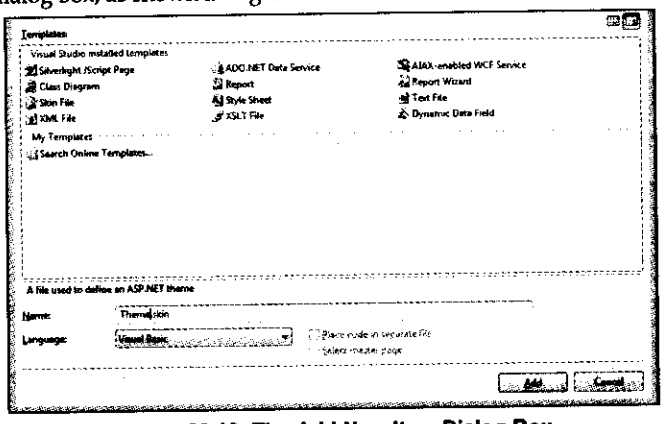


Figure 23.13: The Add New Item Dialog Box

3. Select the Skin File template option and specify the name of the skin as `Theme.skin`, and click the Add button. The `Theme.skin` file is added to the `Theme1` folder in the Solution Explorer,, as shown in Figure 23.14:

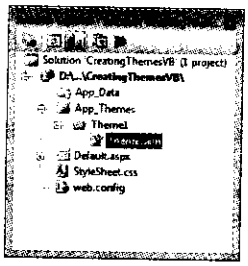


Figure 23.14: The Theme.skin File in the Solution Explorer

4. After creating a new skin file, add the code, as shown in Listing 23.15, to the `Theme.skin` file to define its style properties:

Listing 23.15: Code for the `Theme.skin` File

```
<asp:Label runat="server" width="350px" height="50px" font-bold="True" font-
    size="Large" forecolor="Yellow" backcolor="Gray"/>
<asp:button runat="server" forecolor="Blue" backcolor="Silver"/>
<asp:textbox runat="server" font-bold="True" font-size="Medium" forecolor="Blue"
    backcolor="Silver" font-italic="True"/>
```

In the preceding code, we have defined style properties for three controls, a Label, a Button, and a TextBox.

5. Now, to test the skin, add a Label control, a Button control, and a TextBox control in the `Default.aspx` page, as shown in Listing 23.16:

Listing 23.16: Code for the `Default.aspx` Page

```
<%@ Page Language="VB" Theme="Theme1" AutoEventWireup="false" CodeFile="Default.aspx.vb"
    Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Creating Themes Example</title>
    <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <div id="header">

        </div>
        <div id="sidebar">
            <div id="nav">
                 
            </div>
        </div>
        <div id="content">
            <div class="itemContent">
            <strong><em>Using Custom Themes</em></strong><br />
            <br />
            <br />
            <asp:Label ID="Label1" runat="server" Text="Enter Your Name in the Text
            Box"></asp:Label><br />
            <br />
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
            <br />
            <asp:Button ID="Button1" runat="server" Text="Click To Submit" /></div>
            <div id="footer">
            <p class="left">
                All content copyright &copy; Kogent Solutions Inc.</p>
            </div>
        </div>
    </div>
```

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

The important thing to note here is that we have set the `Theme` attribute in the page directive to `Theme1`, so that the theme specified in the `Theme.skin` file is applied to all the controls on the `Default.aspx` page.

6. Now, let's run the application. Figure 23.15 shows the output of the `CreatingThemes` Web application:

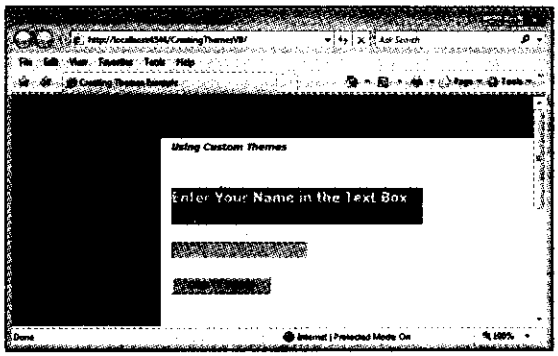


Figure 23.15: Output of the CreatingThemesVB Application

After creating themes in the Web application, let's move on to apply the themes at control and runtime level.

Applying Themes at Control and Runtime Levels

In the previous section, we have created a theme and applied it to the entire page. Apart from that, you can also apply a theme at control level. It means only specific controls whose `EnableTheming` and `Theme` properties are set can inherit the properties defined in the specified theme. Here is a sample text control declaration that applies a theme:

```
<asp:TextBox ID="TextBox1" runat="server" EnableTheming="true" Theme="Summer" />
```

Consider another scenario in which you want the end-users to select and apply themes at runtime. To demonstrate how this happens, let's create a Web application, `ApplyingThemesVB`. You can find the code of `ApplyingThemesVB` application in the `Code\ASP.NET\Chapter 23\ApplyingThemesVB` folder on the CD, and follow these steps:

1. Create two themes, `Simple.skin` and `Inverse.skin`, and place them in the `Theme1` and `Theme2` folders, respectively.
2. Now, add the code, as shown in Listing 23.17, to the `Simple.skin` file:

Listing 23.17: Code for the `Simple.skin` File

```
<asp:label runat="server" width="350px" height="50px" font-bold="True" font-
  size="Large" forecolor="Yellow" backcolor="Gray" />
<asp:button runat="server" forecolor="Blue" backcolor="Silver"/>
<asp:textbox runat="server" font-bold="True" font-size="Medium" forecolor="Blue"
  backcolor="Silver" font-italic="True"/>
```

3. You also need to add the code in the `Inverse.skin` file, as shown in Listing 23.18:

Listing 23.18: Code for the `Inverse.skin` File

```
<asp:label runat="server" width="350px" height="50px" font-bold="True" font-
  size="Large" forecolor="Blue" backcolor="Red"/>
<asp:button runat="server" forecolor="Silver" backcolor="Blue"/>
<asp:textbox runat="server" font-bold="True" font-size="Medium" forecolor="Silver"
  backcolor="Blue" font-italic="True"/>
```

4. After adding the code for themes, add the code, as shown in Listing 23.19, to the `Default.aspx` page:

Listing 23.19: Code for the `Default.aspx` Page

```
<%@ Page Language="VB" Theme="Theme1" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Applying themes Example</title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="form1" runat="server">
  <div>
      <div id="header">
      </div>
      <div id="sidebar">
          <div id="nav">
               
          </div>
      </div>
      <div id="content">
          <div class="itemContent">
          <div style="text-align: left">
          <br />
          <strong><em>Applying Themes at Runtime<br />
          </em></strong>
          <br />
          <asp:Image ID="Image2" runat="server" ImageUrl="~/theme image.bmp" /><br />
          <br />
          <asp:Label ID="Label1" runat="server" Text="Enter your name in the text
          box"></asp:Label><br />
           
          <br />
          <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
          <br />
          <asp:Button ID="Button1" runat="server" Text="Click to Submit" /><br />
          <br />
          <asp:Button ID="Button3" runat="server" Text="Button" /><br />
          <br />
          Select Theme<br />
          <a href="Default.aspx?Theme=Theme1">Simple</a> <a
          href="Default.aspx?Theme=Theme2">Inverse</a>
          <br />
          </div>
          <div id="footer">
              <p class="left">
                  All content copyright &copy; Kogent Solutions Inc.</p>
          </div>
          </div>
      </div>
  </div>
  </form>
</body>
</html>
```

The preceding listing uses an image `theme image.bmp`, so add the corresponding image in the Solution Explorer as well.

5. Now, add the code, as shown in Listing 23.20, to the code-behind file of the `Default.aspx` page:

Listing 23.20: Code for the Code-Behind File

```
Partial Class _Default
    Inherits System.Web.UI.Page
    Protected Sub Page_PreInit(ByVal sender As Object, ByVal e As System.EventArgs) Handles
    Me.PreInit
        Page.Theme = Server.HtmlEncode(Request.QueryString("Theme"))
    End Sub
```

```
End Class
```

6. Now, run the `ApplyingThemesVB` Web application. Figure 23.16 shows the output:

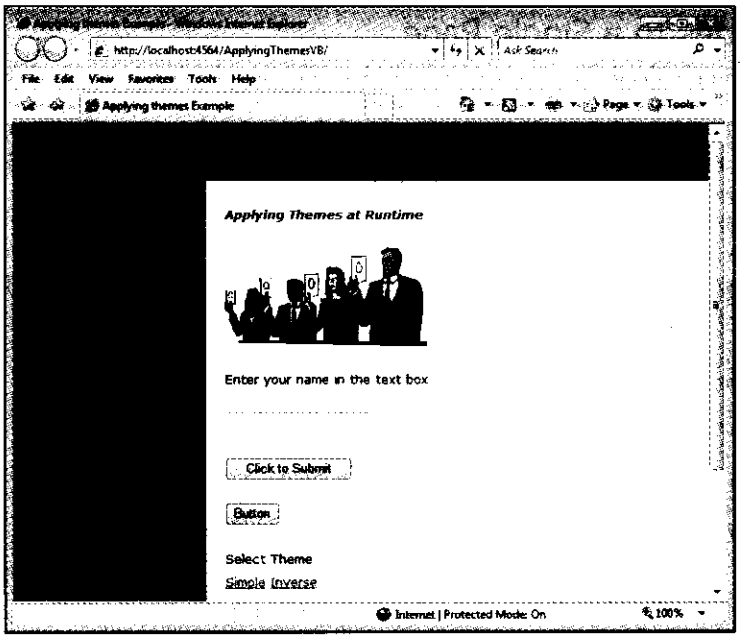


Figure 23.16: Output of the ApplyingThemes Application

7. To change the theme at runtime, select any option from the Select Theme link button, Simple theme link or Inverse theme link, to change the theme. Figure 23.17 shows the output in Simple theme:

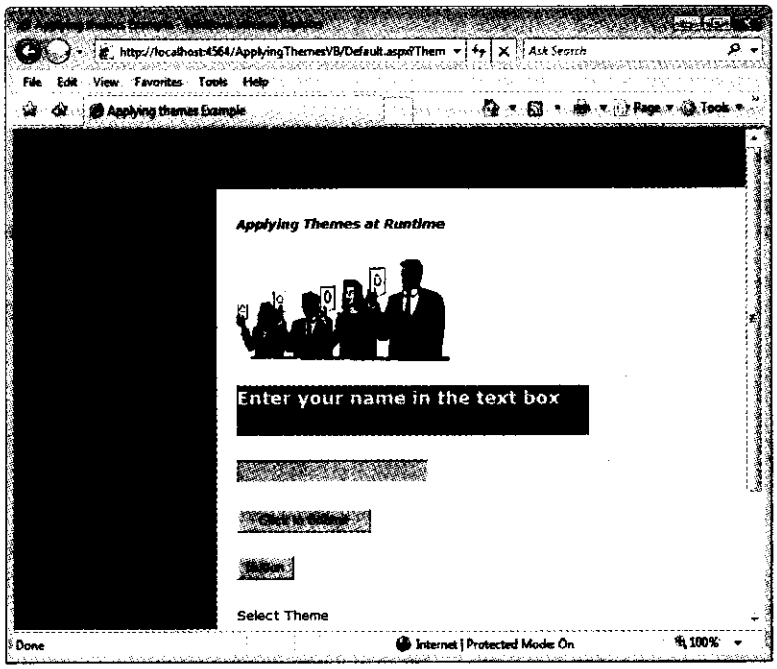


Figure 23.17: The Default.aspx Page in Simple Theme

Now, we have learned how to apply themes at control and runtime level, which is applied on all the pages of the application. Now, let's apply the themes on a single page.

Applying Themes to a Single Page

In this section, we apply the theme on a page of a Web application. To do so, create a Web application named `SinglePageThemeVB`. You can find the code of `SinglePageThemeVB` application in the `Code\ASP.NET\Chapter 23\SinglePageThemeVB` folder on the CD, and follows these steps:

1. After creating the application, drag two Label controls, two TextBox controls, and a Button control on the `Default.aspx` page. You can also add the controls on the `Defualt.aspx` page by writing the code shown in Listing 23.21:

Listing 23.21: Code for the `Default.aspx` Page

```
<%@ Page Language="VB" Theme="Theme1" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">
  <title>Applying Theme to a Single Page</title>
  <link href="StyleSheet.css" rel="Stylesheet" type="text/css" />
</head>
<body>
  <form id="form1" runat="server">
  <div>
      <div id="header">
      </div>
      <div id="sidebar">
          <div id="nav">
               
          </div>
      </div>
      <asp:Label runat="server" Text="Label"></asp:Label>
      <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
      <br />
      <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
      <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
      <br />
      <asp:Button ID="Button1" runat="server" Text="Button" />
      <div id="footer">
          <p class="left">
              All content copyright &copy; Kogent Solutions Inc.</p>
      </div>
  </div>
  </form>
</body>
</html>
```

We have added two `Label` controls, two `TextBox` controls, and a `Button` control in the `Default.aspx` page.

2. Now, create a theme to apply it on the `Default.aspx` page. To create a theme, right-click the name of the application and select the `Add ASP.NET Folder→Theme` option from the context menu. A subfolder named `Theme1` is automatically created inside the `App_Themes` folder.
3. Right-click the `Theme1` folder and select the Add New Item option from the context menu to display the Add New Item dialog box, as shown in Figure 23.18:

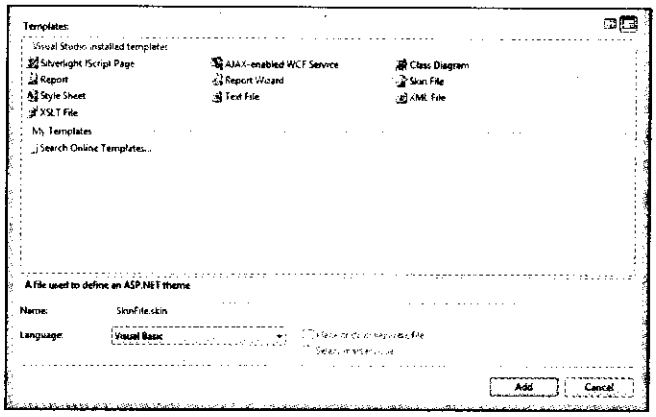


Figure 23.18: The Add New Item Dialog Box

4. Select the Skin File template and click the Add button. A file named `SkinFile.skin` is created in the `Theme1` folder.
5. Now, add the code, as shown in Listing 23.22, to the `SkinFile.skin` file:

Listing 23.22: Code for the `SkinFile.skin` File

```
<asp:label runat="server" font-size="Large" forecolor="HotPink" backcolor="LightBlue"
  font-italic="True"/>
<asp:button runat="server" font-bold="True" width="200px" height="50px"
  forecolor="IndianRed" backcolor="Blue"/>
<asp:textbox runat="server" font-bold="True" font-size="Medium" forecolor="Red"
  backcolor="Yellow" />
<asp:textbox SkinID="TextBox" runat="server" font-italic="True" font-size="Medium"
  forecolor="Yellow" backcolor="Red" />
```

In the preceding listing, we have used the `SkinID` property for a `TextBox` control. You can apply this theme only on the TextBox control referred to with the `SkinID` property.

6. Now, run the application and you will see the output in the Web browser, as shown in Figure 23.19:

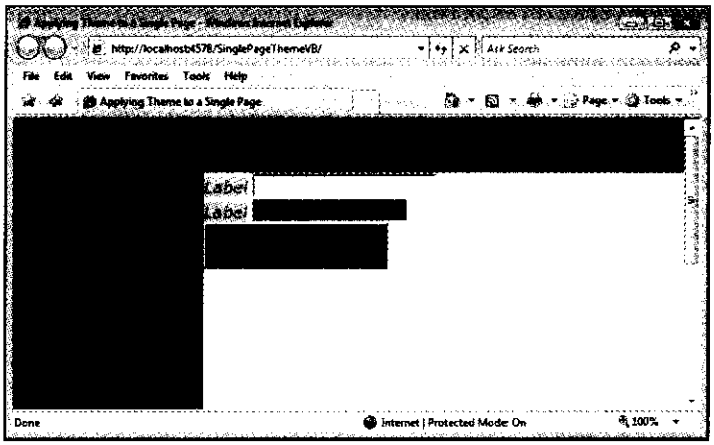


Figure 23.19: Output of the SinglePageTheme Application

In Figure 23.19, you can see that the colors of the second `TextBox` control are not the same as the first TextBox control. Due to the `SkinID` property, the theme set with this property is only applied to the control referred to with the `SkinID` property.

In place of using the `Themes` attribute in the page directive, you can use the `StyleSheetTheme` attribute to apply the themes to a Web page. Now, let's know about the `StyleSheetTheme` attribute.

Now, let's discuss the `StyleSheetTheme` attribute introduced by ASP.NET 3.5.

Understanding the StyleSheetTheme Attribute

The `StyleSheetTheme` attribute is an attribute of the page directive that you can use to apply themes to a Web page. Now, the question that arises is: why do we use the `StyleSheetTheme` attribute instead of using the Themes attribute when the Themes attribute is also used to apply themes to a Web page?

The benefit of using the `StyleSheetTheme` attribute instead of using the Themes attribute is that the theme applied by using the `StyleSheetTheme` attribute does not override the local attributes of the controls with the attributes defined in a theme. However, this overriding process cannot be carried out by using the `Themes` attribute, which overrides all the local attributes of the controls with the theme attributes.

Using the StyleSheetTheme Attribute

In the previous applications, we have used the Themes attribute to apply the theme to a Web page in a Web application. Now, we use the `StyleSheetTheme` attribute. Create a new Web application in the Visual Studio 2008, and name it `StyleSheetThemeVB`. You can find the code of `StyleSheetThemeVB` application in the Code\ASP.NET\Chapter 23\`StyleSheetThemeVB` folder on the CD. After creating the application, follow these steps to use the `StyleSheetTheme` attribute:

1. Create a `SkinFile.skin` file in the `Theme` folder, with the same procedure followed in the in previous applications.
2. Now, add the code shown in Listing 23.23, to the `SkinFile.skin` file:

Listing 23.23: Code for the `SkinFile.skin` File

```
<asp:label runat="server" width="350px" height="50px" font-bold="True" font-size="Large"
  forecolor="Red" backcolor="Blue"/>
<asp:button runat="server" forecolor="Blue" backcolor="Silver"/>
<asp:textbox runat="server" font-bold="True" font-size="Medium" forecolor="Blue"
  backcolor="Silver" font-italic="True"/>
```

In the preceding code, we have set the background and foreground colors of the controls that can be used in a Web page.

3. Add the code, as shown in Listing 23.24, in the `Default.aspx` page of the Web application:

Listing 23.24: Code for the `Default.aspx` Page

```
<%@ Page Language="VB" StylesheetTheme="Theme1" AutoEventWireup="false"
  CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>StyleSheetTheme Page</title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="form1" runat="server">
  <div>
      <div id="header">
      </div>
      <div id="sidebar">
          <div id="nav">
               
          </div>
      </div>
      <div id="content">
          <div class="itemContent">
          <div style="text-align: left">
          <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
```

```
            <br />
            <br />
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
            <br />
            <br />
            <asp:Button ID="Button1" runat="server" Text="Button" ForeColor="Aqua"
            BackColor="DarkViolet" />
            </div>
            <div id="footer">
            <p class="left">
                All content copyright &copy; Kogent Solutions Inc.</p>
            </div>
        </div>
    </div>
    </form>
</body>
</html>
```

In the `Default.aspx` page, we have added three controls: a Label control, a TextBox control, and a Button control. You can also add the controls from the Toolbar by drag and drop. We have specified the `foreground` and `background colors` locally for the Button control. We have also added the `StyleSheetTheme` attribute in the Page directive of the `Default.aspx` page.

4. Now, run the Web application and the output is shown in Figure 23.20:

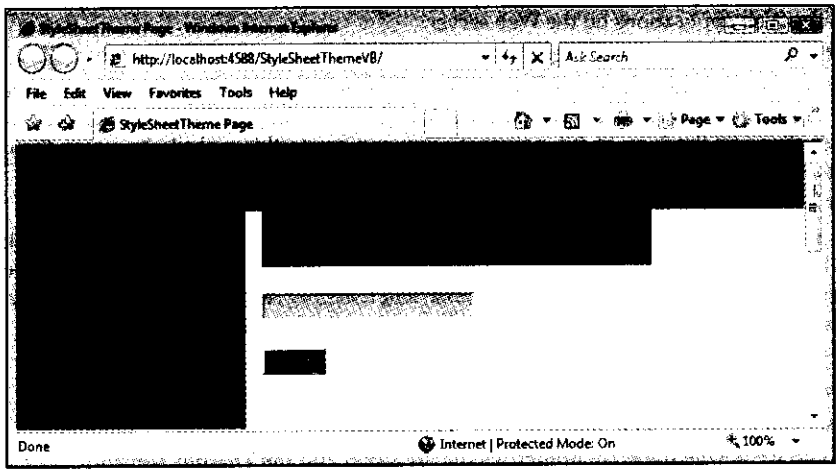


Figure 23.20: Output of the StyleSheetTheme Application

The foreground and background colors of the `Button` control (Figure 23.20) are specified locally in the `Default.aspx` page. As discussed earlier, the colors of the `Button` control are not overridden by the colors we specified in the theme (`SkinFile.skin`) file. This is done by using the `StyleSheetTheme` attribute. On the other hand, if the `Theme` attribute is used, then it will display those colors of the `Button` control that we have specified in the `SkinFile.skin` file.

Summary

In this chapter, we have learned some of the important aspects of master pages and themes, such as attributes, properties, and methods related to master pages and themes. This chapter also discusses the need of the master pages and themes, and also shows how to create and apply master pages and themes on Web pages. We have also discussed about the ASP.NET AJAX master pages and the `StyleSheetTheme` attribute.

In the next chapter, you learn about managing web applications.

Quick Revise

Q1. How does a content page differ from a master page?

Ans. A content page does not have complete HTML source code, whereas a master page has complete HTML source code inside its source file.

Q2. Why do we need nested master pages in a Web site?

Ans: When we have several hierarchical levels in a Web site, then we use nested master pages in the Web site.

Q3. How will you differentiate a submaster page from a top-level master page?

Ans: Like a content page, a submaster page also does not have complete HTML source code whereas a top-level master page has complete HTML source code inside its source file.

Q4. In which folder a skin file stores?

A. App_Data
B. App_Themes
C. App_Code
D. App_Browsers

Ans: C. App_Themes.

Q5. What is the difference between a page theme and a global theme?

Ans: A page theme is stored inside a subfolder of the `App_Themes` folder of a Web application and can be applied to individual Web pages of the Web application. Different page themes can be created to apply for the different Web pages in separate subfolders inside the `App_Themes` folder of the Web application. On the other hand, a global theme is stored inside the `Themes` folder on a Web server and can be applied to all the Web applications on the Web server.

Q6. What is the need for Master Pages?

Ans: Need for master pages arise when you want to create Web pages that contain some common elements. For example, you want to create a website for your company in which all the Web pages must have a header showing the logo of the company. You can create this website by using the concept of master pages.

Q7. What are the Page Directives of a Master Page's code?

Ans: The `MasterPage.master` page contains the `@ Master` directive, which is equivalent to the `@Page` directive of the ASP.NET applications. The `@ Master` directive is, respectively, as follows:

```
<%@ Master Language="VB" CodeFile="MasterPage.master.vb" Inherits="MasterPage" %>
```

Q8. A master page can contain only one ContentPlaceHolders. (True/False)

Ans. False

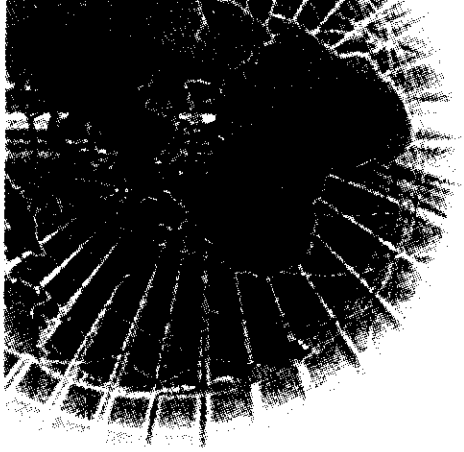
Q9. Can we apply multiple skin files on a single page?

Ans: Yes, we can apply multiple skin files on a single page.

Q10. How can we define the style properties for the Label and the TextBox control?

Ans: We can define the style properties for the controls in the skin file. Following is an example to define the style properties for the Label and the TextBox control:

```
<asp:label runat="server" width="350px" height="50px" font-bold="True" font-
size="Large" forecolor="Red" backcolor="Blue"/>
<asp:button runat="server" forecolor="Blue" backcolor="Silver"/>
<asp:textbox runat="server" font-bold="True" font-size="Medium"
forecolor="Blue" backcolor="Silver" font-italic="True"/>
```



24

Managing Web Applications

Web application management ensures that the Web applications remain error-free for a long time and also meet the end-user requirements. In the previous versions of ASP.NET, such as 1.x, and 2.x, managing a Web application was not an easy task. The developer was required to perform certain coding techniques to manage Web applications. However, performing coding techniques increases the work load and decreases the developer's productivity. Microsoft realized this fact and decided to introduce new tools, configuration files, and Application Programming Interfaces (APIs) for better management of Web applications. This was a welcome news for developers who were using exclusive coding techniques to manage Web applications.

In this chapter, you learn about the need for managing Web applications, ASP.NET configuration system, and Process model configurations. In addition, this chapter introduces various tools for managing Web applications, such as the Web Site Administration Tool, the `aspnet_regiis` tool, and the `aspnet_regsql` tool. This chapter also describes how to verify configuration settings by using the ProtectSection method, and configure ASP.NET applications in IIS.

Need to Manage Applications

You need to manage the applications to make your applications error free and extend the usability of your applications. For instance, suppose you create a Web application, and suddenly the application stops working due to some problem with the Web server and you are unable to connect to the SQL Server. What will you do? You will find these errors and rectify them.

When such a thing occurs at the user's end, the user will be unable to work till the time the website is up again. What should then be done to resolve such errors so that a Web application runs for a long time and is able to fulfill the end-users' requirements? The only solution is to manage the Web applications on a regular basis with the help of the ASP.NET configuration files so as to remove the errors and fulfill the requirements of the end user.

The ASP.NET Configuration File

The ASP.NET configuration system is a tool that includes XML-based files used to define settings for a Web application, or an individual page of an application. The two types of configuration files present in the ASP.NET configuration system are:

- `machine.config`—A server-specific configuration file
- `web.config`—An application-specific configuration file

The default configuration settings for all the Web applications of ASP.NET 3.5, are given in the `machine.config` file. You can find the `machine.config` file at the following location on the hard drive:

```
C:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG\machine.config
```

The application-specific configuration settings specified in the `web.config` file are applied to all the child directories. If a `web.config` file is present in the root directory of a Web application, then the configuration settings specified are applied to the entire Web application. ASP.NET Framework configures the IIS server in such a way that an error is returned when an attempt is made to open a `web.config` file through a browser. You can find the `machine.config` file at the following location on the hard drive:

```
C:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG\web.config
```

A `web.config` file consists of elements, such as tags, comments, and text with the `<configuration>` tag as their root element. The configuration settings of ASP.NET applications are included within the `<configuration>` tag. You can include the `<pages>` section under the `<system.web>` section group, which is nested inside the `<configuration>` tag. The `<pages >` section is used to define configuration settings for the ASP.NET compiler in the following code:

```
<configuration>
  <system.web>
      <pages enableViewState="false" />
  </system.web>
</configuration>
```

In the configuration sections, the collection settings, which are collections of elements, such as HTTP Handler and providers, are included. The <add> and <remove> tags are used for adding and removing elements from a configuration section, respectively. The <clear> tag removes the entire collection settings. In a web.config file, the configuration settings defined within the <configuration> tag apply to the entire current directory where the web.config file is located and also to all the child directories within this current directory. The <location> tag defined in a web.config file with the path attribute is used to apply configuration settings to a specific path directory only.

The <location> tag can also be used with the allowOverride attribute. This attribute, when set to false, ensures that the configuration settings in the current web.config file are not overridden by configuration settings specified in another configuration file. Further, an error is generated if a user tries to override the settings of the current web.config file in another configuration file.

The configuration settings specified within a web.config file are processed by configuration section handlers, which are .NET Framework classes inherited from the ConfigurationSection class.

After discussing about the configuration file, let's discuss verifying the configuration settings in ASP.NET.

Verifying the Configuration Settings

After you have configured settings for your Web application, you can also verify those settings. To verify the configuration settings of a Web application, you need to define a configuration object in the .aspx file. The configuration object allows you to collect information, such as the hierarchy of configuration sections and configuration section groups in a Web application. You can use the Configuration.RootSectionGroup property to obtain the root of the hierarchy in a configuration file. In addition, you can also use the Configuration.GetSection method to access a configuration section to verify the configuration settings.

The following code allows you to read all the section groups and sections in the configuration file of the Web application. To read the configuration settings from a Web application create a new Web application and name it as VerifiedSettingsVB. You can find the code of VerifiedSettingsVB application in the Code\ASP.NET\Chapter 24\VerifiedSettingsVB folder on the CD, and add the code, shown in Listing 24.1, in the Default.aspx page of the VerifiedSettingsVB Web application:

Listing 24.1: Showing the code for the Default.aspx Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<%@ Import Namespace="System.Configuration" %>
<%@ Import Namespace="System.Web.Configuration" %>
<%@ Import Namespace="System.IO" %>
<%@ Import Namespace="System.Xml" %>
<script runat="server" language="VB">
  Public Sub Page_Load(ByVal source As Object, ByVal e As EventArgs)
      Label1.Visible = False
      ConfigSections.Visible = False
  End Sub
  Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
      Label1.Text = TextBox1.Text
      Dim con As System.Configuration.Configuration =
      WebConfigurationManager.OpenWebConfiguration(TextBox1.Text)
      Dim textwriter As StringWriter = New StringWriter()
      Dim textXml As XmlTextWriter = New XmlTextWriter(textwriter)
      textXml.Formatting = Formatting.Indented
      textXml.WriteStartElement("configuration")
      Enumerate(con.RootSectionGroup, textXml)
      textXml.WriteFullEndElement()
      textXml.Flush()
      ConfigSections.Text = Chr(13) & Chr(10) &
      Server.HtmlEncode(textwriter.ToString())
```

```
        Label1.Visible = True
        ConfigSections.Visible = True
        Try
            Dim section As PagesSection = CType(con.GetSection("system.web/pages"),
            PagesSection)
            Label2.Text = section.SectionInformation.ToString()
            Label3.Text = section.EnableViewState.ToString()
            Dim sec As String = TextBox2.Text
            Dim Settings As ConfigurationSection = con.GetSection(sec)
            Label6.Text = Chr(13) & Chr(10) &
            Server.HtmlEncode(Settings.SectionInformation.GetRawXml())
            Label5.Text = TextBox2.Text
            Catch
            TextBox1.Text = "invalid section name"
        End Try
    End Sub
    Sub Enumerate(ByVal grp As ConfigurationSectionGroup, ByVal writeXML As XmlWriter)
        Dim x As Integer
        Dim configsection As ConfigurationSectionCollection = grp.Sections
        For x = 0 To configsection.Count - 1
            writeXML.WriteStartElement(configsection(x).SectionInformation.Name)
            writeXML.WriteEndElement()
        Next
    End Sub
</script>

<html>
<head>
    <title>Configuration Sections</title>
    <link href="Stylesheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <form id="mainForm" runat="server">
    <div>
        <div id="header">

        </div>
        <div id="sidebar" style="height: 1014px">
            <div id="nav">
                 
            </div>
        </div>
        <div id="content">
            <div class="itemContent">
            <h2>
            Enter the Virtual Path of the WebSite:</h2>
            <p>
             <asp:TextBox ID="TextBox1" runat="server" BorderStyle="Double"
            Width="464px"></asp:TextBox></p>
            <p>
            Enter section you need to verify        <asp:TextBox
            ID="TextBox2"
            runat="server" BorderStyle="Double" Width="201px"></asp:TextBox>
            </p>
            <p>
            <asp:Button ID="Button1" runat="server" Text="Verify Settings"
            OnClick="Button1_Click" /> </p>
            <h2>
            <asp:Label runat="server" ID="label1" /> </h2>
            <p>
```

```
                SectionInformation:   <asp:Label runat="server" ID="Label2"
                /> </p>
                <p>
                EnableViewState:      <asp:Label runat="server"
                ID="Label3" /></p>
                <p>
                        <asp:Label runat="server" ID="Label5" width="111px" /> 
                        <asp:Label runat="server" ID="Label6" /></p>
                <p>
                        ____________________________________________</p>
                <p>
                         <asp:Label runat="server" ID="ConfigSections" /></p>
                <pre>
                </pre>
                </div>
                <div id="footer">
                        <p class="left">
                                All content copyright &copy; Kogent Solutions Inc.</p>
                </div>
        </div>
    </div>
    </form>
</body>
</html>
```

The final output of the preceding listing is shown in Figure 24.1:

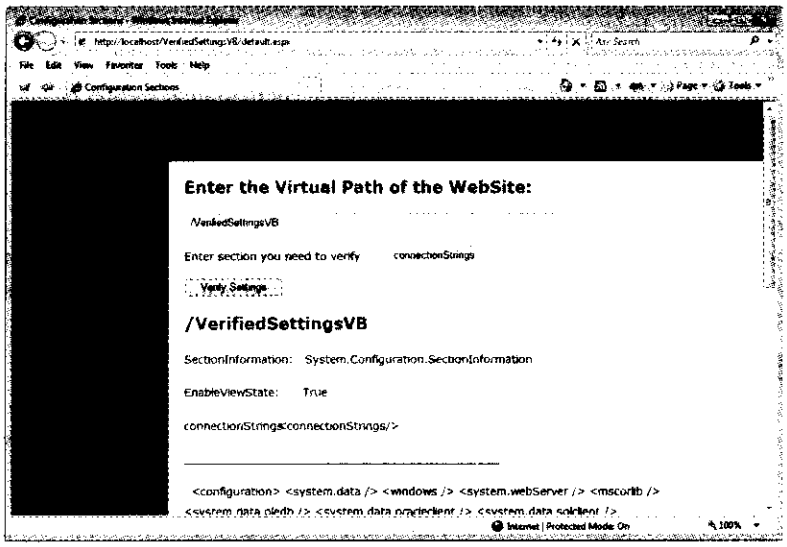


Figure 24.1: Showing Verified Settings

After verifying the settings that you have configured in the web.config file, let's understand the process model configuration in ASP.NET.

The Process Model Configuration

The Process model in ASP.NET protects the processes running on the server from being modified by the end user. The process model configuration settings determine the time for a new process to start serving the requests sent by the requesting client. The `<processModel>` configuration section in the `machine.config` file contains the process model settings. The process recycling feature of process model allows configuring an ASP.NET process so that a process restarts on certain conditions, such as after a specific interval of time or after the

performance of a process is degraded due to memory-leaks or deadlock conditions. The process model supports two types of recycling that determines the beginning of the new processes to replace the current processes. Two types of recycling are as follows:

- Reactive process recycling
- Proactive process recycling

Reactive Process Recycling

Reactive process recycling restarts a process when the process is not able to complete requests sent by the client computer. Various conditions, such as deadlocks and access violations, might cause a process to restart. To control the conditions causing process restart, use the following configuration settings:

- **requestQueueLimit**—Manages deadlock conditions
- **memoryLimit**—Manages memory leak conditions
- **shutdownTimeout**—Specifies the time to shut down a worker process

Proactive Process Recycling

Proactive process recycling restarts a process periodically even if there is no error in the process. In proactive process recycling, a process restarts after the completion of a specific number of requests or the time-out period. The various configuration settings that trigger this process are:

- **timeout**—Specifies a time limit in the `hr:min:sec` format that indicates the time after which a new worker process starts replacing the current process
- **idleTimeout**—Specifies a time limit in the `hr:min:sec` format that indicates the duration of inactivity after which the process shuts down
- **requestLimit**—Indicates the number of requests after which a new process replaces the current process

You can use the `logLevel` attribute in the `<processModel>` configuration section of the `machine.config` file. This writes process model events to the Windows event log when the process cycling events occur. The values of the `logLevel` attribute can be any one of the following:

- `All`—Indicates that all the process cycling events are logged in the Windows event log
- `None`—Indicates that no process cycling events are logged in the Windows event log
- `Errors`—Indicates that only unexpected or error events are logged in the Windows event log

Now, let's discuss the various tools for managing the Web applications in ASP.NET. The first tool is ASP.NET Microsoft Management Console (MMC) Snap-In Tool.

The ASP.NET MMC Snap-In Tool

ASP.NET MMC Snap-In is a management tool that enables you to modify the settings of the `web.config` and `machine.config` files. The MMC Snap-In tool helps in configuring the settings for the applications deployed on a Web server.

NOTE

Only the administrator can modify the settings of the web.config and machine.config files.

In IIS 7, the MMC Snap-In tool is included in the admin tool. The new admin tool of IIS 7 basically modifies the settings of the `web.config` and the `machine.config` files. This new IIS 7 admin tool also provides the following features:

- Does not require administrator authority to manage the individual sites
- Provides extensibility support so that we can extend the use of IIS 7 admin tool
- Provides better server health information, diagnostics, and more user information as compared to the information provided by the MMC Snap-In tool

After discussing the ASP.NET MMC Snap-In tool, let's move on to discuss another tool `aspnet_regiis`.

The ASPNET_REGIIS Tool

The `aspnet_regiis` tool is a command line tool in .NET Framework, which is available in the version-specific framework directory under the `Microsoft.NET\Framework` subdirectory of the Windows system folder. When multiple websites are hosted on a Web server and each of these websites is bound to a particular version of .NET Framework, the `aspnet_regiis` tool proves to be quite useful. This is because it is shipped with each version of the framework. To simplify the configuration process for an ASP.NET application, each version of ASP.NET comes with a linked version of the `aspnet_regiis` tool. The syntax of the `aspnet_regiis` tool is:

```
aspnet_regiis [options]
```

Table 24.1 shows various options that can be used with the `aspnet_regiis` tool:

Table 24.1: Options for the aspnet_regiis Tool

Option	Description
`-disable`	Disables the ASP.NET on the IIS Security Console, when used with `-i`, `-ir` or `-r` option.
`-?`	Lists all the options that can be used with the `aspnet_regiis` tool.
`-c`	Used to install the client-side scripts for the ASP.NET to the `aspnet_client` subdirectory of the IIS site directory (only those version of client-side script supported by the `aspnet_regiis` tool)
`-e`	Removes the client-side scripts for the current version of ASP.NET.
`-ea`	Removes the client-side scripts from the `aspnet_client` subdirectory of the IIS directory for all the versions available.
`-enable`	Enables ASP.NET in the IIS Security Console, when used with `-i`, `-ir` and `-r` option.
`-ga user`	Provides a user or a user group access to the IIS metabase and the directories used by ASP.NET.
`-i`	Installs the `aspnet_regiis` tool supported version of ASP.NET. It also updates the script maps of the applications that use previous versions of ASP.NET.
`-ir`	Installs the version of ASP.NET that supports the `aspnet_regiis` tool. However, if you use the `ir` option with the `aspnet_regiis` command, the script maps of the applications does not update. For this you need to the -i option.
`-k path`	Removes all the script maps from an ASP.NET application at a specified application path and the subdirectories of the root directory.
`-kn path`	Used to remove all the script maps from an ASP.NET application at a specified application root path and its subdirectories
`-lk`	Lists the path and version of IIS metabase keys. Inherited keys are not displayed.
`-lv`	Lists the status and installation path of all the versions of installed ASP.NET applications.
`-norestart`	Restricts the restart of the World Wide Web Publishing Service after installing or updating ASP.NET script maps. Without using this option, all applications pools are recycled.
`-r`	Installs the script maps for the ASP.NET version, which is supported by the `aspnet_regiis` tool, at the IIS metabase root and for all script maps below the root. The existing script maps are changed to this version, regardless of the current installed version of ASP.NET.
`-s path`	Installs the script map for the current ASP.NET version at the specified path and updates the existing script maps of previous versions with this version. The `-s` option updates the existing script maps to the current version.
`-sn path`	Installs the script map for this ASP.NET version at a specified IIS metabase path of the application.

Table 24.1: Options for the aspnet_regiis Tool	
Option	Description
-u	Uninstalls the version of ASP.NET that supports the aspnet_regiis tool.
-ua	Uninstalls all the versions of ASP.NET already installed on a computer.

Now let's discuss about how to use the ASPNET_REGIIS tool.

Using the ASPNET_REGIIS Tool

To use the aspnet_regiis tool, you need to first open the **Visual Studio 2008 Command Prompt**. Now, type the following command to view the list of all the options of the aspnet_regiis tool:

```
aspnet_regiis -?
```

Figure 24.2 shows the output after executing this command:

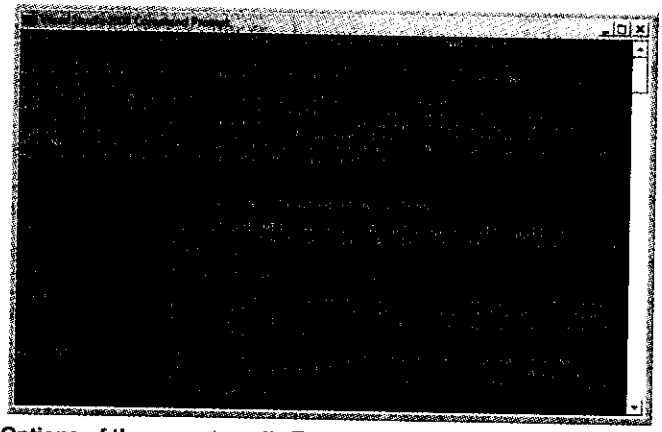


Figure 24.2: Options of the aspnet_regiis Tool in the Visual Studio 2008 Command Prompt

Now, let's see the execution of another option -lk with the aspnet_regiis command in the Visual Studio 2008 Command Prompt that allows you to view the path and version of the IIS metabase keys:

```
aspnet_regiis -lk
```

Figure 24.3 shows the output after executing this command:

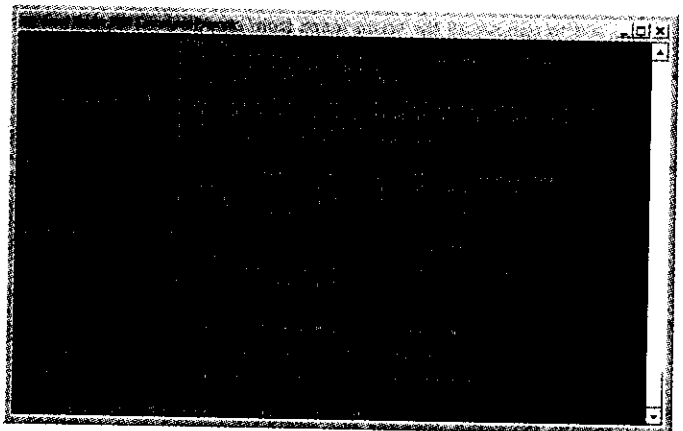


Figure 24.3: Path and Version of the IIS Metabase Keys in the Visual Studio 2008 Command Prompt

Now, let's use the ProtectSection method to secure the configuration sections of the web.config file.

Using the **ProtectSection** Method

You can use the `ProtectSection` method to encrypt the configuration sections of the `web.config` file. This method is used with the `SectionInformation` property to get the section information. Now, let's create an application named `ProtectedSectionVB` in which we use the `ProtectSection` method. You can find the code of `ProtectedSectionVB` application in the `Code\ASP.NET\Chapter 24\ProtectedSectionVB` folder on the CD. Perform the following steps to do so:

1. In the `Default.aspx` page, add the code to use the `ProtectSection` method to encrypt a connection string section of the configuration file, as shown in Listing 24.2:

Listing 24.2: Showing the code for the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<%@ Import Namespace="System.Configuration" %>
<%@ Import Namespace="System.Web.Configuration" %>
<script runat="server" language="VB">
  Protected Sub button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
  Handles button1.Click
        Const PROVIDER As String = "DataProtectionConfigurationProvider"
        Dim con As Configuration
        con = WebConfigurationManager.OpenWebConfiguration(Request.ApplicationPath)
        Dim sect As ConnectionStringsSection = con.ConnectionStrings
        sect.SectionInformation.ProtectSection(PROVIDER)
        con.Save()
        label1.Text = "Section of web.config is now encrypted<br>"
        label1.Text = label1.Text & "Connection String you have provided is:" &
        ConfigurationManager.ConnectionStrings("Pubs").ConnectionString
  End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>ProtectSection demo</title>
  <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="form1" runat="server">
  <div>
        <div id="header">
        </div>
        <div id="sidebar">
              <div id="nav">
                     
              </div>
        </div>
        <div id="content">
              <div class="itemContent">
              <br />
              <strong><span style="text-decoration: underline"><em>Use of
              ProtectSection Method
              <br />
              </em></span></strong>
              <br />
              <br />
              <asp:Button ID="button1" runat="server" Text="Click Me to Encrypt the
              Connection String"
              Width="384px" Height="35px" OnClick="button1_Click" Font-Size="Medium" />
              <br />
              <br />
              <br />
```

```
                <asp:Label ID="label1" runat="server" Height="19px"
                Width="435px"></asp:Label>
                </div>
                <div id="footer">
                <p class="left">
                        All content copyright &copy; Kogent Solutions Inc.</p>
                </div>
            </div>
        </div>
        </form>
    </body>
    </html>
```

2. Replace the `<connectionStrings>` section of the `web.config` file with the following code:

```
<connectionStrings>
  <add name="Pubs" connectionString="server=localhost;database=pubs;
  integrated security=true;"/>
</connectionStrings>
```

3. Now, run this application. A window similar to Figure 24.4 is displayed:

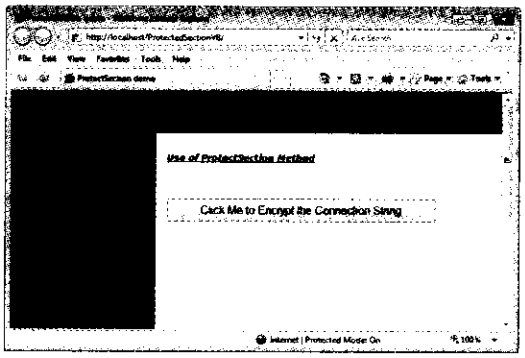


Figure 24.4: ProtectedSectionVB Example at Work

4. Click the Click Me to Encrypt the Connection String button on the `Default.aspx` page to encrypt the `web.config` file connection string section. Figure 24.5 shows the output after clicking the button:

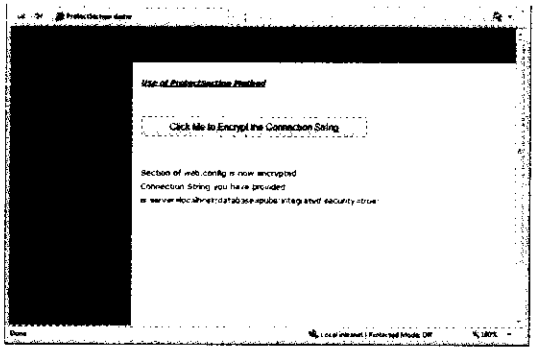


Figure 24.5: Default.aspx with Config Settings

Now, when you open the `web.config` file, you are presented with the following encrypted data in the `<connectionStrings>` section:

```
<connectionStrings configProtectionProvider="DataProtectionConfigurationProvider">
  <EncryptedData>
        <CipherData>
                <CipherValue>AQAAANCMnd8BFdERjHoAwE/Cl+
                sBAAAA1AQCUYJi80mLY9VzjtRHMAQAAAACAAAAAAADZgAAqAAAABAAAAC65AUb6hEDIKK1Jr
```

```
P+iF+uAAAAAASAAACgAAAAEAAAAPOtZY9B02jnZAY5gHPlsdEgAQAA4FBClsFwKsV4328wN
qUxod6/iLjmoUq1qzkAOls/P3QuKwNMPzIxNgIi5wczGMIYhUMsJ/lrbTsfr/2xrIWv+mLcuk
leAPSQwwbfYUBz8yIw8R650QjIFeqgYPv58GTp+U6uS7d4ofv8HiFuqmXHCTMKnRUBq3n1/
DuRgpk5RjYbFhEAUrgxjZa0Z4mnczRZ9Pz7bk4J6QzBLvBVh1FvxTgKi1utVxzhnvafr
3NPSMscx7NbwKupEhO1KzF7zt/axNp3zox9jYwbNYsYhA0TruduKv1BGEyVCxpC+1DPadgB
SwSrMZIKelbvIXinmkq+Sb6WzyFgM6jRHr73hk/rGsDtPEGtAqxA+aRYFSpoxrhQ/gDSQOpzI
E4ny3iv58+hFAAAAHRxQrz51Q6XulNVmkeZraA0Usfw</CipherValue>
        </CipherData>
    </EncryptedData>
</connectionStrings>
```

NOTE

You must give write permission to the web.config file for the ASP.NET account to run the above code. You can also use the aspnet_regiis tool to encrypt the configuration files.

Now, let's explore the Web Site Administration Tool to manage ASP.NET Web applications.

The Web Site Administration Tool

The Web Site Administration Tool is used to configure the application settings and security settings of an ASP.NET Web application. The other tasks performed using the Web Site Administration Tool are as follows:

- Configuring the application security settings, such as authentication and authorization
- Managing users and roles for an application
- Creating and managing application settings
- Configuring Simple Mail Transfer Protocol (SMTP) e-mail settings
- Making an application temporarily offline to update its content and data in a secure manner
- Adjusting the debugging and tracing settings
- Defining a default custom error page
- Selecting providers to use them with the site features, such as membership

Using the Web Site Administration Tool

To use the Web Site Administration Tool, open a Web application in Visual Studio 2008 and select `Website→ASP.NET Configuration` from the menu bar. The ASP.NET Web Site Administration Tool page appears in the ASP.NET Web Application Administration - Windows Internet Explorer window, as shown in Figure 24.6:

Figure 24.6: ASP.NET Web Site Administration Tool Page

In Figure 24.6, the user interface (UI) of the Web Site Administration tool consists of different tabs that can be used to group the configuration settings of a Web application.

The various tabs on the Web Site Administration page are:

- **Home tab**—Enables to return to the home page of the Web Site Administration tool.
- **Security tab**—Manages the user accounts associated with websites that are accessing a Web application, creates roles for the users, and provides different access rules for the users. For example, you might create roles, such as managers and members, and provide different access rights to specific folders.
- **Application tab**—Manages the settings to determine how a Web application sends an e-mail. The settings are, for example, SMTP and other application settings. In addition, the settings also enable you to determine the online and offline settings of a Web application.
- **Provider tab**—Configures providers for storing website management data.

Using the Security Tab

The Security tab of the Web Site Administration Tool enables you to manage user accounts, roles, and rules for accessing Web applications. The page that opens when you select the Security tab of the Web Site Administration Tool is shown in Figure 24.7:

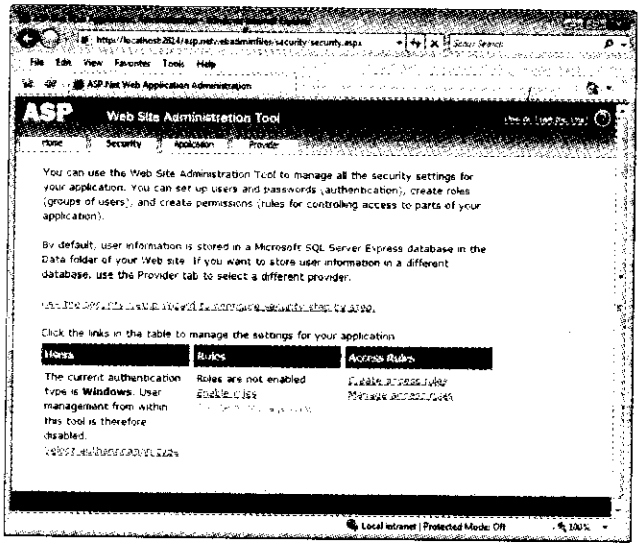


Figure 24.7: Security Tab of the Web Site Administration Tool

NOTE

By default, the Roles on the Security page are disabled and the authentication type is windows.

Various panes in the Security page, as shown in Figure 24.7, are as follows:

- Users pane
- Roles pane
- Access Rules pane

The Users pane of the Security page enables you to configure a Web application to use either the Forms-based authentication or Integrated Microsoft Windows authentication method. The Forms-based authentication method is used for Web applications that are accessed over the Internet. However, the Integrated Microsoft Windows authentication method is used when the users access Web applications over a local network. When the Integrated Microsoft Windows authentication method is used, the users can automatically log on to Web applications after logging on to a Windows-based network.

Now, follow the steps to work with the Security tab:

1. Click the Select authentication type link in the Users pane of the Security page to open a page containing two options, From the internet and From a local network.
2. Select an option on the page to indicate if the Web application can be accessed over the Internet or from a local network. In our case, we have selected the From the internet option, as shown in Figure 24.8:

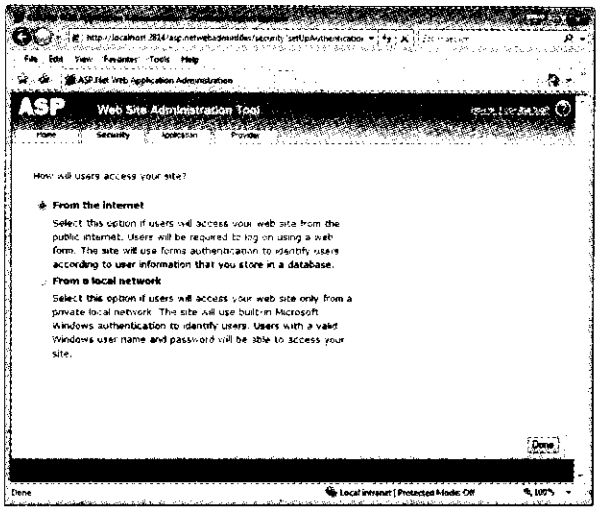


Figure 24.8: The Authentication Type Link Page

3. Click the Done button at the bottom of the page to apply the settings. The Create user and Manage users link appears, as shown in Figure 24.9:

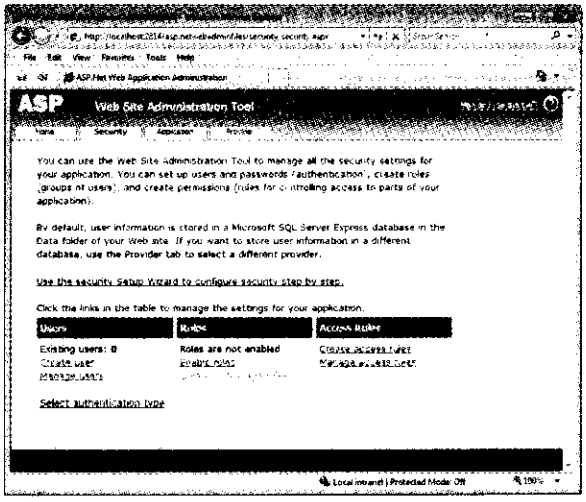


Figure 24.9: The Create User and Manage Users Link

4. Click the Create user link on the Users pane of the Security page to specify a user account for the new user of the Web application (Figure 24.10):

NOTE

You can click the manage user link to manage (add, modify, or delete) the existing users.

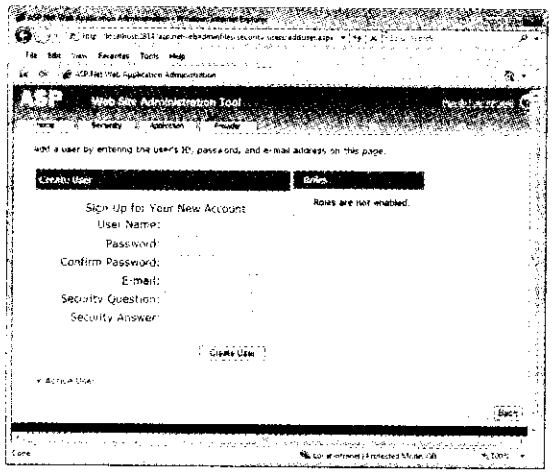


Figure 24.10: The Create User page

Figure 24.10 shows various fields that need to be filled:

- **User Name**—Specifies the name of a user for creating a user account.
- **Password**—Specifies a password for the user name. The password specified is case-sensitive.
- **Confirm Password**—Confirms the password again for the user.
- **E-mail**—Provides the e-mail address of the new user.
- **Security Question**—Specifies a question that should appear when the user needs to reset or recover the password.
- **Security Answer**—Specifies an answer to the security question given in the preceding box.
- **Active User**—Enables the user account created as the active (current) user of the Web application.
- **Roles**—Indicates the role for the user account created. The roles for the user accounts are created separately by using the Roles section of the Security page.

5. Enter the new user information and click the Create User button to accept the new user information.
6. You can use the Roles section of the Security page to create roles for the users accessing the Web application. To create roles, click the Enable roles under the Roles pane of the Security page. When you click the Enable roles link, the name of the link changed to Disable roles and also enables the Create or Manage roles links, as shown in Figure 24.11:

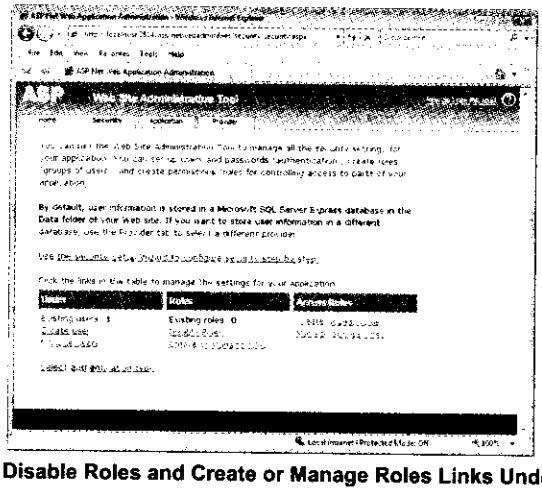


Figure 24.11: The Disable Roles and Create or Manage Roles Links Under the Roles Pane

7. Click the Create or Manage roles link under the Roles pane of the Security page to display a page that allows you to create a new role, as shown in Figure 24.12:

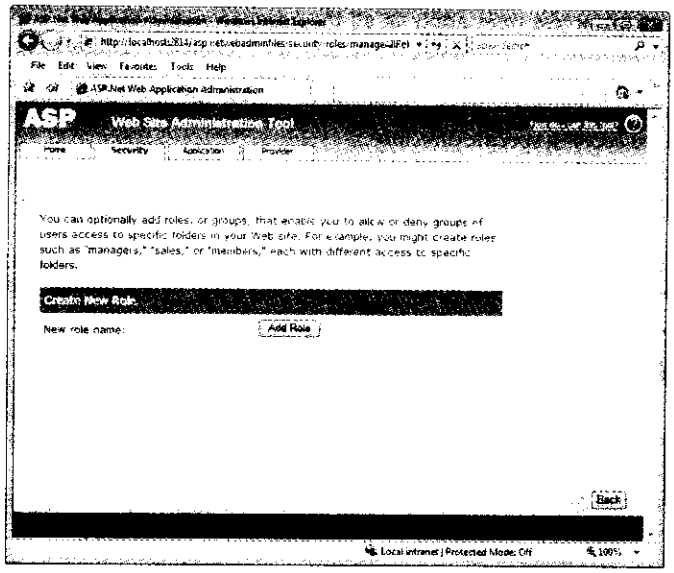


Figure 24.12: Page for Creating a New Role

8. Enter the new role name and click the Add Role button to add a role for the user of the Web application after specifying a new role name.
9. You can use the Access Rules pane in the Security page to create access rules for the users. Click Create access rules under the Access Rules pane of the Security page to open a page, as shown in Figure 24.13:

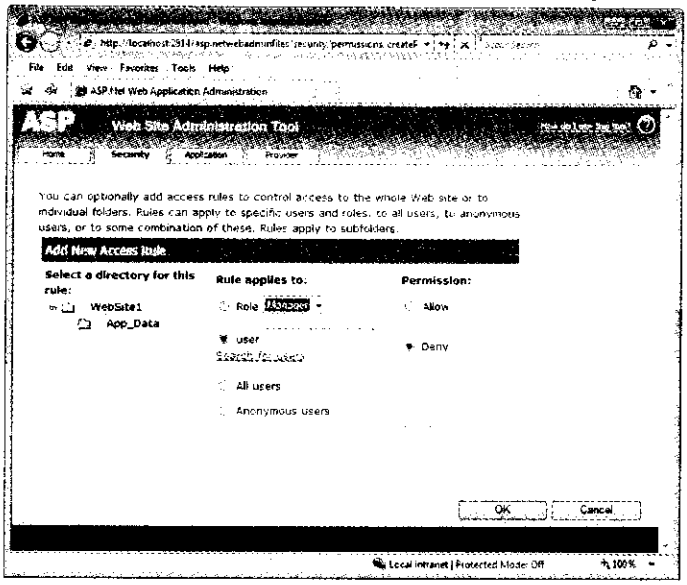


Figure 24.13: Adding New Access Rule Page

Figure 24.13 shows various fields in the page that assist in adding a new access rule:

- **Select a directory for this rule** – Enables the user to select the rules that are created either for the entire Web application or for only a specific directory.

- **Rule applies to**—Enables the user to select a role for which the access rules are applied. You can select the user option to enter the user account for which the access rules are specified. You can also select the All users option so that the access rules apply to all the users accessing the Web application. In addition, the Anonymous users option allows you to apply access rules to the non-registered user accounts only.
- **Permission**—Enables you to select either the Allow or Deny option. The Allow option helps a specified user account or role to access a specific directory. The Deny option disallows a user account or role indicated in the previous fields to access a specific directory.

10. Click the OK button to close the page and apply the settings specified in the page. After applying the settings, you will be redirected to the Home tab.

NOTE

You can click the Manage access rules link in the Access rules pane of the Security page to modify the access rules created earlier.

Using the Application Tab

The Application tab of the Web Site Administration Tool enables you to modify application-specific configurations, such as `appSettings`, `Debugging`, and `Tracing`. Figure 24.14 shows the Application page of the Web Site Administration Tool:

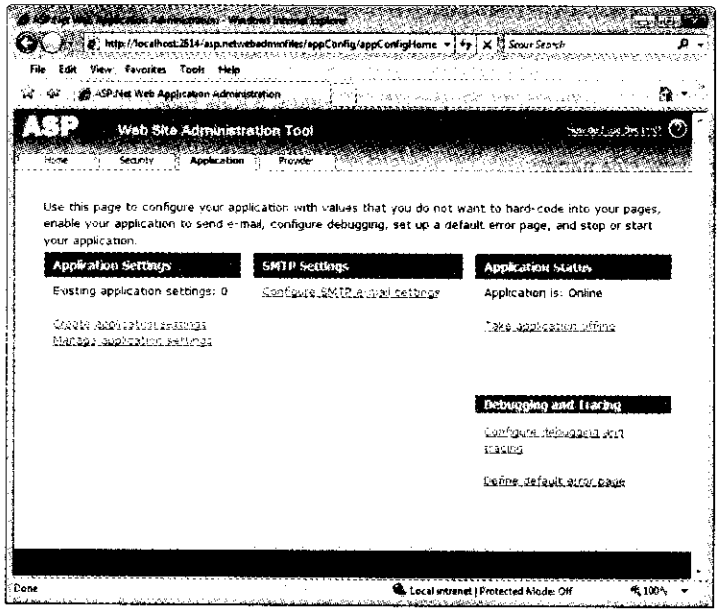


Figure 24.14: The Application Page

The Applications page consists of the following four panes:

- Application Settings
- SMTP Settings
- Application Status
- Debugging and Tracing

The configuration settings of an ASP.NET application can be modified using the Application page. In the Application Settings pane, click the Create application settings link. It opens a page that allows you to specify values for the Name and Value fields. Figure 24.15 shows the Create Application Settings page:

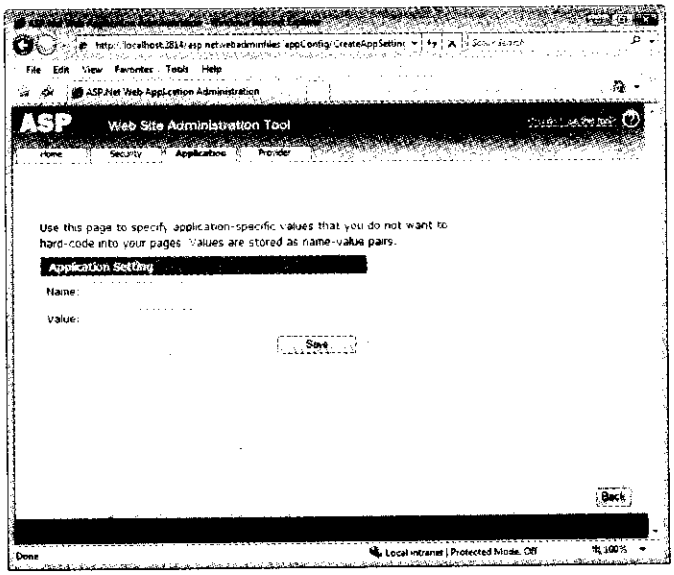


Figure 24.15: Specifying Application Settings

Specify the application settings and click the Save button to accept the specified settings.

NOTE

You can also manage the Application settings by clicking the Manage application settings link in the Application Settings pane that allows you to view the existing settings and create new application settings, as shown in Figure 24.16:

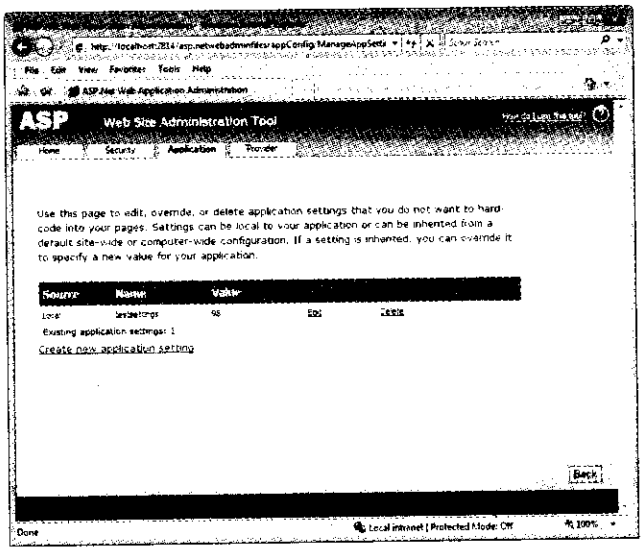


Figure 24.16: Managing Application Settings

In addition, the page also allows you to create new settings. You can modify the application status by clicking the Take application (offline link to take the Web application offline) in the Application Status pane.

The debugging settings of a Web application can also be edited by clicking the Configure debugging and tracing link to display a page in the Debugging and Tracing pane, as shown in Figure 24.17:

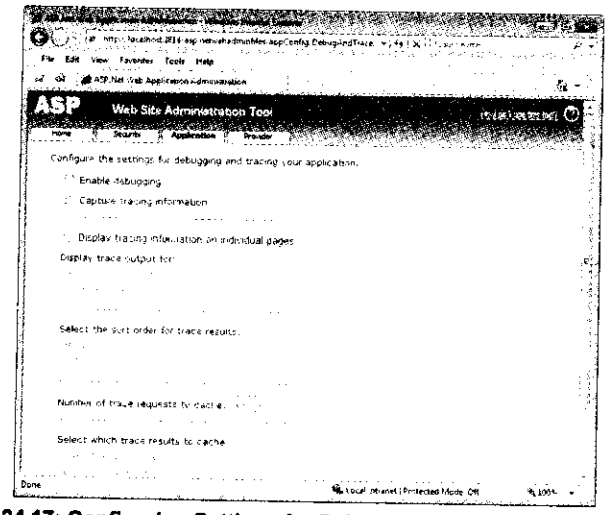


Figure 24.17: Configuring Settings for Debugging and Tracing Application

In Figure 24.17, select the Enable debugging option to enable debugging. Select the Capture tracing information option to display tracing information for the Web application. You can also define the default error page by clicking the Define default error page link in the Debugging and Tracing pane (Figure 24.18):

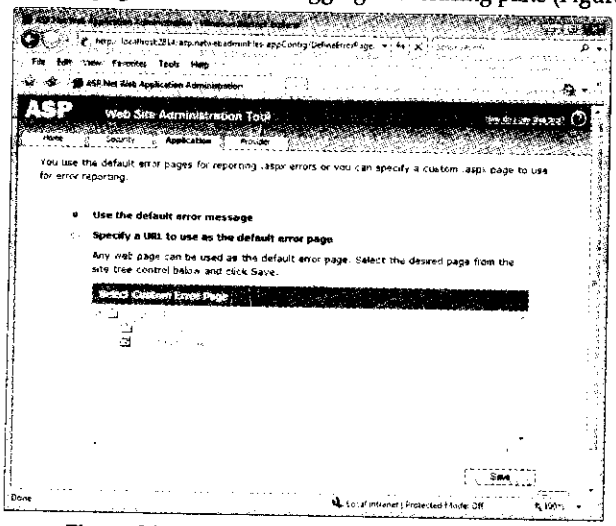


Figure 24.18: Displaying the Default Error Page

NOTE

In Figure 24.18, either select a default error message or select a URL, which is used for redirection in case an error condition occurs.

Sometimes, the SMTP settings need to be configured for Web applications to send Web data by using SMTP. You can configure SMTP settings by clicking the Configure SMTP e-mail settings link in the SMTP Settings pane of the Application tab. When you open the Configure SMTP Settings pane, you will be asked to configure the following:

- **Server Name**—Specifies the server name for the e-mail server
- **Server Port**—Specifies the port used for sending mails

- **From**—Specifies the e-mail address of the sender
- **Authentication**—Specifies the authentication type required for sending mails

Using the Provider Tab

The Provider tab of the Web Site Administration Tool enables you to manage the data stored in the database, such as user accounts and roles. You can change and test providers such as `AspNetSqlRoleProvider`, `AspNetWindowsTokenRoleProvider`, `AspNetSqlMembershipProvider`, and `AspNetSqlProfileProvider` used for the Web application. The Web Site Administration Tool uses the `AspNetSqlProvider` provider, by default, for all the Web applications. Figure 24.19 shows the page that opens when you click the Provider tab of the Web Administration tool:

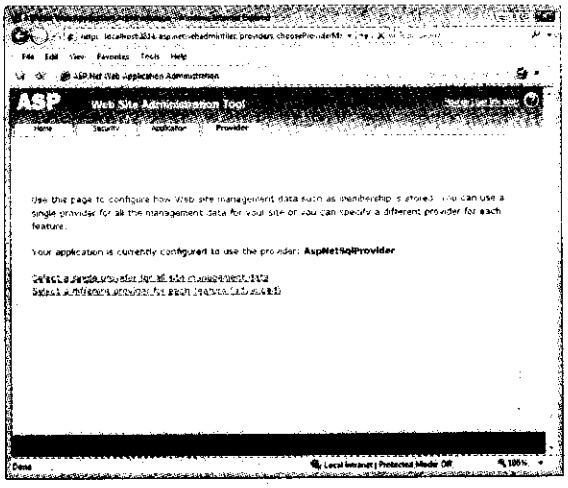


Figure 24.19: The Provider Page

You can use the Web Site Administration Tool to manage providers for the website in the following two ways:

- Changing the default `AspNetSqlRoleProvider` provider to an alternate `AspNetWindowsTokenRoleProvider` provider
- Specifying whether you want to use the same provider for all features, or use different providers for different features

After using the Web Site Administration Tool of ASP.NET, let's explore the `Aspnet_regsql` tool.

The **Aspnet_regsql** Tool

The `Aspnet_regsql` tool is a command line tool used to create a Microsoft SQL Server database. The Microsoft SQL Server database that is created with the help of the `Aspnet_regsql` tool is used by SQL Server providers in ASP.NET and is also used for adding or removing options from the existing database. The following is the syntax for using the tool:

```
Aspnet_regsql.exe <options>
```

`Aspnet_regsql` tool options are mentioned in Table 24.2:

Table 24.2: Aspnet_regsql Tool Options

Option	Description
`-?`	Displays the Help text for the `Aspnet_regsql` tool
`-C <connection string>`	Specifies the connection string
`-E`	Authenticates the user currently logged in, with current credentials provided by Windows

Table 24.2: Aspnet_regsql Tool Options	
Option	**Description**
`-P <password>`	Specifies the password for SQL server, with `-U` option
`-S <server>`	Specifies the name of the computer where the SQL server database is installed or need to be installed
`-sqlexportonly <filename>`	Generates an SQL script file that we can use to add or remove the specified features in the ASP.NET application
`-U <login ID>`	Specifies the user name for SQL Server, with `-P` option
`-W`	Runs the `Aspnet_regsql` tool in the wizard mode

Using the **Aspnet_regsql** Tool

To use the `Aspnet_regsql` tool, open the Visual Studio 2008 Command Prompt window and write the following command to view the list of all options of the `Aspnet_regsql` tool:

```
Aspnet_regsql.exe -?
```

The output of the above command is shown in Figure 24.20:

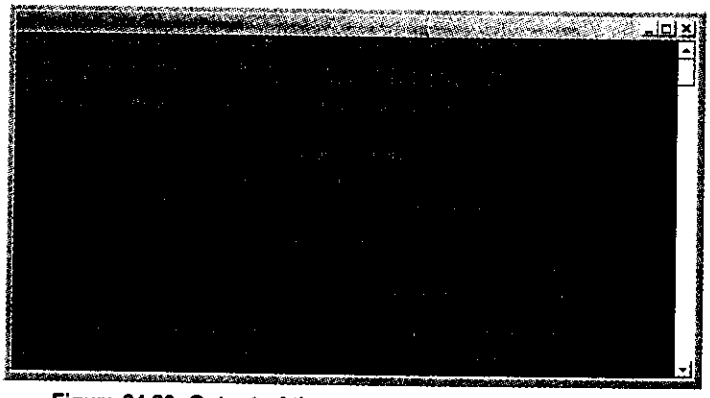


Figure 24.20: Output of the aspnet_regsql.exe -? Command

The following command can also be used to run the `Aspnet_regsql` tool in the wizard mode:

```
Aspnet_regsql.exe -W
```

The output of the preceding command displays the ASP.NET SQL Server Setup Wizard. This wizard enables you to configure the SQL Server database, as shown in Figure 24.21:

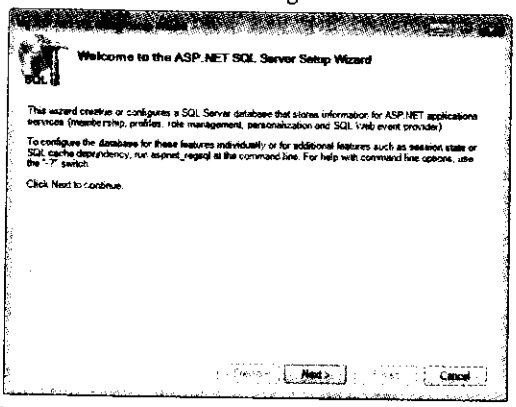


Figure 24.21: The ASP.NET SQL Server Setup Wizard

The wizard displayed in Figure 24.21 can be used to configure the SQL Server database for the required application services. When you click the Next button, you are presented with two options, either to configure the SQL Server for application services or to remove the application services from an existing database. Select the options according to your requirement and configure the database for the required services.

After using the tools for ASP.NET for managing Web applications, let's discuss configuring ASP.NET applications in the IIS.

Configuring ASP.NET Applications in IIS

Nowadays, configuring ASP.NET applications has become quite easy with the help of IIS Manager. You can open the IIS Manager window by entering `inetmgr` in the Run dialog box. Figure 24.22 shows the IIS Manager window:

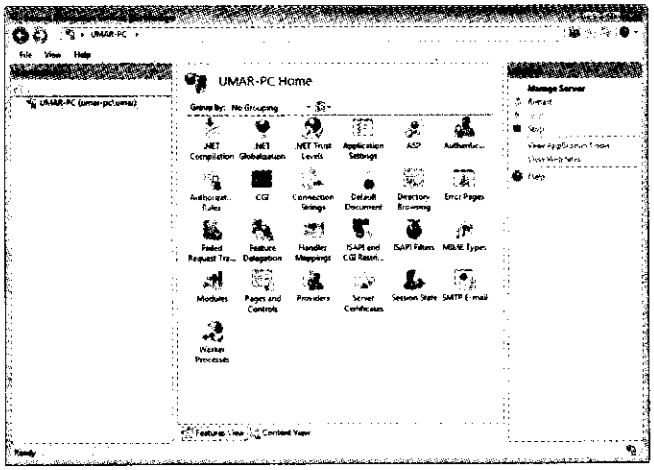


Figure 24.22: IIS Manager Window

Expand the nodes on the left pane of the IIS Manager window. You will find a Web Sites node, which contains all the sites configured to work with IIS. In the Web Site node, right-click the website that you want to configure. You notice that the options available to you for the configuration of ASP.NET applications will appear in the IIS Manager window, as shown in Figure 24.23:

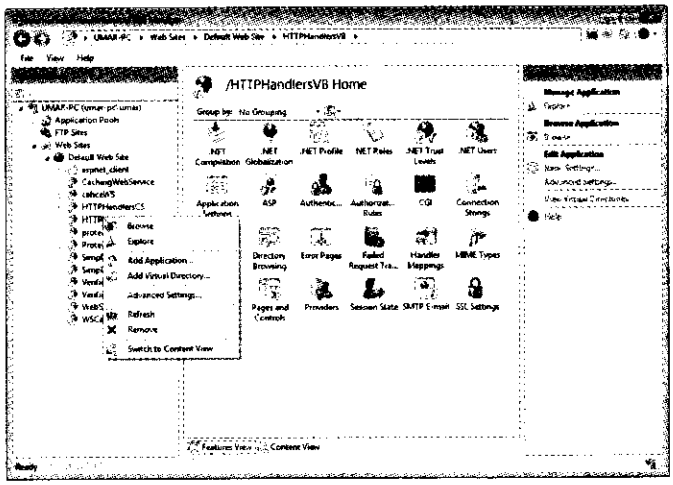


Figure 24.23: Configuration Options in the IIS Manager Window

With the options available in the IIS Manager window, you can completely configure ASP.NET applications or even the IIS itself. You can also select the actions to be performed from the Actions pane on the right side of the IIS Manager window. You can even configure some of the basic settings of the sites by clicking the Basic Settings link in the Actions pane. An Edit Application dialog box appears, as shown in Figure 24.24:

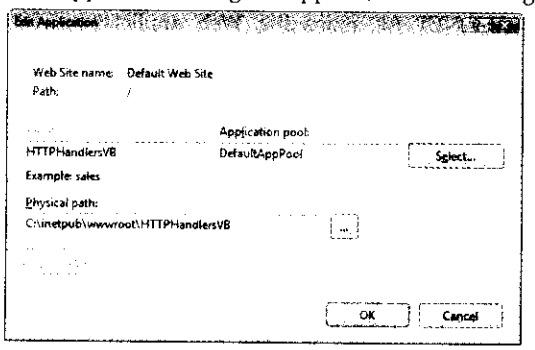


Figure 24.24: The Edit Application Dialog Box

The Edit Application dialog box displays the following items:

- The name of the website.
- The Application pool, which is used for the application. There are two options in the Application pool, `DefaultAppPool` (which uses .NET Framework 2.0 and integrated pipeline mode) and `Classic .NET AppPool` (which uses .NET Framework 2.0 and classic pipeline mode).
- The Physical path, which contains the location of the ASP.NET application.

In addition to the basic settings of the applications, some options provided to you through the icons in the IIS Manager window are as follows:

- **.NET Compilation**—Specifies the compilation and running process of the application. You can also define the default language that is used in the compilation process, such as VB.
- **.NET Globalization**—Enables in specifying the way in which the ASP.NET application is expected to deal with the culture and the encoding of the requests and responses, such as **UTF-8, UTF-6, and US**-ASCII.
- **.NET Profile**—Specifies the way in which the ASP.NET application is expected to deal with the ASP.NET personalization system. You can also specify the provider that is used by the system. By default, `AspNetSqlProfileProvider` provider is used.
- **.NET Roles**—Adds the roles to the ASP.NET application, which is used in the role-based management.
- **.NET Trust Levels**—Specify the level of security applied to the application through specific, pre-generated configuration file, such as, `web_hightrust.config`, `web_mediumtrust.config`, and `web_lowtrust.config`. By default, the application uses the `web.config` file.
- **.Net Users**—Enable the ASP.NET application to work with the ASP.NET membership system. You can add a user by providing certain details, such as username, password, security question and its answer, and the role of the user.
- **Connection Strings**—Add connection strings to your ASP.NET application. The default connection string used by the ASP.NET application is `LocalSqlServer`. You can also modify or delete the existing connections strings.
- **Pages and Controls**—Specify the settings that control ASP.NET pages (`.aspx`) and user controls (`.ascx`) in the application.
- **Providers**—Specify the providers for the ASP.NET application. There are two providers, `AspNetSqlRoleProvider` and `AspNetWindowsTokenRoleProvider`, defined for the .NET Roles engine. The `AspNetSqlMembershipProvider` provider is defined for.NET users, and the `AspNetSqlProfileProvider` provider is defined for the .NET profile. All these providers are specified by default.

- **Session State** – Enables you to determine the administration of the state management for the ASP.NET application. You can apply the state management to the application in a number of ways that includes Session state mode, Cookieless mode, and Session timeout mode.
- **SMTP E-mail** – Enables you to work with the applications that deliver e-mails.

These are some of the options through which you can configure your ASP.NET application in IIS and manage and make your Web applications error free.

Summary

In this chapter, we have learned how to use various tools (`ASPNET_REGIIS`, `ASPNET_REGSQL`, and ASP.NET Web Site Administration tools) for managing Web applications. Further, we have also verified the configuration settings of Web applications and used the `ProtectSection` method to protect Web applications. We have also learned how to configure ASP.NET applications in IIS.

In the next chapter, you learn about Application Globalization.

Quick Revise

Q1. Define web.config and machine.config configuration files?

Ans: The machine.config file is a server-specific configuration file, whereas a web.config file is an application-specific configuration file. The default configuration settings for all the Web applications of ASP.NET 3.5, are given in the machine.config file. The application-specific configuration settings specified in the web.config file are applied to all the child directories. If a web.config file is present in the root directory of a Web application, then the configuration settings specified are applied to the entire Web application.

Q2. What is process model in ASP.NET?

Ans: The Process model in ASP.NET protects the processes running on the server from being modified by the end user. The process model configuration settings determine the time for a new process to start serving the requests sent by the requesting client.

Q3. What is the difference between reactive process recycling and proactive process recycling?

Ans: Reactive process recycling restarts a process when the process is not able to complete requests sent by the client computer. Various conditions, such as deadlocks and access violations, might cause a process to restart whereas Proactive process recycling restarts a process periodically even if there is no error in the process. In proactive process recycling, a process restarts after the completion of a specific number of requests or the time-out period.

Q4. What is ASP.NET MMC Snap-In tool?

Ans: ASP.NET MMC Snap-In is a management tool that enables you to modify the settings of the web.config and machine.config files. The MMC Snap-In tool helps in configuring the settings for the applications deployed on a Web server.

Q5. What is ASPNET_REGIIS tool?

Ans: The aspnet_regiis tool is a command line tool in .NET Framework, which is available in the version-specific framework directory under the Microsoft.NET\Framework subdirectory of the Windows system folder. When multiple websites are hosted on a Web server and each of these websites is bound to a particular version of .NET Framework, the aspnet_regiis tool proves to be quite useful. This is because it is shipped with each version of the framework. To simplify the configuration process for an ASP.NET application, each version of ASP.NET comes with a linked version of the aspnet_regiis tool.

Q6. The ProtectSection method is used to decrypt the configuration sections of the web.config file. (True/False)

Ans: False

Q7. What is ASP.NET Web Site Administration tool?

Ans: The ASP.NET Web Site Administration Tool is used to configure the application settings and security settings of an ASP.NET Web application.

Q8. Define Aspnet_regsql tool?

Ans: The Aspnet_regsql tool is a command line tool used to create a Microsoft SQL Server database. The Microsoft SQL Server database that is created with the help of the Aspnet_regsql tool is used by SQL Server providers in ASP.NET and is also used for adding or removing options from the existing database.

Q9. Describe various tabs available on the Web Site Administration page?

Ans: The tabs available on the Web Site Administration page are as follows:

- **Home tab**—Enables to return to the home page of the Web Site Administration tool.
- **Security tab**—Manages the user accounts associated with websites that are accessing a Web application, creates roles for the users, and provides different access rules for the users. For example, you might create roles, such as managers and members, and provide different access rights to specific folders.
- **Application tab**—Manages the settings to determine how a Web application sends an e-mail. The settings are, for example, SMTP and other application settings. In addition, the settings also enable you to determine the online and offline settings of a Web application.
- **Provider tab**—Configures providers for storing website management data.

Q10. Which attribute in the <processModel> configuration section of the machine.config file writes process model events to the Windows event log when the process cycling events occur?

Ans: logLevel attribute