ASP .Net

No.	Topic Name
Topic-1	Building ASP.NET Pages
Topic-2	Building Forms with Web Server Controls
Topic-3	Performing Form Validations with Validation Controls
Topic-4	Advanced Control Programming
Topic-5	Introduction to ADO.NET
Topic-6	Binding Data to Web Controls
Topic-7	Using the Repeater, DataList and DataGrid Controls
Topic-8	Working with DataSets
Topic-9	Working with XML
Topic-10	Using ADO.NET to Create a Search Page
Topic-11	Creating ASP.NET Applications
Topic-12	Tracking User Sessions
Topic-13	Caching ASP.NET Applications
Topic-14	Application Tracing and Error Handling

Building ASP.NET Pages

ASP.NET 2.0 is the latest version of ASP, and it represents the most dramatic change yet. With ASP.NET, developers no longer need to paste together a jumble of HTML and script code in order to program the Web. Instead, you can create full-scale web applications using nothing but code and a design tool such as Visual Studio 2005. The cost of all this innovation is the learning curve.

Server-Side Programming

To understand why ASP.NET was created, it helps to understand the problems of other web development technologies. With the original CGI standard, for example, the web server must launch a completely separate instance of the application for each web request. If the website is popular, the web server must struggle under the weight of hundreds of separate copies of the application, eventually becoming a victim of its own success.

Client-Side Programming

At the same time that server-side web development was moving through an alphabet soup of technologies, a new type of programming was gaining popularity. Developers began to experiment with the different ways they could enhance web pages by embedding multimedia and miniature applets built with JavaScript, DHTML (Dynamic HTML), and Java code. These client-side technologies don't involve any server processing. Instead, the complete application is downloaded to the client browser, which executes it locally.

The greatest problem with client-side technologies is that they aren't supported equally by all browsers and operating systems. One of the reasons that web development is so popular in the first place is because web applications don't require setup CDs, downloads, and other tedious (and error-prone) deployment steps. Instead, a web application can be used on any computer that has Internet access. But when developers use client side technologies, they encounter a few familiar headaches. Suddenly, cross-browser compatibility becomes a problem. Developers are forced to test their websites with different operating systems and browsers, and they might even need to distribute browser updates to their clients. In other words, the client-side model sacrifices some of the most important benefits of web development.

ASP.NET allows you to combine the best of client-side programming with server-side programming. For example, the best ASP.NET controls can intelligently detect the features of the client browser. If the browser supports JavaScript, these controls will return a web page that incorporates JavaScript for a richer, more responsive user interface. However, no matter what the capabilities of the browser, your code is always executed on the server.

ASP.NET deals with these problems (and many more) by introducing a completely new model for web pages. This model is based on a remarkable piece of technology called the .NET Framework.

❑ The .NET Framework

You should understand that the .NET Framework is really a cluster of several technologies:

The .NET languages: These include C# and VB .NET (Visual Basic .NET), the object oriented and modernized successor to Visual Basic 6.0; these languages also include JScript .NET (a server-side version of JavaScript), J# (a Java clone), and C++ with Managed Extensions.

The CLR (Common Language Runtime): The CLR is the engine that executes all .NET programs and provides automatic services for these applications, such as security checking, memory management, and optimization.

The .NET Framework class library: The class library collects thousands of pieces of prebuilt functionality that you can "snap in" to your applications. These features are sometimes organized into technology sets, such as ADO.NET (the technology for creating database applications) and Windows Forms (the technology for creating desktop user interfaces).

ASP.NET: This is the engine that hosts web applications and web services, with almost any feature from the .NET class library. ASP.NET also includes a set of web-specific services.

Visual Studio: This optional development tool contains a rich set of productivity and debugging features. The Visual Studio setup CDs (or DVD) include the complete .NET Framework, so you won't need to download it separately.

❑ASP.NET File Types

ASP.NET have many types of files. They are,

.aspx : These are ASP.NET web pages (the .NET equivalent of the .asp file in an ASP application). They contain the user interface and, optionally, the underlying application code. Users request or navigate directly to one of these pages to start your web application.

.ascx : These are ASP.NET user controls. User controls are similar to web pages, except that they can't be accessed directly. Instead, they must be hosted inside an ASP.NET web page. User controls allow you to develop a small piece of user interface and reuse it in as many web forms as you want without repetitive code.

.asmx : These are ASP.NET web services. Web services work differently than web pages, but they still share the same application resources, configuration settings, and memory.

web.config : This is the XML-based configuration file for your ASP.NET application. It includes settings for customizing security, state management, memory management, and much more.

global.asax : This is the global application file. You can use this file to define global variables (variables that can be accessed from any web page in the web application) and react to global events (such as when a web application first starts).

.cs / .vb : These are code-behind files that contain C# code or VB code. They allow you to separate the application from the user interface of a web page.

The Page Class : Every web page is a custom class that inherits from System.Web.UI.Page. By inheriting from this class, your web page class acquires a number of properties that your code can use. These include properties for enabling caching, validation, and tracing.

Property	Description
Application and Session	These collections hold state information on the server.
Cache	This collection allows you to store objects for reuse in other pages or for other clients.
Controls	Provides a collection of all the controls contained on the web page. You

	can also use the methods of this collection to add new controls dynamically.
EnableViewState	When set to false, this overrides the EnableViewState property of the contained controls, thereby ensuring that no controls will maintain state information.
IsPostBack	This Boolean property indicates whether this is the first time the page is being run (false) or whether the page is being resubmitted in response to a control event, typically with stored view state information (true). This property is often used in the Page.Load event handler, thereby ensuring that basic setup is performed only once for controls that maintain view state
Request	Refers to an HttpRequest object that contains information about the current web request, including client certificates, cookies, and values submitted through HTML form elements. It supports the same features as the built-in ASP Request object.
Response	Refers to an HttpResponse object that allows you to set the web response or redirect the user to another web page. It supports the same features as the built-in ASP Response object, although it's used much less in .NET development.
Server	Refers to an HttpServerUtility object that allows you to perform some miscellaneous tasks, such as URL and HTML encoding. It supports the same features as the built-in ASP Server object.
User	If the user has been authenticated, this property will be initialized with user information

The Control Class : The Page.Controls collection includes all the controls on the current web form. You can loop through this collection and access each control. You can also use the Controls collection to add a dynamic control.

The HttpRequest Class : The HttpRequest class encapsulates all the information related to a client request for a web page. Most of this information corresponds to low-level details such as posted-back form values, server variables, the response encoding, and so on. If you're using ASP.NET to its fullest, you'll almost never dive down to that level. Other properties are generally useful for retrieving information, particularly about the capabilities of the client browser.

Property	Description
ApplicationPath and PhysicalPath	These collections hold state information on the server.
Browser	This collection allows you to store objects for reuse in other pages or for other clients.
ClientCerticate	Provides a collection of all the controls contained on the web page. You can also use the methods of this collection to add new controls dynamically.
Cookies	When set to false, this overrides the EnableViewState property of the contained controls, thereby ensuring that no controls will maintain state information.
Headers and Server Variables	This Boolean property indicates whether this is the first time the page is being run (false) or whether the page is being resubmitted in response to a control event, typically with stored view state information (true). This property is often used in the Page.Load event handler, thereby ensuring that basic setup is performed only once for controls that maintain view state

IsAuthenticated and IsSecureConnection	Returns true if the user has been successfully authenticated and if the user is connected over SSL (also known as the Secure Sockets Layer).
QueryString	Provides the parameters that were passed along with the query string.
Url and UrlReferrer	Provides a Uri object that represents the current address for the page and the page where the user is coming from (the previous page that linked to this page).
UserAgent	A string representing the browser type. Internet Explorer provides the value MSIE for this property.
UserHostAddress and UserHostName	Gets the IP address and the DNS name of the remote client. You could also access this information through the ServerVariables collection.
UserLanguages	Provides a sorted string array that lists the client's language preferences. This can be useful if you need to create multilingual pages.

The HttpResponse Class : The HttpResponse class allows you to send information directly to the client. In traditional ASP development, the Response object was used heavily to create dynamic pages. The HttpResponse does still provide some important functionality, namely, caching support, cookie features, and the Redirect method.

Property	Description
BufferOutput	When set to true (the default), the page isn't sent to the client until it's completely rendered and ready, as opposed to being sent piecemeal.
Cache	References an HttpCachePolicy object that allows you to configure how this page will be cached
Cookies	The collection of cookies sent with the response.
Write(), BinaryWrite(), and WriteFile()	These methods allow you to write text or binary content directly to the response stream. You can even write the contents of a file. These methods are de-emphasized in ASP.NET and shouldn't be used in conjunction with server controls.
Redirect()	This method transfers the user to another page in your application or a different website.

The ServerUtility Class : The ServerUtility class provides some miscellaneous helper methods.

Property	Description
CreateObject	When set to true (the default), the page isn't sent to the client until it's completely rendered and ready, as opposed to being sent piecemeal.
HtmlEncode and HtmlDecode	Changes an ordinary string into a string with legal HTML characters and back again.
UrlEncode and UrlDecode	Changes an ordinary string into a string with legal URL characters and back again.
MapPath	Returns the physical file path that corresponds to a specified virtual file path on the web server.
Transfer	Transfers execution to another web page in the current application. This is similar to the Response.Redirect() method but is slightly faster. It cannot be used to transfer the user to a site on another web server or to a non-ASP.NET page (such as an HTML page or a ASP page).

❑ Introducing ASP.NET Controls

The ASP.NET Framework (version 2.0) contains over 70 controls. These controls can be divided into eight groups:

- **Standard Controls :** The standard controls enable you to render standard form elements such as buttons, input fields, and labels. We examine these controls in detail in the following chapter, "Using the Standard Controls."
- **Validation Controls :** The validation controls enable you to validate form data before you submit the data to the server. For example, you can use a RequiredFieldValidator control to check whether a user entered a value for a required input field.
- **Rich Controls :** The rich controls enable you to render things such as calendars, file upload buttons, rotating banner advertisements, and multi-step wizards.
- **Data Controls :** The data controls enable you to work with data such as database data. For example, you can use these controls to submit new records to a database table or display a list of database records. Navigation Controls The navigation controls enable you to display standard navigation elements such as menus, tree views, and bread crumb trails.
- **Login Controls :** The login controls enable you to display login, change password, and registration forms. These controls are discussed in Chapter 20, "Using the Login Controls."
- **Web Part Controls :** The Web Part controls enable you to build personalizable portal applications.
- **HTML Controls :** The HTML controls enable you to convert any HTML tag into a server-side control.

With the exception of the HTML controls, you declare and use all the ASP.NET controls in a page in exactly the same way. For example, if you want to display a text input field in a page, then you can declare a TextBox control like this:

```
<asp:TextBox id="TextBox1" runat="Server" />
```

This control declaration looks like the declaration for an HTML tag. Remember, however, unlike an HTML tag, a control is a .NET class that executes on the server and not in the web browser.

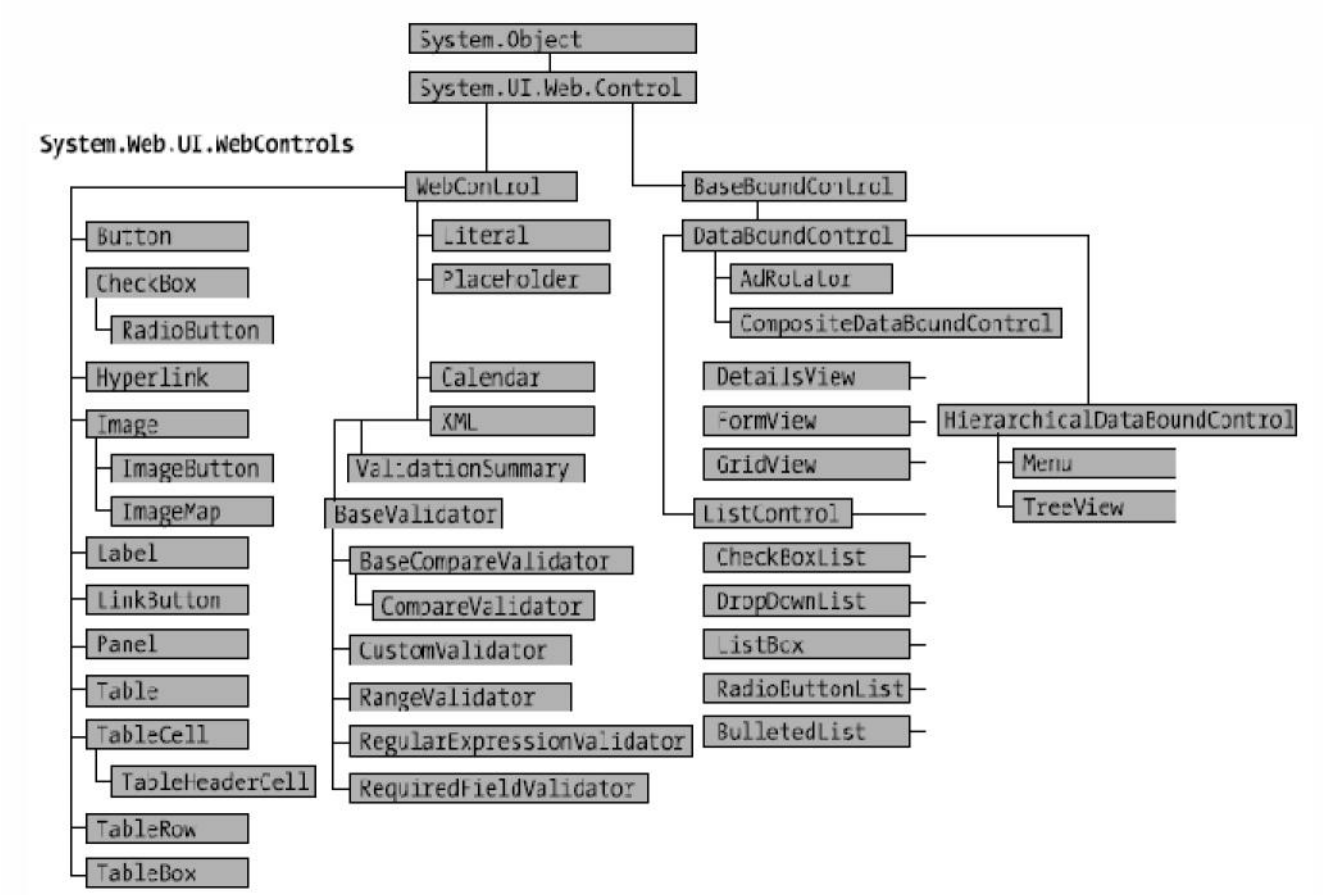
When the TextBox control is rendered to the browser, it renders the following content:

```
<input name="TextBox1" type="text" id="TextBox1" />
```

The first part of the control declaration, the asp: prefix, indicates the namespace for the control. All the standard ASP.NET controls are contained in the System.Web.UI.WebControls namespace. The prefix asp: represents this namespace.

Next, the declaration contains the name of the control being declared. In this case, a TextBox control is being declared.

This declaration also includes an ID attribute. You use the ID to refer to the control in the page within your code. Every control must have a unique ID.



❑ Adding Application Logic to an ASP.NET Page

You can add programming logic in your ASP.NET page. You can write coding as per standard event driven programming. The majority of the ASP.NET controls support one or more events. For example, the ASP.NET Button control supports the Click event. The Click event is raised on the server after you click the button rendered by the Button control in the browser.

Consider following code of Button control.

```
<asp:Button id="btnSubmit" Text="Click Here"
            OnClick="btnSubmit_Click" Runat="server" />
```

Notice that the Button control includes an **OnClick** attribute. This attribute points to a subroutine named **btnSubmit_Click()**. The **btnSubmit_Click()** subroutine is the handler for the Button Click event. This subroutine executes whenever you click the button.

In Design view, you can double-click a control to add a handler for the control's default event. Double-clicking a control switches you to Source view and adds the event handler.

Finally, from Design view, after selecting a control on the designer surface you can add an event handler from the Properties window by clicking the Events button (the lightning bolt) and double-clicking next to the name of any of the events

It is important to understand that all ASP.NET control events happen on the server. For example, the Click event is not raised when you actually click a button. The Click event is not raised until the page containing the Button control is posted back to the server.

The ASP.NET Framework is a server-side web application framework. The .NET Framework code that you write executes on the server and not within the web browser. From the perspective of ASP.NET, nothing happens until the page is posted back to the server and can execute within the context of the .NET Framework.

```
protected void btn_Submit_Click(object sender, EventArg e)
{
        -
        -
}
```

Notice that two parameters are passed to the btnSubmit_Click() handler. All event handlers for ASP.NET controls have the same general signature.

The first parameter, the object parameter named sender, represents the control that raised the event. In other words, it represents the Button control which you clicked.

You can wire multiple controls in a page to the same event handler and use this first parameter to determine the particular control that raised the event. The second parameter passed to the Click event handler, the EventArgs parameter named e, represents any additional event information associated with the event.

The WebControl Base Class

All web controls begin by inheriting from the WebControl base class. This class defines the essential functionality for tasks such as data binding and includes some basic properties that you can use with any control.

Some of the new properties of WebControl class are as follows :

Property	Description
AccessKey	Specifies the keyboard shortcut as one letter. For example, if you set this to Y, the Alt+Y keyboard combination will automatically change focus to this web control. This feature is supported only on Internet Explorer 4.0 and higher.
Controls	Provides a collection of all the controls contained inside the current control. Each object is provided as a generic System.Web.UI.Control object, so you will need to cast the reference to access control-specific properties.
EnableViewState	Set this to false to disable the automatic state management for this control. In this case, the control will be reset to the properties and formatting specified in the control tag every time the page is posted back. If this is set to true (the default), the control uses the hidden input field to store information about its properties, ensuring that any changes you make in code are remembered.
Page	Provides a reference to the web page that contains this control as a System.Web.UI.Page object.
Parent	Provides a reference to the control that contains this control. If the control is placed directly on the page (rather than inside another control), it will return a reference to the page object.

TabIndex	A number that allows you to control the tab order. The control with a TabIndex of 0 has the focus when the page first loads. Pressing Tab moves the user to the control with the next lowest TabIndex, provided it is enabled. This property is supported only in Internet Explorer 4.0 and higher.
ToolTip	Displays a text message when the user hovers the mouse above the control. Many older browsers don't support this property.
Visible	When set to false, the control will be hidden and will not be rendered to the final HTML page that is sent to the client.

Building Forms with Web Server Controls

Web Forms are the basics of building ASP.NET based web sites. Web site is collection of number of web Forms / pages where each Web Form contains number of Web Server controls. Which are described as follows.

❑ Label Control

Whenever you need to modify the text displayed in a page dynamically, you can use the Label control. Any string that you assign to the Label control's Text property is displayed by the Label when the control is rendered. You can assign simple text to the Text property or you can assign HTML content.

As an alternative to assigning text to the Text property, you can place the text between the Label control's opening and closing tags. Any text that you place before the opening and closing tags gets assigned to the Text property.

The Label control supports several properties you can use to format the text displayed by the Label (this is not a complete list):

- **BackColor :** Enables you to change the background color of the label.
- **BorderColor :** Enables you to set the color of a border rendered around the label.
- **BorderStyle :** Enables you to display a border around the label. Possible values are NotSet, None, Dotted, Dashed, Solid, Double, Groove, Ridge, Inset, and Outset.
- **BorderWidth :** Enables you to set the size of a border rendered around the label.
- **CssClass :** Enables you to associate a Cascading Style Sheet class with the label.
- **Font** : Enables you to set the label's font properties.
- **ForeColor :** Enables you to set the color of the content rendered by the label.
- **Style :** Enables you to assign style attributes to the label.
- **ToolTip :** Enables you to set a label's title attribute. (In Microsoft Internet Explorer, the title attribute is displayed as a floating tooltip.)

❑ Literal Control

The Literal control is similar to the Label control. You can use the Literal control to display text or HTML content in a browser. The Literal control does support one property that is not supported by the Label control: the Mode property. The Mode property enables you to encode HTML content. The Mode property accepts any of the following three values:

- **PassThrough :** Displays the contents of the control without encoding.
- **Encode :** Displays the contents of the control after HTML encoding the content.
- **Transform :** Displays the contents of the control after stripping markup that is not supported by the requesting device.

❑ TextBox Control

The TextBox control can be used to display three different types of input fields depending on the value of its TextMode property. The TextMode property accepts the following three values:

- **SingleLine :** Displays a single-line input field.
- **MultiLine :** Displays a multi-line input field.
- **Password :** Displays a single-line input field in which the text is hidden.

You can use the following properties to control the rendering characteristics of the TextBox control (this is not a complete list):

- **AccessKey :** Enables you to specify a key that navigates to the TextBox control.
- **AutoCompleteType :** Enables you to associate an AutoComplete class with the TextBox control.
- **AutoPostBack :** Enables you to post the form containing the TextBox back to the server automatically when the contents of the TextBox is changed.
- **Columns :** Enables you to specify the number of columns to display.
- **Enabled :** Enables you to disable the text box.
- **MaxLength** Enables you to specify the maximum length of data that a user can enter in a text box (does not work when TextMode is set to Multiline).
- **ReadOnly :** Enables you to prevent users from changing the text in a text box.
- **Rows :** Enables you to specify the number of rows to display.
- **TabIndex :** Enables you to specify the tab order of the text box.
- **Wrap :** Enables you to specify whether text word-wraps when the TextMode is set to Multiline.

The TextBox control also supports the following method:

- **Focus :** Enables you to set the initial form focus to the text box.

The TextBox control supports the following event:

- **TextChanged :** Raised on the server when the contents of the text box are changed.

Notice that the TextBox control also includes a property that enables you to associate the TextBox with a particular AutoComplete class. When AutoComplete is enabled, the user does not need to re-enter common information such as a first name, last name, or phone number in a form field. If the user has not disabled AutoComplete on his browser, then his browser prompts him to enter the same value that he entered previously for the form field (even if the user entered the value for a form field at a different website).

❑ CheckBox Control

CheckBox control is used to accept the choice from user. It is used to display multiple choices from which user can select none of them or many or all of them. For example, if you want to accept Hobbies of user, you can use CheckBox control.

The CheckBox control supports the following properties (this is not a complete list):

- **AccessKey :** Enables you to specify a key that navigates to the TextBox control.
- **AutoPostBack :** Enables you to post the form containing the CheckBox back to the server automatically when the CheckBox is checked or unchecked.
- **Checked :** Enables you to get or set whether the CheckBox is checked.
- **Enabled :** Enables you to disable the TextBox.
- **TabIndex :** Enables you to specify the tab order of the check box.
- **Text :** Enables you to provide a label for the check box.
- **TextAlign :** Enables you to align the label for the check box. Possible values are Left and Right.

The CheckBox control also supports the following method:

- **Focus :** Enables you to set the initial form focus to the check box.

The CheckBox control supports the following event:

- **CheckedChanged :** Raised on the server when the check box is checked or unchecked.

❑ RadioButton Control

You always use the RadioButton control in a group. Only one radio button in a group of RadioButton controls can be checked at a time.

The RadioButton control supports the following properties (this is not a complete list):

- **AccessKey :** Enables you to specify a key that navigates to the RadioButton control.
- **AutoPostBack :** Enables you to post the form containing the RadioButton back to the server automatically when the radio button is checked or unchecked.
- **Checked :** Enables you to get or set whether the RadioButton control is checked.
- **Enabled :** Enables you to disable the RadioButton.
- **GroupName :** Enables you to group RadioButton controls.
- **TabIndex :** Enables you to specify the tab order of the RadioButton control.
- **Text :** Enables you to label the RadioButton control.
- **TextAlign :** Enables you to align the RadioButton label. Possible values are Left and Right.

The RadioButton control supports the following method:

- **Focus :** Enables you to set the initial form focus to the RadionButton control.

Finally, the RadioButton control supports the following event:

- **CheckedChanged :** Raised on the server when the RadioButton is checked or unchecked.

❑ Button Control

Button control is used to submit the data to the server. Button control works like a Push Button when you click the data is submitted to the server. The Button control supports the following properties (this is not a complete list):

- **AccessKey :** Enables you to specify a key that navigates to the Button control.
- **CommandArgument :** Enables you to specify a command argument that is passed to the Command event.
- **CommandName :** Enables you to specify a command name that is passed to the Command event.
- **Enabled :** Enables you to disable the Button control.
- **OnClientClick :** Enables you to specify a client-side script that executes when the button is clicked.
- **PostBackUrl :** Enables you to post a form to a particular page.
- **TabIndex :** Enables you to specify the tab order of the Button control.
- **Text :** Enables you to label the Button control.
- **UseSubmitBehavior :** Enables you to use JavaScript to post a form.

The Button control also supports the following method:

- **Focus :** Enables you to set the initial form focus to the Button control.

The Button control also supports the following two events:

- **Click :** Raised when the Button control is clicked.
- **Command :** Raised when the Button control is clicked. The CommandName and CommandArgument are passed to this event.

❑ LinkButton Control

The LinkButton control, like the Button control, enables you to post a form to the server. Unlike a Button control, however, the LinkButton control renders a link instead of a push button.

Behind the scenes, the LinkButton control uses JavaScript to post the form back to the server. The hyperlink rendered by the LinkButton control looks like this:

```
<a id="lnkSubmit" href="javascript:__doPostBack('lnkSubmit','')">Submit</a>
```

Clicking the LinkButton invokes the JavaScript __doPostBack() method, which posts the form to the server. When the form is posted, the values of all the other form fields in the page are also posted to the server.

The LinkButton control supports the following properties (this is not a complete list):

- **AccessKey :** Enables you to specify a key that navigates to the Button control.
- **CommandArgument :** Enables you to specify a command argument that is passed to the Command event.
- **CommandName :** Enables you to specify a command name that is passed to the Command event.
- **Enabled :** Enables you to disable the LinkButton control.
- **OnClientClick :** Enables you to specify a client-side script that executes when the LinkButton is clicked.
- **PostBackUrl :** Enables you to post a form to a particular page.
- **TabIndex :** Enables you to specify the tab order of the LinkButton control.
- **Text :** Enables you to label the LinkButton control.

The LinkButton control also supports the following method:

- **Focus :** Enables you to set the initial form focus to the LinkButton control.

The LinkButton control also supports the following two events:

- **Click :** Raised when the LinkButton control is clicked.
- **Command :** Raised when the LinkButton control is clicked. The CommandName and CommandArgument are passed to this event.

❑ ImageButton Control

The ImageButton control, like the Button and LinkButton controls, enables you to post a form to the server. However, the ImageButton control always displays an image.

The ImageButton includes both an ImageUrl and AlternateText property. The ImageUrl contains the path to the image that the ImageButton displays. The AlternateText property is used to provide alternate text for the image used by screen readers and text-only browsers.

Notice that the event handler for an Image control's Click event is different than that for the other button controls. The second parameter passed to the event handler is an instance of the ImageClickEventArgs class. This class has the following properties:

- **X :** The x coordinate relative to the image the user clicked.
- **Y :** The y coordinate relative to the image the user clicked.

You can use the ImageButton control to create a simple image map. The ImageButton can be used to create a server-side image map. Server-side image maps are not accessible to persons with disabilities. A better method for creating an ImageMap is to use the ImageMap control, which enables you to create a client-side image map.

The ImageButton control supports the following properties (this is not a complete list):

- **AccessKey :** Enables you to specify a key that navigates to the ImageButton control.
- **AlternateText :** Enables you to provide alternate text for the image (required for accessibility).
- **DescriptionUrl :** Enables you to provide a link to a page that contains a detailed description of the image (required to make a complex image accessible).
- **CommandArgument :** Enables you to specify a command argument that is passed to the Command event.
- **CommandName :** Enables you to specify a command name that is passed to the Command event.
- **Enabled :** Enables you to disable the ImageButton control.
- **GenerateEmptyAlternateText :** Enables you to set the AlternateText property to an empty string.
- **ImageAlign :** Enables you to align the image relative to other HTML elements in the page. Possible values are AbsBottom, AbsMiddle, Baseline, Bottom, Left, Middle, NotSet, Right, TextTop, and Top.
- **ImageUrl :** Enables you to specify the URL to the image.
- **OnClientClick :** Enables you to specify a client-side script that executes when the ImageButton is clicked.
- **PostBackUrl :** Enables you to post a form to a particular page.
- **TabIndex :** Enables you to specify the tab order of the ImageButton control.

The ImageButton control also supports the following method:

- **Focus :** Enables you to set the initial form focus to the ImageButton control.

The ImageButton control also supports the following two events:

- **Click :** Raised when the ImageButton control is clicked.
- **Command :** Raised when the ImageButton control is clicked. The CommandName and CommandArgument are passed to this event.

❑ Image Control

Image control is used to display an image. It has got some following properties.

- AlternateText Enables you to provide alternate text for the image (required for accessibility).
- DescriptionUrl Enables you to provide a link to a page that contains a detailed description of the image (required to make a complex image accessible).
- GenerateEmptyAlternateText Enables you to set the AlternateText property to an empty string.
- ImageAlign Enables you to align the image relative to other HTML elements in the page. Possible values are AbsBottom, AbsMiddle, Baseline, Bottom, Left, Middle, NotSet, Right, TextTop, and Top.
- ImageUrl Enables you to specify the URL to the image.

The Image control supports three methods for supplying alternate text. If an image represents page content, then you should supply a value for the AlternateText property. For example, if you have an image for your company's logo, then you should assign the text "My Company Logo" to the AlternateText property.

If an Image control represents something really complex such as a bar chart, pie graph, or company organizational chart hen you should supply a value for the DescriptionUrl property. The DescriptionUrl property links to a page that contains a long textual description of the image.

Finally, if the image is used purely for decoration (it expresses no content), then you should set the GenerateEmptyAlternateText property to the value True. When this property has the value True, then an alt="" attribute is included in the rendered <img> tag. Screen readers know to ignore images with empty alt attributes.

❑ ImageMap Control

The ImageMap control enables you to create a client-side image map. An image map displays an image. When you click different areas of the image, things happen.

For example, you can use an image map as a fancy navigation bar. In that case, clicking different areas of the image map navigates to different pages in your website.

You also can use an image map as an input mechanism. For example, you can click different product images to add a particular product to a shopping cart.

An ImageMap control is composed out of instances of the HotSpot class. A HotSpot defines the clickable regions in an image map. The ASP.NET framework ships with three HotSpot classes:

- **CircleHotSpot :** Enables you to define a circular region in an image map.
- **PolygonHotSpot :** Enables you to define an irregularly shaped region in an image map.
- **RectangleHotSpot :** Enables you to define a rectangular region in an image map.

Each RectangleHotSpot includes Left, Top, Right, and Bottom properties that describe the area of the rectangle. Each RectangleHotSpot also includes a NavigateUrl property that contains the URL to which the region of the image map links.

The ImageMap control supports the following properties (this is not a complete list):

- **AccessKey** Enables you to specify a key that navigates to the ImageMap control.
- **AlternateText** Enables you to provide alternate text for the image (required for accessibility).
- **DescriptionUrl** Enables you to provide a link to a page which contains a detailed description of the image (required to make a complex image accessible).
- **GenerateEmptyAlternateText** Enables you to set the AlternateText property to an empty string.
- **HotSpotMode** Enables you to specify the behavior of the image map when you click a region. Possible values are Inactive, Navigate, NotSet, and PostBack.
- **HotSpots** Enables you to retrieve the collection of HotSpots contained in the ImageMap control.
- **ImageAlign** Enables you to align the image map with other HTML elements in the page. Possible values are AbsBottom, AbsMiddle, Baseline, Bottom, Left, Middle, NotSet, Right, TextTop, and Top.
- **ImageUrl** Enables you to specify the URL to the image.
- **TabIndex** Enables you to specify the tab order of the ImageMap control.
- **Target** Enables you to open a page in a new window.

The ImageMap control also supports the following method:

- **Focus** Enables you to set the initial form focus to the ImageMap control.

Finally, the ImageMap control supports the following event:

- **Click** Raised when you click a region of the ImageMap and the HotSpotMode property is set to the value PostBack.

❑ Panel Control

The Panel control enables you to work with a group of ASP.NET controls. You can use a Panel control to hide or show a group of ASP.NET controls.

The Panel control supports the following properties (this is not a complete list):

- **DefaultButton :** Enables you to specify the default button in a Panel. The default button is invoked when you press the Enter button.
- **Direction :** Enables you to get or set the direction in which controls that display text are rendered. Possible values are NotSet, LeftToRight, and RightToLeft.
- **GroupingText :** Enables you to render the Panel control as a fieldset with a particular legend.
- **HorizontalAlign :** Enables you to specify the horizontal alignment of the contents of the Panel. Possible values are Center, Justify, Left, NotSet, and Right.
- **ScrollBars :** Enables you to display scrollbars around the panel's contents. Possible values are Auto, Both, Horizontal, None, and Vertical.

❑ HyperLink Control

The HyperLink control enables you to create a link to a page. Unlike the LinkButton control, the HyperLink control does not submit a form to a server.

The HyperLink control supports the following properties (this is not a complete list):

- **Enabled :** Enables you to disable the hyperlink.
- **ImageUrl :** Enables you to specify an image for the hyperlink.
- **NavigateUrl :** Enables you to specify the URL represented by the hyperlink.
- **Target :** Enables you to open a new window.
- **Text :** Enables you to label the hyperlink.

Notice that you can specify an image for the HyperLink control by setting the ImageUrl property. If you set both the Text and ImageUrl properties, then the ImageUrl property takes precedence.

Performing Form Validations with Validation Controls

Six validation controls are included in the ASP.NET 2.0 Framework:

- **RequiredFieldValidator :** Enables you to require a user to enter a value in a form field.
- **RangeValidator :** Enables you to check whether a value falls between a certain minimum and maximum value.
- **CompareValidator :** Enables you to compare a value against another value or perform a data type check.
- **RegularExpressionValidator :** Enables you to compare a value against a regular expression.
- **CustomValidator :** Enables you to perform custom validation.
- **ValidationSummary :** Enables you to display a summary of all validation errors in a page.

You can associate the validation controls with any of the form controls included in the ASP.NET Framework. For example, if you want to require a user to enter a value into a TextBox control, then you can associate a RequiredFieldValidator control with the TextBox control. Technically, you can use the validation controls with any control that is decorated with the ValidationProperty attribute.

Each RequiredFieldValidator is associated with a particular control through its ControlToValidate property. This property accepts the name of the control to validate on the page.

CompareValidator controls are associated with the txtProductPrice and txtProductQuantity TextBox controls. The first CompareValidator is used to check whether the txtProductPrice text field contains a currency value, and the second CompareValidator is used to check whether the txtProductQuantity text field contains an integer value.

Notice that there is nothing wrong with associating more than one validation control with a form field. If you need to make a form field required and check the data type entered into the form field, then you need to associate both a RequiredFieldValidator and CompareValidator control with the form field.

By default, the validation controls perform validation on both the client (the browser) and the server. The validation controls use client-side JavaScript. This is great from a user experience perspective because you get immediate feedback whenever you enter an invalid value into a form field.

You can use the validation controls with browsers that do not support JavaScript (or do not have JavaScript enabled). If a browser does not support JavaScript, the form must be posted back to the server before a validation error message is displayed.

Even when validation happens on the client, validation is still performed on the server. This is done for security reasons. If someone creates a fake form and submits the form data to your web server, the person still won’t be able to submit invalid data.

Highlighting Validation Errors

When a validation control displays a validation error, the control displays the value of its Text property. Normally, you assign a simple text string, such as "(Required)" to the Text property. However, the Text property accepts any HTML string.

Another way that you can emphasize errors is to take advantage of the SetFocusOnError property that is supported by all the validation controls. When this property has the value TRue, the form focus is automatically shifted to the control associated with the validation control when there is a validation error.

Using Validation Groups

If you added more than one form to a page, and both forms contained validation controls, then the validation controls in both forms were evaluated regardless of which form you submitted.

For example, imagine that you wanted to create a page that contained both a login and registration form. The login form appeared in the left column and the registration form appeared in the right column. If both forms included validation controls, then submitting the login form caused any validation controls contained in the registration form to be evaluated.

Disabling Validation

All the button controls the Button, LinkButton, and ImageButton control include a CausesValidation property. If you assign the value False to this property, then clicking the button bypasses any validation in the page.

❑ RequiredFieldValidator Control

The RequiredFieldValidator control enables you to require a user to enter a value into a form field before submitting the form. You must set two important properties when using the RequiredFieldValdiator control:

- **ControlToValidate :** The ID of the form field being validated.
- **Text :** The error message displayed when validation fails.

By default, the RequiredFieldValidator checks for a nonempty string (spaces don't count). If you enter anything into the form field associated with the RequiredFieldValidator, then the RequiredFieldValidator does not display its validation error message. You can use the RequiredFieldValidator control's InitialValue property to specify a default value other than an empty string.

❑ RangeValidator Control

The RangeValidator control enables you to check whether the value of a form field falls between a certain minimum and maximum value. You must set five properties when using this control:

- **ControlToValidate :** The ID of the form field being validated.
- **Text :** The error message displayed when validation fails.
- **MinimumValue :** The minimum value of the validation range.
- **MaximumValue :** The maximum value of the validation range.
- **Type :** The type of comparison to perform. Possible values are String, Integer, Double, Date, and Currency.

If you don't enter any value into the age field and submit the form, no error message is displayed. If you want to require a user to enter a value, you must associate a RequiredFieldValidator with the form field.

Don’t forget to set the Type property when using the RangeValidator control. By default, the Type property has the value String, and the RangeValidator performs a string comparison to determine whether a values falls between the minimum and maximum value.

❑ CompareValidator Control

The CompareValidator control enables you to perform three different types of validation tasks. You can use the CompareValidator to perform a data type check. In other words, you can use the control to determine whether a user has entered the proper type of value into a form field, such as a date in a birth date field.

You also can use the CompareValidator to compare the value entered into a form field against a fixed value. For example, if you are building an auction website, you can use the CompareValidator to check whether a new minimum bid is greater than the previous minimum bid.

Finally, you can use the CompareValidator to compare the value of one form field against another. For example, you use the CompareValidator to check whether the value entered into the meeting start date is less than the value entered into the meeting end date.

The CompareValidator has six important properties:

- **ControlToValidate :** The ID of the form field being validated.
- **Text :** The error message displayed when validation fails.
- **Type :** The type of value being compared. Possible values are String, Integer, Double, Date, and Currency.
- **Operator :** The type of comparison to perform. Possible values are DataTypeCheck, Equal, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and NotEqual.
- **ValueToCompare :** The fixed value against which to compare.
- **ControlToCompare :** The ID of a control against which to compare.

Just like the RangeValidator, the CompareValidator does not display an error if you don’t enter a value into the form field being validated. If you want to require that a user enter a value, then you must associate a RequiredFieldValidator control with the field.

❑ RegularExpressionValidator Control

The RegularExpressionValidator control enables you to compare the value of a form field against a regular expression. You can use a regular expression to represent string patterns such as email addresses, Social Security numbers, phone numbers, dates, currency amounts, and product codes.

Just like the other validation controls, the RegularExpressionValidator doesn’t validate a form field unless the form field contains a value. To make a form field required, you must associate a RequiredFieldValidator control with the form field.

There are different expressions that can be given as Expression

Character	Meaning for Character in Expression
\w	Any word character (Letter, Number or Underscore)
\d	Any digit
\D	Any character that is not Digit

\s	Any white space character like Tab or Space
\S	Any non-white space character

Some expressions with examples.

Expression	**Meaning for Expression**
\w {10}	10 word character (Letter, Number or Underscore)
\w {5,10}	Word character between 5 to 10
\d {5,10)	Any digits ranging from 5 to 10
[A-D]	Any 1 alphabet between A to D
[A-Z]{5,15}	Any alphabet between A to D but the no. of alphabets can be between 5 to 15.
\d{2}-\d{7,10}	Ex. 99-999999 (2 digits then hyphen and after that digits between 7 to 10

❑ CustomValidator Control

If none of the other validation controls perform the type of validation that you need, you can always use the CustomValidator control. You can associate a custom validation function with the CustomValidator control.

The CustomValidator control has three important properties:

- **ControlToValidate :** The ID of the form field being validated.
- **Text :** The error message displayed when validation fails.
- **ClientValidationFunction :** The name of a client-side function used to perform client-side validation.

The CustomValidator also supports one event:

- **ServerValidate :** This event is raised when the CustomValidator performs validation.

You associate your custom validation function with the CustomValidator control by handling the ServerValidate event.

The second parameter passed to the ServerValidate event handler is an instance of the ServerValidateEventArgs class. This class has two properties:

- **Value :** Represents the value of the form field being validated.
- **IsValid :** Represents whether validation fails or succeeds.
- **ValidateEmptyText :** Represents whether validation is performed when the form field being validated does not contain a value.

❑ ValidationSummary Control

The ValidationSummary control enables you to display a list of all the validation errors in a page in one location. This control is particularly useful when working with large forms. If a user enters the

wrong value for a form field located toward the end of the page, then the user might never see the error message. If you use the ValidationSummary control, however, you can always display a list of errors at the top of the form.

You might have noticed that each of the validation controls includes an ErrorMessage property. We have not been using the ErrorMessage property to represent the validation error message. Instead, we have used the Text property.

The distinction between the ErrorMessage and Text property is that any message that you assign to the ErrorMessage property appears in the ValidationSummary control, and any message that you assign to the Text property appears in the body of the page. Normally, you want to keep the error message for the Text property short (for example, "Required!"). The message assigned to the ErrorMessage property, on the other hand, should identify the form field that has the error (for example, "First name is required!").

The ValidationSummary control supports the following properties:

- **DisplayMode :** Enables you to specify how the error messages are formatted. Possible values are BulletList, List, and SingleParagraph.
- **HeaderText :** Enables you to display header text above the validation summary.
- **ShowMessageBox :** Enables you to display a popup alert box.
- **ShowSummary :** Enables you to hide the validation summary in the page.

If you set the ShowMessageBox property to the value true and the ShowSummary property to the value False, then you can display the validation summary only within a popup alert box.

Advanced Control Programming

You can do advanced control programming with many controls / facilities available under ASP.NET. For example, you can work with View State, Hidden Field. Apart from this ASP.NET have no. of rich controls which are used to give some additional facilities.

❑ Working with View State

The HTTP protocol, the fundamental protocol of the World Wide Web, is a stateless protocol. Each time you request a web page from a website, from the website's perspective, you are a completely new person.

The ASP.NET Framework, however, manages to transcend this limitation of the HTTP protocol. For example, if you assign a value to a Label control's Text property, the Label control retains this value across multiple page requests.

The standard ASP.NET controls retain the values of their properties across postbacks. For example, if you change the text displayed by a Label control, the Label control will continue to display the new text even if you repeatedly post the page containing the Label control back to the server.

The ASP.NET Framework takes advantage of a hidden form field named __VIEWSTATE to preserve the state of control properties across postbacks. If you want your controls to preserve the values of their properties, then you need to add the values of your control properties to this hidden form field.

You can use the ViewState property of the Control or Page class to add values to View State. The ViewState property exposes a dictionary of key and value pairs. For example, the following statement adds the string Hello World! to View State:

```
ViewState("message") = "Hello World!"
```

The ASP.NET Framework uses a trick called View State. If you open the page in your browser and select View Source, you'll notice that the page includes a hidden form field named __VIEWSTATE that looks like this:

```
<input type="hidden" name="__VIEWSTATE" id="__
      VIEWSTATE" value="/wEPDwUKLTc2ODE1OTYxNw9kFgICBA9kFgIC
      Aw8PFgIeBFRleHQFATFkZGT3tMnThg9KZpGak55p367vfInj1w==" />
```

This hidden form field contains the value of the Label control's Text property (and the values of any other control properties that are stored in View State). When the page is posted back to the server, the ASP.NET Framework rips apart this string and re-creates the values of all the properties stored in View State. In this way, the ASP.NET Framework preserves the state of control properties across postbacks to the web server.

By default, View State is enabled for every control in the ASP.NET Framework. If you change the background color of a Calendar control, the new background color is remembered across postbacks. If you change the selected item in a DropDownList, the selected item is remembered across postbacks. The values of these properties are automatically stored in View State.

View State is a good thing, but sometimes it can be too much of a good thing. The __VIEWSTATE hidden form field can become very large. Stuffing too much data into View State can slow down the rendering of a page because the contents of the hidden field must be pushed back and forth between the web server and web browser.

You can determine how much View State each control contained in a page is consuming by enabling tracing for a page. The page should include a trace="true" attribute in its <%@ Page %> directive, which enables tracing.

Every ASP.NET control includes a property named EnableViewState. If you set this property to the value False, then View State is disabled for the control. In that case, the values of the control properties are not remembered across postbacks to the server.

❑ Displaying and Hiding Content

You can Hide and Display contents in no. of ways.

- All controls have visible property, which can be set to “true” or “false” to display or hide the contents of control.
- You can use HiddenField where you can save some value. HiddenField can retain value but its not visible on web page.
- You can put some controls inside panel and set it to “true” or “false” as per your requirement to display or hide controls.

HiddenField

HiddenField is specially used to hide the contents. HiddenField has only one important property called Value. Inside which we can assign some value. When the page is rendered, along with other controls HiddenField control is also rendered with them. But that is not visible. Although Hidden field is specially used to hide the value, it has visible property which can be set to true and false.

❑ Using Rich Controls

In your website you need no. of rich controls to give some more facilities to the client. Each of the rich control is used for different purpose. Some of mostly used Rich Controls are :

- **FileUpload Control**
- **Calendar Control**
- **AdRotator Control**
- **MultiView Control**
- **Wizard Control**
- **Treeview Control**

❑ FileUpload Control :

The FileUpload control enables users to upload files to your web application. After the file is uploaded, you can store the file anywhere you please. Normally, you store the file either on the file system or in a database. This section explores both options.

The FileUpload control supports the following properties (this is not a complete list):

- **Enabled** : Enables you to disable the FileUpload control.
- **FileBytes** : Enables you to get the uploaded file contents as a byte array.

- **FileContent** : Enables you to get the uploaded file contents as a stream.
- **FileName:** Enables you to get the name of the file uploaded.
- **HasFileReturns :** true when a file has been uploaded.
- **PostedFile :** Enables you to get the uploaded file wrapped in the HttpPostedFile object.

The FileUpload control also supports the following methods:

- **Focus :** Enables you to shift the form focus to the FileUpload control.
- **SaveAs :**Enables you to save the uploaded file to the file system.

The FileUpload control's PostedFile property enables you to retrieve the uploaded file wrapped in an HttpPostedFile object. This object exposes additional information about the uploaded file. The HttpPostedFile class has the following properties (this is not a complete list):

- **ContentLength :** Enables you to get the size of the uploaded file in bytes.
- **ContentType :** Enables you to get the MIME type of the uploaded file.
- **FileName :**Enables you to get the name of the uploaded file.
- **InputStream :** Enables you to retrieve the uploaded file as a stream.

The HttpPostedFile class also supports the following method:

- **SaveAs :** Enables you to save the uploaded file to the file system.

❑ Calendar Control :

The Calendar control enables you to display a calendar. You can use the calendar as a date picker or you can use the calendar to display a list of upcoming events.

The Calendar control supports the following properties (this is not a complete list):

- **DayNameFormat :** Enables you to specify the appearance of the days of the week. Possible values are FirstLetter, FirstTwoLetters, Full, Short, and Shortest.
- **NextMonthText :** Enables you to specify the text that appears for the next month link.
- **NextPrevFormat :** Enables you to specify the format of the next month and previous month link. Possible values are CustomText, FullMonth, and ShortMonth.
- **PrevMonthText :** Enables you to specify the text that appears for the previous month link.
- **SelectedDate :** Enables you to get or set the selected date.
- **SelectedDates :** Enables you to get or set a collection of selected dates.
- **SelectionMode :** Enables you to specify how dates are selected. Possible values are Day, DayWeek, DayWeekMonth, and None.
- **SelectMonthText :** Enables you to specify the text that appears for selecting a month.
- **SelectWeekText :** Enables you to specify the text that appears for selecting a week.
- **ShowDayHeader :** Enables you to hide or display the day names at the top of the Calendar control.
- **ShowNextPrevMonth :** Enables you to hide or display the links for the next and previous months.
- **ShowTitle :** Enables you to hide or display the title bar displayed at the top of the calendar.
- **TitleFormat :** Enables you to format the title bar. Possible values are Month and MonthYear.
- **TodaysDate :** Enables you to specify the current date. This property defaults to the current date on the server.
- **VisibleDate :** Enables you to specify the month displayed by the Calendar control. This property defaults to displaying the month that contains the date specified by TodaysDate.

The Calendar control also supports the following events:

- **DayRender :** Raised as each day is rendered.
- **SelectionChanged :** Raised when a new day, week, or month is selected.
- **VisibleMonthChanged :** Raised when the next or previous month link is clicked.

❑ AdRotator Control :

The AdRotator control enables you to randomly display different advertisements in a page. You can store the list of advertisements in either an XML file or in a database table.

The AdRotator control supports the following properties (this is not a complete list):

- **AdvertisementFileEnables** you to specify the path to an XML file that contains a list of banner advertisements.
- **AlternateTextFieldEnables** you to specify the name of the field for displaying alternate text for the banner advertisement image. The default value is AlternateText.
- **DataMemberEnables** you to bind to a particular data member in the data source.
- **DataSourceEnables** you to specify a data source programmatically for the list of banner advertisements.
- **DataSourceIDEnables** you to bind to a data source declaratively.
- **ImageUrlFieldEnables** you to specify the name of the field for the image URL for the banner advertisement. The default value for this field is ImageUrl.
- **KeywordFilterEnables** you to filter advertisements by a single keyword.
- **NavigateUrlFieldEnables** you to specify the name of the field for the advertisement link. The default value for this field is NavigateUrl.
- **TargetEnables** you to open a new window when a user clicks the banner advertisement.

The AdRotator control also supports the following event:

- **AdCreatedRaised** after the AdRotator control selects an advertisement but before the AdRotator control renders the advertisement.

Notice that the AdRotator control includes a KeywordFilter property. You can provide each banner advertisement with a keyword and then filter the advertisements displayed by the AdRotator control by using the value of the KeywordFilter property.

This property can be used in multiple ways. For example, if you are displaying more than one advertisement in the same page, then you can filter the advertisements by page regions. You can use the KeywordFilter to show the big banner advertisement on the top of the page and box ads on the side of the page.

You can also use the KeywordFilter property to filter advertisements by website section. For example, you might want to show different advertisements on your website’s home page than on your website’s search page.

While AdRotator control the important property is Advertisement File which is XML file. Following is the example of Advertisement File, which has no. of fields which are related to your AdRotator property file.

File Name : Ad.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements>
  <Ad>
        <ImageUrl>/Images/img1.gif</ImageUrl>
        <Width>500</Width>
        <Height>100</Height>
        <NavigateUrl>http://www.orkut.com</NavigateUrl>
        <AlternateText>Orkut Image</AlternateText>
        <Impressions>50</Impressions>
        <Keyword>Group</Keyword>
  </Ad>

  <Ad>
        <ImageUrl>/Images/img2.gif</ImageUrl>
        <Width>500</Width>
        <Height>100</Height>
        <NavigateUrl>http://www.gmail.com</NavigateUrl>
        <AlternateText>Gmail Image</AlternateText>
        <Impressions>50</Impressions>
        <Keyword>EMail</Keyword>
  </Ad>
</Advertisements>
```

Explanations of tags in Ad.xml File

- **<ImageUrl> :** Tag to specify virtual path of image
- **<Width> and <Height> :** Tags to specify width and height of image to be displayed.
- **<NavigateUrl> :** Tag to specify the link when you click in image.
- **<AlternateText> :** Tag to specify text if image is not showed due to some reason.
- **<Impressions> :** Tag to specify importance of image to show. Higher impression images are show more no. of times.
- **<Keyword> :** Tag to specify the keyword. If you specify some keyword, only related images are loaded. If not specified, all images are loaded.

❑ MultiView Control :

The MultiView control enables you to hide and display different areas of a page. This control is useful when you need to create a tabbed page. It is also useful when you need to divide a long form into multiple forms.

The MultiView control contains one or more View controls. You use the MultiView control to select a particular View control to render. (The selected View control is the Active View.) The contents of the remaining View controls are hidden. You can render only one View control at a time.

The MultiView control supports the following properties (this is not a complete list):

- **ActiveViewIndex** Enables you to select the View control to render by index.
- **Views** Enables you to retrieve the collection of View controls contained in the MultiView control

The MultiView control also supports the following methods:

- **GetActiveView** Enables you to retrieve the selected View control.
- **SetActiveView** Enables you to select the active view.

Finally, the MultiView control supports the following event:

- **ActiveViewChanged** Raised when a new View control is selected.

The View control does not support any special properties or methods. Its primary purpose is to act as a container for other controls. However, the View control does support the following two events:

- **ActivateRaised** when the view becomes the selected view in the MultiView control.
- **DeactivateRaised** when another view becomes the selected view in the MultiView control.

You can use the MultiView control to divide a large form into several sub-forms. You can associate particular commands with button controls contained in a MultiView. When the button is clicked, the MultiView changes the active view.

The MultiView control recognizes the following commands:

- **NextView :** Causes the MultiView to activate the next View control.
- **PrevView :** Causes the MultiView to activate the previous View control.
- **SwitchViewByID :** Causes the MultiView to activate the view specified by the button control's CommandArgument.
- **SwitchViewByIndex :** Causes the MultiView to activate the view specified by the button control's CommandArgument.

You can use these commands with any of the button controlsButton, LinkButton, and ImageButtonby setting the button control's **CommandName** property and, in the case of the SwitchViewByID and SwitchViewByIndex, by setting the **CommandArgument** property.

❑ Wizard Control :

The MultiView control enables you to hide and display different areas of a page. The Wizard control, like the MultiView control, can be used to divide a large form into multiple sub-forms. The Wizard control, however, supports many advanced features that are not supported by the MultiView control. The Wizard control contains one or more WizardStep child controls. Only one WizardStep is displayed at a time.

The Wizard control supports the following properties (this is not a complete list):

- **ActiveStep :** Enables you to retrieve the active WizardStep control.
- **ActiveStepIndex :** Enables you to set or get the index of the active WizardStep control.
- **CancelDestinationPageUrl :** Enables you to specify the URL where the user is sent when the Cancel button is clicked.
- **DisplayCancelButton :** Enables you to hide or display the Cancel button.
- **DisplaySideBar :** Enables you to hide or display the Wizard control's side bar. The side bar displays a list of all the wizard steps.
- **FinishDestinationPageUrl :** Enables you to specify the URL where the user is sent when the Finish button is clicked.
- **HeaderText :** Enables you to specify the header text that appears at the top of the Wizard control.

- **WizardSteps :** Enables you to retrieve the WizardStep controls contained in the Wizard control.

The Wizard control also supports the following templates:

- **FinishNavigationTemplate :** Enables you to control the appearance of the navigation area of the finish step.
- **HeaderTemplate :** Enables you control the appearance of the header area of the Wizard control.
- **SideBarTemplate :** Enables you to control the appearance of the side bar area of the Wizard control.
- **StartNavigationTemplate :** Enables you to control the appearance of the navigation area of the start step.
- **StepNavigationTemplate :** Enables you to control the appearance of the navigation area of steps that are not start, finish, or complete steps.

The Wizard control also supports the following methods:

- **GetHistory():** Enables you to retrieve the collection of WizardStep controls that have been accessed.
- **GetStepType():** Enables you to return the type of WizardStep at a particular index. Possible values are Auto, Complete, Finish, Start, and Step.
- **MoveTo():** Enables you to move to a particular WizardStep.

The Wizard control also supports the following events:

- **ActiveStepChanged :** Raised when a new WizardStep becomes the active step.
- **CancelButtonClick :** Raised when the Cancel button is clicked.
- **FinishButtonClick :** Raised when the Finish button is clicked.
- **NextButtonClick :** Raised when the Next button is clicked.
- **PreviousButtonClick :** Raised when the Previous button is clicked.
- **SideBarButtonClick :** Raised when a side bar button is clicked.

A Wizard control contains one or more WizardStep controls that represent steps in the wizard. The WizardStep control supports the following properties:

- **AllowReturn :** Enables you to prevent or allow a user to return to this step from a future step.
- **Name :** Enables you to return the name of the WizardStep control.
- **StepType :** Enables you to get or set the type of wizard step. Possible values are Auto, Complete, Finish, Start and Step.
- **Title :** Enables you to get or set the title of the WizardStep. The title is displayed in the wizard side bar.
- **Wizard :** Enables you to retrieve the Wizard control containing the WizardStep.

The WizardStep also supports the following two events:

- **Activate :** Raised when a WizardStep becomes active.
- **Deactivate :** Raised when another WizardStep becomes active.

The StepType property is the most important property. This property determines how a WizardStep is rendered. The default value of StepType is Auto. When StepType is set to the value

Auto, the position of the WizardStep in the WizardSteps collection determines how the WizardStep is rendered.

You can explicitly set the StepType property to a particular value. If you set StepType to the value Start, then a Previous button is not rendered. If you set the StepType to Step, then both Previous and Next buttons are rendered. If you set StepType to the value Finish, then Previous and Finish buttons are rendered. Finally, when StepType is set to the value Complete, no buttons are rendered.

❑ TREEVIEW CONTROL:

The TreeView Web server control is used to display hierarchical data, such as a table of contents or file directory, in a tree structure. It supports the following features:

- Automatic data binding, which allows the nodes of the control to be bound to hierarchical data, such as an XML document.
- Site navigation support through integration with the SiteMapDataSource control.
- Node text that can be displayed as either selectable text or hyperlinks.
- Customizable appearance through themes, user-defined images, and styles.
- Programmatic access to the TreeView object model, which allows you to dynamically create trees, populate nodes, set properties, and so on.
- Node population through client-side callbacks to the server (on supported browsers).
- The ability to display a check box next to each node.
- When TreeView is displayed for the first time, it displays all its nodes. However, it can be controlled by setting ExpandDepth property.

TREEVIEW NODE TYPE

- **Root** A node that has no parent node and one or more child nodes.
- **Parent** A node that has a parent node and one or more child nodes.
- **Leaf** A node that has no child nodes.

Properties of TreeView Control :

- **Nodes :** Gets a collection of TreeNode objects that represents the root nodes in the TreeView control.
- **NodeStyle :** Gets a reference to the TreeNodeStyle object that allows you to set the default appearance of the nodes in the TreeView control.
- **SelectedNode :** Gets a TreeNode object that represents the selected node in the TreeView control.
- **SelectedNodeStyle :** Gets the TreeNodeStyle object that controls the appearance of the selected node in the TreeView control.
- **SelectedValue :** Gets the value of the selected node.
- **ShowCheckBoxes :** Gets or sets a value indicating which node types will display a check box in the TreeView control.
- **ShowExpandCollapse :** Gets or sets a value indicating whether expansion node indicators are displayed.
- **ShowLines :** Gets or sets a value indicating whether lines connecting child nodes to parent nodes are displayed.
- **DataSourceID** : Indicates the data source to be used. (You can use .sitemap file as datasource).
- **Text** : Indicates the text to display in the node.

- **Tooltip :** Indicates the tooltip of the node when you mouse over.
- **Value** : Indicates the non displayed value (usually unique id to use in server side events)
- **NavigateUrl :** Indicates the target location to send to the user when node is clicked. If not set you can handle TreeView.SelectedNodeChanged event to decide what to do.
- **Target :** If NavigationUrl property is set, it indicates where to open the target location (in new window or same window).
- **ImageUrl :** Indicates the image that appears next to the node.

Styles of TreeView Control

- **NodeSpacing :** Space (in pixel) between current node and the node above or below it.
- **VerticalPadding :** Space (in pixel) between the top and bottom of the node text.
- **HorizontalPadding :** Space (in pixel) between the left and right of the node text.
- **ChildNodePadding :** Space (in pixel) between the parent node and its child node.

EVENTS

TreeView control events are raised only when a user interacts with the control by doing such things as selecting, expanding, or collapsing a node. They are not raised if the select, expand, or collapse methods are called programmatically.

- **TreeNodeCheckChanged :** Occurs when a check box of the TreeView control changes state between posts to the server. Occurs once for every TreeNode object that changes.
- **SelectedNodeChanged :** Occurs when a node is selected in the TreeView control.
- **TreeNodeExpanded :** Occurs when a node is expanded in the TreeView control.
- **TreeNodeCollapsed :** Occurs when a node is collapsed in the TreeView control.
- **TreeNodePopulate :** Occurs when a node, with its PopulateOnDemand property set to true, is expanded in the TreeView control.
- **TreeNodeDataBound :** Occurs when a data item is bound to a node in the TreeView control.

❑ MENU CONTROL:

The ASP.NET Menu control allows you to develop both statically and dynamically displayed menus for your ASP.NET Web pages. You can configure the contents of the Menu control directly in the control, or you can specify the contents by binding the control to a data source. Without writing any code, you can control the appearance, orientation, and content of an ASP.NET Menu control.

Static Display and Dynamic Display
The Menu control has two modes of display: static and dynamic. Static display means that the Menu control is fully expanded all the time. The entire structure is visible, and a user can click on any part. In a dynamically displayed menu, only the portions you specify are static, while their child menu items are displayed when the user holds the mouse pointer over the parent node.

Properties :

- **DataSource :** Gets or sets the object from which the data-bound control retrieves its list of data items..
- **DisappearAfter :** Gets or sets the duration for which a dynamic menu is displayed after the mouse pointer is no longer positioned over the menu.

- **Items :** Gets a MenuItemCollection object that contains all menu items in the Menu control.
- **Orientation :** Gets or sets the direction in which to render the Menu control.
- **PathSeparator :** Gets or sets the character used to delimit the path of a menu item in a Menu control.
- **SelectedItem :** Gets the selected menu item. We can use following runtime property with this. Text, Value, ValuePath, Depth, NavigateUrl, Parent, Selected, ChildItems.
- **SelectedValue :** Gets the value of the selected menu item.

Event :

- **MenuItemClick :** Occurs when a menu item in a Menu control is clicked.

❑ SITEMAP CONTROL:

Displays a set of text or image hyperlinks that enable users to more easily navigate a Web site, while taking a minimal amount of page space. A Sitemap represent the relationship between the pages in an application. The SiteMapPath is useful for sites that have deep hierarchical page structures.

The SiteMapPath control works directly with your Web site's site map data. If you use it on a page that is not represented in your site map, it will not be displayed. The SiteMapPath relies on a SiteMapProvider to fetch the data. It contains an XML (sitemap) file which contains URLs and other information about the pages to be displayed.

Nodes

The SiteMapPath is made up of nodes. Each element in the path is called a node and is represented by a SiteMapNodeItem object. The node that anchors the path and represents the base of the hierarchical tree is called the root node. The node that represents the currently displayed page is the current node. Any other node between the current node and root node is a parent node.

Example of Sitemap file :

```
<siteMapNode url="default.aspx" title="Home Page"  description="">
<siteMapNode url="page1.aspx" title="Page1"  description="" />
<siteMapNode url="page2.aspx" title="Page2"  description="" />
<siteMapNode url="Default2.aspx" title="Default2"  description="" />
</siteMapNode>
```

EXAMPLES :

FileUpload :

```
protected void UploadButton_Click(object sender, EventArgs e)
{
    if(FileUploadControl.HasFile)
    {
        try
        {
            if(FileUploadControl.PostedFile.ContentType == "image/jpeg")
            {
                if(FileUploadControl.PostedFile.ContentLength < 102400)
                {
                    string filename = Path.GetFileName(FileUploadControl.FileName);
                    FileUploadControl.SaveAs(Server.MapPath("~/") + filename);
                    StatusLabel.Text = "Upload status: File uploaded!";
                }
                else
                    StatusLabel.Text = "Upload status: The file has to be less than
                                        100 kb!";
            }
            else
                StatusLabel.Text = "Upload status: Only JPEG files are accepted!";
        }
        catch(Exception ex)
        {
            StatusLabel.Text = "Upload status: The file could not be uploaded. The
                                following error occured: " + ex.Message;
        }
    }
}
```

- **Upload file after checking its extension…**

```
String str = Path.GetExtension(FileUpload1.FileName);
if(str.Equals(“.jpg”)
{
   FileUpload1.SaveAs(Server.MapPath("~/images/") + filename);
   StatusLabel.Text = "Upload status: File uploaded!";
}
else
{
   StatusLabel.Text = "Upload status: The file is not type of JPEG!";
}
```

Note : Default Upload file size is 4MB. To Change it use **web.config.comments** file. In that file search **<httpRuntime>** tag. It contains the details of file being uploaded.

Calender :

Add this event in source tag for calendar control.

```
OnDayRender="Calendar1_DayRender" OnSelectionChanged="Calendar1_SelectionChanged"
OnVisibleMonthChanged="Calendar1_VisibleMonthChanged"
```

Code Behind File:

```
Hashtable HolidayList;     //Declare Globally in Page Class

protected void Page_Load(object sender, EventArgs e)
{
        HolidayList = Getholiday();
        Calendar1.FirstDayOfWeek = FirstDayOfWeek.Sunday;
```

```
        Calendar1.NextPrevFormat = NextPrevFormat.ShortMonth;
        Calendar1.TitleFormat = TitleFormat.Month;
        Calendar1.ShowGridLines = true;
}

private Hashtable Getholiday()
{
        Hashtable holiday = new Hashtable();
        holiday["1/1/2011"] = "New Year";
        holiday["1/8/2011"] = "Muharam";
        holiday["1/26/2011"] = "Republic Day";
        holiday["2/23/2011"] = "Maha Shivaratri";
        holiday["3/21/2011"] = "Holi";
        holiday["4/3/2011"] = "Ram Navmi";
        holiday["6/24/2011"] = "Rath yatra";
        holiday["8/14/2011"] = "Janmashtami";
        holiday["8/15/2011"] = "Independence Day";
        holiday["9/28/2011"] = "Vijaya Dasami (Dusherra)";
        holiday["10/17/2011"] = "Diwali & Govardhan Puja";
        holiday["10/19/2011"] = "Bhaidooj";
        holiday["12/25/2011"] = "Christmas";
        return holiday;
}

protected void Calendar1_SelectionChanged(object sender, EventArgs e)
{
               LabelAction.Text = "Date changed to :" +
               Calendar1.SelectedDate.ToShortDateString();
}

protected void Calendar1_VisibleMonthChanged(object s,MonthChangedEventArgs e)
{
          LabelAction.Text = "Month changed to :" + e.NewDate.ToShortDateString();
}

protected void Calendar1_DayRender(object sender, DayRenderEventArgs e)
{
          if (HolidayList[e.Day.Date.ToShortDateString()] != null)
          {
            //  Label label1 = new Label();
            //  label1.Text = (string)HolidayList[e.Day.Date.ToShortDateString()];
            //  label1.Font.Size = new FontUnit(FontSize.Small);
            //  e.Cell.Controls.Add(label1);
                e.Cell.Text = e.Day.DayNumberText + “<br>” +
                              HolidayList[e.Day.Date.ToShortDateString()];
          }
}
```

Introduction to ADO.NET

ADO.NET allows you to interact with relational databases and other data sources. Quite simply, ADO.NET is the technology that ASP.NET applications use to communicate with a database.

Almost every piece of software ever written works with data. In fact, a typical Internet application is often just a thin user interface shell on top of a sophisticated database program that reads and writes information from a database on the web server. At its simplest, a database program might allow a user to perform simple searches and display results in a formatted table. A more sophisticated ASP.NET application might use a database behind the scenes to retrieve information, which is then processed and displayed in the appropriate format and location in the browser. The user might not even be aware (or
care) that the displayed information originates from a database. Using a database is an excellent way to start dividing the user interface logic from the content, which allows you to create a site that can work with dynamic, easily updated data.

ADO.NET has a few characteristics that make it different from previous data access technologies (such as ADO, the database library that was used in classic ASP pages).

The DataSet

Many ADO.NET tasks revolve around a new object called the DataSet. The DataSet is a cache of information that has been queried from your database. The innovative features of the DataSet are that it's disconnected (see the next section) and can store more than one table. For example, a DataSet could store a list of customers, a list of products, and a list of customer orders. You can even define all these relationships in the DataSet to prevent invalid data and make it easier to answer questions such as "What products did Joe Smith order?"

Disconnected Access

Disconnected access is the one of the most important characteristics of ADO.NET and perhaps the single best example of the new .NET philosophy for accessing data. With previous database access technologies, it's easy to hold open a connection with the database server while your code does some work. This live connection allows you to make immediate updates, and you can even see the changes made by other users in real time. Unfortunately, database servers can provide only a limited number of connections before they reject connection requests. The longer you keep a connection open, the greater the chance becomes that another user will be prevented from accessing the database. In a poorly written program, the database connection is kept open while other tasks are being performed. But even in a well-written program using an old data access technology such as ADO, the connection must be kept open until all the data is processed and the query results are no longer needed.

ADO.NET has an entirely different philosophy. In ADO.NET you still create a connection to a database, but you're able to close the connection much faster. That's because you're able to fill a DataSet object with a *copy* of the information drawn from the database. You can then close the connection before you start processing the data. This means you can easily process and manipulate the data without worrying, because you aren't using a valuable database connection. (Of course, if you change the information in the DataSet, the information in the corresponding table in the database isn't changed. You'll need to reconnect to commit any changes.)

XML Integration

ADO.NET has deep support for XML. This fact isn't immediately obvious when you're working with a DataSet object, because you'll usually use the built-in methods and properties of the DataSet to perform all the data manipulation you need. But if you delve a little deeper, you'll discover that you can access the information in the DataSet as an XML document. You can even modify values, remove rows, and add new records by modifying the XML, and the DataSet will be updated automatically.

➔ ADO.NET Basics

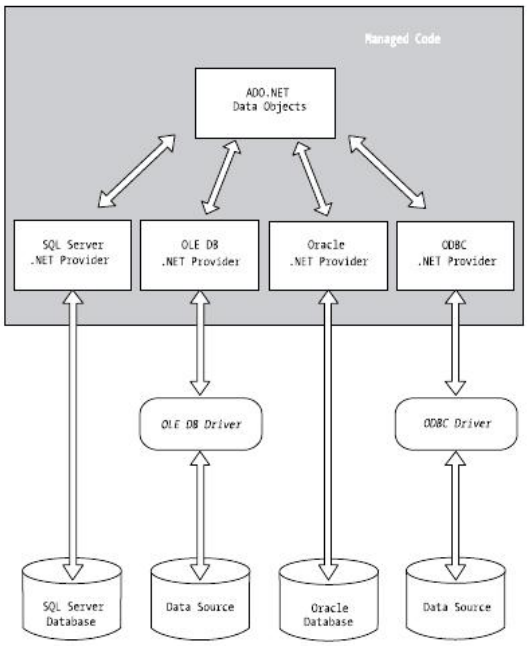
ADO.NET relies on the functionality in a small set of core objects. You can divide these objects into two groups: those that are used to contain and manage data (such as DataSet, DataTable, DataRow and DataColumn) and those that are used to connect to a specific data source (such as Connection, Command and DataReader).

Each set of data interaction objects is called an ADO.NET **data provider**. Data providers are customized so that each one uses the best-performing way of interacting with its data source.

Microsoft includes the following four providers:

- **SQL Server provider:** Provides optimized access to a SQL Server database (version 7.0 or later).
- **OLE DB provider:** Provides access to any data source that has an OLE DB driver.
- **Oracle provider:** Provides optimized access to an Oracle database (version 8i or later).
- **ODBC provider:** Provides access to any data source that has an ODBC (Open Database Connectivity) driver.

Following is the figure which shows how all providers



In addition, third-party developers and database vendors have released their own ADO.NET providers, which follow the same conventions and can be used in the same way as those that are included with the .NET Framework.

When choosing a provider, you should first try to find one that's customized for your data source. If you can't find a suitable provider, you can use the OLE DB provider, as long as you have an OLE DB driver for your data source. The OLE DB technology has been around for many years as part of ADO, so most data sources provide an OLE DB driver (including SQL Server, Oracle, Access, MySQL, and many more). In the rare situation that you can't find a full provider or an OLE DB driver, you can fall back on the ODBC provider, which works in conjunction with an ODBC driver.

➔ Data Namespaces

The ADO.NET components live in seven namespaces in the .NET class library. Together, these namespaces hold all the functionality of ADO.NET. Following tables shows Data namespaces along with their purposes.

Namespace	Purpose
System.Data	Contains fundamental classes with the core ADO.NET functionality. This includes DataSet and DataRelation, which allow you to manipulate structured relational data. These classes are totally independent of any specific type of database or the way you use to connect to it.
System.Data.Common	These classes aren't used directly in your code. Instead, they are used by other data provider classes that inherit from them and provide versions customized for a specific data source.
System.Data.OleDb	Contains the classes you use to connect to an OLE DB data source, including OleDbCommand and OleDbConnection.
System.Data.SqlClient	Contains the classes you use to connect to a Microsoft SQL Server database (version 7.0 or later). These classes, such as SqlCommand and SqlConnection, provide all the same properties and methods as their counterparts in the System.Data.OleDb namespace. The only difference is that they are optimized for SQL Server and provide better performance by eliminating the extra OLE DB layer (and by connecting directly to the optimized TDS interface).
System.Data.SqlTypes	Contains structures for SQL Server–specific data types such as SqlMoney and SqlDateTime. You can use these types to work with SQL Server data types without needing to convert them into the standard .NET equivalents (such as System.Decimal and System.DateTime). These types aren't required, but they do allow you to avoid any potential rounding or conversion problems that could adversely affect data.
System.Data.OracleClient	Contains the classes you use to connect to an Oracle database, such as OracleCommand and OracleConnection.
System.Data.Odbc	Contains the classes you use to connect to a data source through an ODBC driver. These classes include OdbcCommand and OdbcConnection.

Each provider designates its own prefix for naming objects. Thus, the SQL Server provider includes SqlConnection and SqlCommand objects, and the Oracle provider includes OracleConnection and OracleCommand objects. Internally, these objects work quite differently, because they need to connect to different databases using different low level protocols. Externally, however, these objects look quite similar and provide an identical set of basic methods because

they implement the same common interfaces. This means your application is shielded from the complexity of different standards and can use the SQL Server provider in the same way the Oracle provider uses it. In fact, you can often translate a block of code for interacting with a SQL Server database into a block of Oracle-specific code just by renaming the objects.

Data Provider objects provided by ADO.NET

	SQL Server .NET Provider	OLEDB .NET Provider	Oracle .NET Provider	ODBC .NET Provider
Connection	SqlConnection	OleDbConnection	OracleConnection	OdbcConnection
Command	SqlCommand	OleDbCommand	OracleCommand	OdbcCommand
DataReader	SqlDataReader	OleDbDataReader	OracleDataReader	OdbcDataReader
DataAdaptor	SqlDataAdaptor	OleDbDataAdaptor	OracleDataAdaptor	OdbcDataAdaptor

➔ Direct Data Access

The easiest way to access data is to perform all your database operations directly and not worry about maintaining disconnected information. This model is closest to traditional ADO programming, and it allows you to sidestep potential concurrency problems, which occur when multiple users try to update information at once. It's also well suited to ASP.NET web pages, which don't need to store disconnected information for long periods of time. Remember, an ASP.NET web page is loaded when the page is requested and shut down as soon as the response is returned to the user. That means a page typically has a lifetime of only a few seconds.

Simple data access is ideal if you need only to read information or if you need to perform only simple update operations, such as adding a record to a log or allowing a user to modify values in a single record (for example, the customer information for an e-commerce site). Simple data access may not be as useful if you want to modify several different records or tables at the same time.

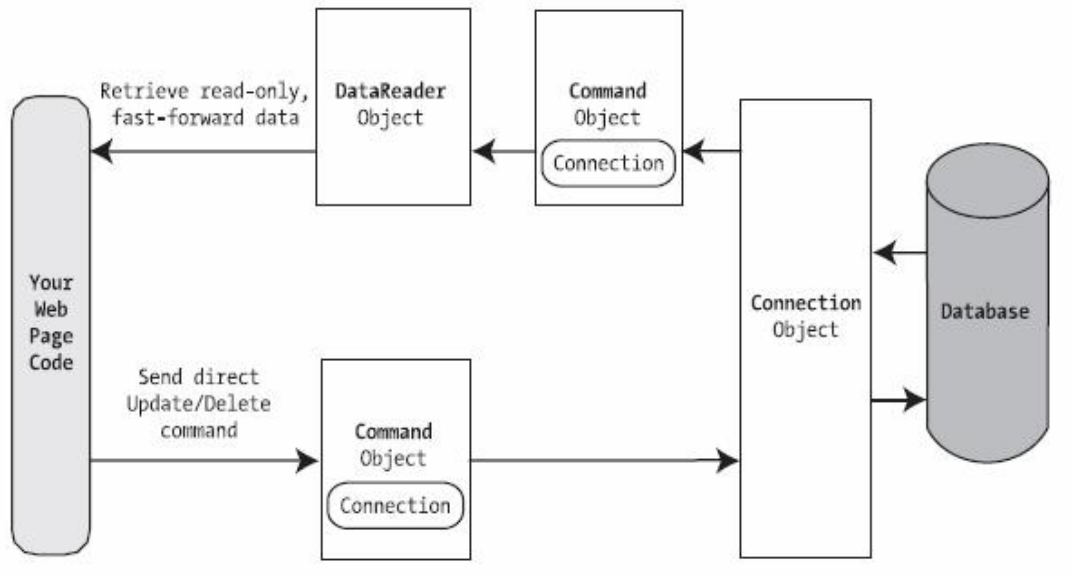
To retrieve information with simple data access, follow these steps:

1. Create Connection, Command, and DataReader objects.
2. Use the DataReader to retrieve information from the database, and display it in a control on a web form.
3. Close your connection.
4. Send the page to the user. At this point, the information your user sees and the information in the database no longer have any connection, and all the ADO.NET objects have been destroyed.

To add or update information, follow these steps:

1. Create new Connection and Command objects.
2. Execute the Command (with the appropriate SQL statement).

Following is the diagram showing direct access with ADO.NET



Creating Connection

To do any operation on database, the first step is to get the connection. When creating a Connection object, you need to specify a value for its Connection-String property. This ConnectionString defines all the information the computer needs to find the data source, log in, and choose an initial database.

As specified earlier, you can use different ways to perform your database operations. You can use OleDb connection method to connect database or can choose specific provider. If you have specific provider, its better to use that one instead of using OleDb, because specific provider speeds up rather then OleDb.

Following is an example of creating connection using OleDb. You need to specify provider if you are using OleDb connection.

```
String x = "Provider=SQLOLEDB.1; Data Source=ASHISH\TEST; Initial Catalog=TESTDB;
           Integrated Security=SSPI";
OleDbConnection con = new OleDbConnection(x);
```

Following is an example of creating connection using SqlConnection. You do not need to specify provider as it is specially used to connect sql server.

```
String x = "Data Source=ASHISH\TEST; Initial Catalog=TESTDB; Integrated
           Security=SSPI";
SqlConnection con = new SqlConnection(x);
```

Instead of specifying connection string each time, you can specify a global application variable in Web.config file, which you can retrieve in any page. Web.config file has a special tag under <configuration> tag. Inside <configuration> tag, you can specify <connectionStrings>, which can be used in all application to connect database. You can specify more then one connection string variables in this section. Following is a peace of code that can be specified under Web.config file.

```
<configuration xmlns=http://schemas.microsoft.com/.NetConfiguration/v2.0>
    <connectionStrings>
        <add name="a" connectionString="Data Source=ASHISH\TEST;
                Initial Catalog=TESTDB; Integrated Security=SSPI" />
    </connectionStrings>
</configuration>
```

Later, when you want to connect database in any form, you can use following code to use the "a" variable specified under Web.config file.

```
String x = WebConfigurationManager.ConnectionStrings["a"].ConnectionString;
SqlConnection con = new SqlConnection(x);
```

➔ Using Data Reader to Fetch Data

Following are the steps to use Data Reader for retrieving record using Data Reader.

- Create and Open Connection
- Create Command object and specify Select command
- Create Data Reader object and assign ExecuteReader() method of command.
- Repeat through loop using Read() method of data reader.
- Close Data Reader and Connection.

Consider following code to retrieve data using Data Reader.

```
protected void Button1_Click(object sender, EventArgs e)
{
     String x = "Data Source=ASHISH\\SQLTEST; Initial
         Catalog=TESTDB; User Id=sa; Password=sasa";
     SqlConnection con = new SqlConnection(x);
     con.Open();
     SqlCommand cmd = new SqlCommand("select * from
         student", con);

     SqlDataReader dr;
     dr = cmd.ExecuteReader();

     while (dr.Read())
     {
         Response.Write(dr["studid"].ToString());
         Response.Write(dr["name"].ToString());
         Response.Write(dr["sex"].ToString());
         Response.Write(dr["age"].ToString());
         Response.Write(dr["city"].ToString());
         Response.Write("<hr>");
     }
     dr.Close();
     con.Close();
}
```

Consider following code to fill List Box using Data Reader.

```
protected void Button2_Click(object sender, EventArgs e)
{
    String x = "Data Source=ASHISH\\SQLTEST; Initial
   Catalog=TESTDB; User Id=sa; Password=sasa";

    SqlConnection con = new SqlConnection(x);
    con.Open();
    SqlCommand cmd = new SqlCommand("select name from
        student", con);

    SqlDataReader dr;
    dr = cmd.ExecuteReader();

    lst_City.Items.Clear();
    while (dr.Read())
    {
        lst_City.Items.Add(dr["name"].ToString());
    }

    lst_City.SelectedIndex = 0;

    dr.Close();
    con.Close();
}
```

➔ Using Data Reader to Update Data

To execute an Update, Insert, or Delete statement, you need to create a Command object. You can then execute the command with the ExecuteNonQuery() method. This method returns the number of rows that were affected, which allows you to check your assumptions. For example, if you attempt to update or delete a record and are informed that no records were affected, you probably have an error in your Where clause that is preventing any records from being selected. (If, on the other hand, your SQL command has a syntax error or attempts to retrieve information from a nonexistent table, an exception will occur.)

(Beginning ASP.NET Page No. 500 to 518)

Binding Data to Web Controls

Most software applications involve, in one way or another, data access and reporting. Web applications in particular must perform these two functions, so Web programmers are looking for any automated (or even semi-automated) method of associating rows of data with graphical HTML elements such as drop-down lists or tables. The answer to this growing demand for automatic binding between data sources and graphical elements is .NET data bound controls.

Data bound controls play a key role in the development of .NET applications because they allow you to associate controls and their individual properties with one or more fields in any .NET-compliant data source. The power of .NET is the versatility of these data bound controls. You use them in the same way regardless of the programming model—Microsoft Windows Forms, Microsoft Web Forms, or Microsoft .NET Web services.

In this chapter, we'll discuss the two levels of data binding, simple and complex, as well as the most important Web controls that can be data bound. Data binding and data bound Web controls are the focus of any Active Server Pages (ASP) .NET application, so we'll explore them at length in the next few chapters. (Web controls are also referred to as ASP.NET controls, server controls, and .NET Web controls.) Getting a handle on the key programming techniques is vital for building effective Web solutions.

ASP.NET Data Binding

Data binding is the process of retrieving data from a source and dynamically associating it to a property of a visual element. Depending on the context in which the element will be displayed, you can map the element to either an HTML tag or a .NET Web control.

In ASP.NET, the atomic elements that work together to generate page output are the Web controls. In ASP.NET data binding, the application retrieves data from any .NET-compliant data source and then allows you to selectively manipulate and assign this data to properties of controls that have been specially designed to support data binding. These controls are known as data bound controls. ASP.NET Web server controls can be bound to a data source by using a heterogeneous bunch of properties, including Text, DataSource, and DataTextField. Later in the chapter, you'll see how to handle binding by using a combination of these and other control-specific properties.

The DataSource property in particular lets you specify the data source object to which the control is linked. Notice that this link is logical and does not result in any underlying operation until you explicitly call another method. Not all properties in a Web control programming interface can be data bound. You activate data binding on a control by calling the method DataBind. When this method executes, the control loads data from the associated data source, evaluates the data bound properties, and redraws its user interface to reflect changes.

Feasible Data-Binding Sources

Many .NET classes can be used as data sources—not just those dealing with database contents. In general, any object that exposes the ICollection interface is a potential source of data binding. The ICollection interface defines size, enumerators, and synchronization methods for all .NET collections. As a result, you can bind a Web control to any of the following data structures:

- In-memory .NET collection classes including arrays, dictionaries, sorted and linked lists, hash tables, stacks, and queues.

- User-defined data structures, as long as the structure exposes ICollection or one of the interfaces based on it (such as IList).
- Database-oriented classes such as DataTable and DataSet. Bear in mind that in general a DataSet object can contain multiple tables. In this case, you would assign the DataSet object to the DataSource property and assign the name of the selected DataTable object to the control's DataMember property.
- Filtered views built on top of DataTable objects that show only a subset of the contained rows. Views are represented by a DataView class or any class that you have built from a DataTable object.

> You cannot directly bind XML documents unless you load their contents in one of the classes mentioned in the preceding bulleted list. You can bind XML documents in two ways. You can load the XML document into a DataSet object and then bind the DataSet object. Alternatively, you can parse the XML document with an XML reader class and bind the collection of nodes you obtain.

Simple Data Binding

In ASP.NET, simple data binding is a connection between one piece of data and a server control property. This connection is established through a special expression that is evaluated when the code in the page being processed calls the DataBind method either on the Page object or on the control.

A data-binding expression is any text enclosed in angle brackets and percent signs (<%…%>) and prefixed by the number symbol (#). You can include a data-binding expression as the value of an attribute in the opening tag of a server control or anywhere else on the page. You cannot programmatically assign a data-binding expression to a property of a Web control. If the data-binding expression contains quotation marks, use a single quotation mark (') to wrap the whole expression, as shown here:

```
theLabel.Text='<%# "Hello, world" %>';
```

Within the delimiters, you can invoke user-defined page methods, static methods and properties, and methods of any other page components. The following code shows a label control whose Text property is bound to the name of the currently selected element in a drop-down list control:

```
<asp:label runat="server" Text='<%# dropdown.SelectedItem.Text %>' />
```

The data-binding expression can accept a minimal set of operators, mostly for concatenating subexpressions. When you need more advanced processing and are using external arguments, resort to a user-defined method. The only requirement for a user-defined method is to be callable within the page. When you plan to use code that is not resident in the .ASPX page (for example, if you use code-behind techniques), make sure the method is declared public or protected.

Let's walk through an example that illustrates how to arrange a form-based record viewer by using simple data binding with text boxes. (The full source code for the FormViewer.aspx application is available on the companion CD.) The user interface supplies three text boxes, one for each of the retrieved fields. The data source is the Employees table of the Northwind database in SQL Server 2000. The query selects three fields: employeeid, firstname, and lastname. The binding takes place between the text boxes and the fields. Only one record at a time is displayed; the user clicks the Previous and Next buttons to move through the data. You can see what the application looks like in Figure 1-1.

The binding expression employs a user-defined method named GetBoundData.

```
<asp:textbox runat="server" id="txtID" cssclass="StdTextBox"
    Text = '<%# GetBoundData("employeeid") %>' />
```

GetBoundData takes the name of the field to display and retrieves the position of the current record from the ASP.NET Session object. It then retrieves the value and passes it to the ASP.NET run time for actual rendering.

```
public String GetBoundData(String fieldName)
{
    DataSet ds = (DataSet) Session["MyData"];
    DataTable dt = ds.Tables["EmpTable"];
    int nRowPos = (int) Session["CurrentRecord"];

    String buf = dt.Rows[nRowPos][fieldName].ToString();
    return buf;
}
```

Complex Data Binding

Complex data binding occurs when you bind a list control or an iterative control to one or more columns of data. List controls comprise DropDownList, CheckBoxList, RadioButtonList, and ListBox. Iterative controls are Repeater, DataList, and DataGrid. The characteristics of ASP.NET iterative controls are briefly summarized in Table 1-1.

Table 1-1 ASP.NET Iterative Controls

Control	Description
Repeater	Creates a custom layout for displaying the control's contents by repeating a specified template for each item in the bound list. You define an ASP.NET template for various categories of items (header, footer, item, separator, and so forth), and the control applies it repeatedly in the page. The Repeater control has no built-in layout, so you must explicitly declare all ASP.NET formatting and style tags.
DataList	Displays the contents of a data bound list through ASP.NET templates. The DataList control supports selecting and editing. The appearance of the DataList control may be customized to some extent by setting style properties for different parts of the control. Compared to the Repeater control, DataList shows off predefined layouts, more advanced formatting capabilities, plus support for selecting and editing.
DataGrid	The DataGrid control renders a multi-column, data bound grid of data. It allows you to define various types of columns and control the layout of the resulting cells through predefined layouts and ASP.NET templates. You can also add specific functionality such as edit button columns and link columns. The DataGrid control also supports a variety of options for paging through data.

List controls have a more complex and versatile user interface than labels and text boxes. Because a list control handles (or at least has in memory) more items simultaneously than a label or a text box, you should associate the list control explicitly with a collection of data—that is, the data source. Depending on its expected behavior, a list control will select the needed items from memory and properly format and display them.

Iterative controls take a data source, loop through its items, and apply HTML templates to its rows. This basic behavior is common to all three ASP.NET iterative controls. The controls differ in their layout capabilities and the functionality they support.

The DropDownList Web Control

The DropDownList Web control enables users to select from a single-selection drop-down list (for example, a combo box). You can control the look of the DropDownList control by setting its height and width in pixels, but you cannot control the number of items displayed when the list drops down. The SelectedIndex and SelectedItem properties provide details about the currently selected element. The DropDownList control supports data binding by using the following five properties: DataSource, DataMember, DataTextField, DataValueField, and DataTextFormatString. As shown in the following code, of these five properties, only DataSource, DataTextField, and DataValueField have a corresponding ASP.NET attribute you can declare:

```
<asp:DropDownList id="DropDownList1" runat="server"
    DataSource="<%# databindingexpression %>"
    DataTextField="DataSourceField"
    DataValueField="DataSourceField">
```

You can also use a data-binding expression to set the DataSource property. The expression you use must evaluate to a .NET object that exposes the ICollection interface. You cannot use expressions to set the other properties. The values for DataTextField and DataValueField each must match the name of one field in the data source, so you cannot assign DataTextField by combining two or more fields in the data source. Let's see how to work around this limitation.

Suppose you want a drop-down list to display both the first name and the last name of each employee in the company. You could ask SQL Server to return a calculated column that has the required format. In this case, you would use a query string, as shown in the following code, and set DataTextField to the name of the precalculated field:

```
SELECT lastname + ', ' + firstname AS 'EmployeeName'
    FROM Employees
```

You can obtain the same result, however, without the involvement of SQL Server. After you hold a DataTable object that contains the result of the query, you can add a new column on the fly. The content of the column is determined by the expression you use. The following code uses this approach to generate the drop-down list shown in Figure 1-2:

You can create a drop-down list box such as this as you work by using the DataTable object.

The next code example shows how to bind the drop-down list control named EmpList to the dynamically created EmployeeName field:

```
EmpList.DataSource = oDS.Tables["EmployeesList"].DefaultView;
EmpList.DataTextField = "EmployeeName";
EmpList.DataValueField = "employeeid";
```

DataTextField links the Text property of any individual item in the list with the EmployeeName field. DataValueField ensures that the Value property of each item is set with the value stored in the corresponding record of the employeeid field. The full source code for the DropDown.aspx application is available on the companion CD.

An expression-based column does not have to be filled explicitly. Whenever the program needs to read the value of one of its rows, it just evaluates the expression as it processes data and then takes the result. Note that expressions can reference other expression columns. Circular references, though, are not allowed. The ADO.NET run time promptly detects them and raises an exception.

Which of the two approaches is preferable? Should you ask SQL Server to return a made-to-measure column, or should you create the extra column you need when you need it? The final result of both approaches is identical, but the cost of using each is not. SQL Server is not as efficient. It returns a new column of potentially duplicated data. Processing data to produce a column creates overhead, especially when the expression is complex. Unless the precalculated column is needed for further processing, you should avoid using SQL Server for this type of operation.

The CheckBoxList Web Control

The CheckBoxList control is a single control that acts as a parent control for a collection of checkable list items, each of which is rendered through an individual CheckBox control. You insert a check box list as follows:

```
<asp:CheckBoxList runat="server" id="cbEmployees">
```

The CheckBoxList control supports data binding by using five properties: DataSource, DataMember, DataTextField, DataValueField, and DataTextFormatString. You bind the CheckBoxList control to a data source by using the following code:

```
cbEmployees.DataSource = oDS.Tables["EmployeesList"];
cbEmployees.DataTextField = "lastname";
cbEmployees.DataValueField = "employeeid";
```

The properties of the CheckBoxList control play the same role as the properties for the DropDownList control, which we discussed earlier in the chapter. Unlike the DropDownList control, however, the CheckBoxList control does not supply properties that reveal information about the selected items. As you know, this information is vital for any Web application that uses checkable elements.

At any time, CheckBoxList can have one or more items selected. How can you retrieve these items? All list controls have an Items property that contains the collection of the child items. The Items property is implemented through the ListItemCollection class, and each list item can be accessed via a ListItem object. The following code loops through the items stored in a CheckBoxList control and checks the Selected property of each. (Selected is a Boolean property that indicates whether the item is selected.) The code then uses the StringBuilder object to concatenate the text of each selected item into a single comma-separated string.

```
StringBuilder buf = new StringBuilder("");
foreach(ListItem item in chkList.Items)
{
    if (item.Selected)
    {
        buf.Append(item.Text);
        buf.Append(", ");
    }
}
```

In .NET, the String object is immutable, so each time you modify a string, a brand new String object is created. This might have an unpleasant impact on performance, especially when repeated modifications to a string are required. Repeated modifications typically occur when you loop through potentially lengthy data sources to build up a summary string. The StringBuilder class solves the problem by letting you modify a string without creating a new object.

Notice in the following code that the ASP.NET CheckBox control is a bit more powerful than the ASP.NET counterpart of the well-known HTML <input> tag. The ASP.NET CheckBox control also supplies the Text property that allows you to automatically associate the check box with descriptive text.

```
<asp:checkbox runat="server" id="theCheckBox"
    Text="Click me..." />
```

When the preceding code is processed by the ASP.NET run time, the ASP.NET CheckBox control generates the following HTML code:

```
<input id="theCheckBox" type="checkbox" name="theCheckBox" />
<label for="theCheckBox">Click me...</label>
```

A slick trick used by ASP developers to quickly obtain the selected items of a logically related group of check boxes does not work in ASP.NET. When you have several check boxes with identical names in an ASP application, you can use a single line of code to obtain the corresponding values of check boxes that are selected when a form is posted.

```
<input type="checkbox" name="foo" value="...">
```

After the form values in the above code are posted, the following line of code written in Microsoft Visual Basic, Scripting Edition (VBScript) sets an array with all the checked values:

```
<input type="checkbox" name="foo" value="...">
```

Request.Form("foo") returns a comma-separated string formed by the value strings of all checked items. You then pass this string to VBScript's Split function, which transforms the string into an array.

This code won't work in ASP.NET if you use the CheckBox server control to render the check box component, but it will work if you use the <input> tag. However, using the CheckBoxList control is the preferred ASP.NET way to handle lists of check boxes.

The HTML tag <input> and all other HTML tags are rendered through an ASP.NET control taken from the System.Web.UI.HtmlControls namespace. The HTML check box element is rendered through the ASP.NET HtmlInputCheckBox class.

The RadioButtonList Web Control

The RadioButtonList control acts as a parent control for a collection of radio button list items. All child items of RadioButtonList are rendered by using a RadioButton control. A RadioButtonList control can have either no items or one item selected. The SelectedItem property returns the selected element as a ListItem object. However, you have no guarantee that only one item is selected at any given time. For this reason, be extremely careful when you access the

SelectedItem property of a RadioButtonList control. Depending on the structure and flow of your code, the item returned could be a null object. A workaround that always shields you from unpleasant surprises is wrapping any call to SelectedItem in a try/catch block, but don't abuse this technique because a try/catch block is always costly. An even better solution is to design your code to eliminate the risk of returning an unselected item.

The contents of a RadioButtonList control can be obtained from a data source, as shown here:

```
<asp:RadioButtonList id="rbEmployees" runat="server"
    DataValueField="employeeid"
    DataTextField="employeename" />
```

You also need to programmatically set the DataSource property as follows:

```
rbEmployees.DataSource = myDataTable;
```

The RadioButtonList control supports data binding by using five properties: DataSource, DataMember, DataTextField, DataValueField, and DataTextFormatString. These properties behave in the same way as the properties of controls discussed earlier in the chapter.

Programmers and designers who work with data bound controls in lists are always greatly concerned about content presentation. A list of items can flow horizontally or vertically, can be expressed in a fixed number of columns or rows, and so on.

Some Web controls, such as RadioButtonList, accept layout directives. For example, you can control how a list is rendered by using the RepeatLayout and/or RepeatDirection properties of RadioButtonList. RepeatLayout governs the default layout in which the list items are rendered within a table, ensuring the vertical alignment of the companion text. The alternative is displaying the items as free HTML text using blanks and breaks to guarantee a minimum of structure. With or without a tabular structure, RepeatDirection controls how the items flow. Feasible values for RepeatDirection are Vertical (the default) and Horizontal. The RepeatColumns property determines how many columns the list should have. The default value of 0 indicates that all items are displayed in a single row. The direction of that row—vertical or horizontal—depends on the value of RepeatDirection.

The ListBox Web Control

The ListBox control represents a vertical sequence of items displayed in a scrollable window. As shown in the following code, ListBox allows single or multiple item selection and exposes its contents by using the familiar Items collection:

```
<asp:listbox runat="server" id="theListBox"
    rows="5" selectionmode="Multiple" />
```

You can determine the height of the control by using the Rows property. As you might have guessed, the height is measured in numbers of rows rather than in pixels or percentages.

When it comes to data binding, the ListBox control behaves like the other controls discussed earlier in the chapter—that is, it supports properties such as DataSource, DataMember, DataTextField, DataValueField, and DataTextFormatString, and it can be bound to a data source and show its contents, as follows:

```
lstEmp.DataSource = ds.Tables["EmpTable"];
lstEmp.DataTextField = "lastname";
lstEmp.DataValueField = "employeeid";
lstEmp.DataBind();
```

The next code snippet illustrates how to write a comma-separated string with the values of the selected items. (This code is nearly identical to the code you would write to accomplish the same operation for a CheckBoxList control.)

```
lstEmp.DataSourc
public void ShowSelectedItems(Object sender, EventArgs e)
{
    StringBuilder sb = new StringBuilder("");
    for (int i=0; i < lstEmp.Items.Count; i++)
    {
        if (lstEmp.Items[i].Selected)
        {
            sb.Append(lstEmp.Items[i].Text);
            sb.Append(", ");
        }
    }
    lblResults.Text = sb.ToString();
}
```

This sample application illustrates the use of a data bound, multi-selection list box.

The programming interface of the list box also contains a SelectedItem property, which at first appears to make little sense because you are working with a multi-selection control. In this case, however, the SelectedItem property returns the selected item with the lowest index.

Using the Repeater, DataList and DataGrid Control

The Repeater Control

The Repeater control is a simple container control that binds to a list of items. It walks through the bound items and produces graphical elements according to a basic rendering algorithm and the HTML templates you supply. The Repeater control supports from one through five templates. These templates, which form a tabular structure, are described in Table.

Templates Supported by the Repeater Control	
Template	Description
HeaderTemplate	Determines the heading of the final output. It is rendered only once and prior to any row. It cannot contain data bound information.
ItemTemplate	Determines the output format of each row in the data source. This template is called for each item in the list and can contain data binding expressions.
AlternatingItemTemplate	Functions similarly to ItemTemplate. This template applies only to rows with an odd ordinal position.
SeparatorTemplate	Determines the HTML content that goes between each row. It cannot contain data bound information.
FooterTemplate	Rendered only once when all items have been rendered. It cannot contain data bound information.

When you call DataBind on the Repeater control, the control first attempts to locate the HeaderTemplate and, if found, applies it. The control then loops through the list items and applies the ItemTemplate to each one. When the AlternatingItemTemplate is defined, odd rows take it. If the SeparatorTemplate is found, the template is applied in between two consecutive items, regardless of whether the template item is normal or an alternate. When the FooterTemplate is specified, it is applied at the end of the loop.

The Repeater control has no built-in layout facility or predefined style that you can apply by using a declaration, nor does it support selection and editing. You could manually code selection and editing, but for those tasks, you are better off dropping the Repeater control in favor of the more powerful DataList and DataGrid controls.

Accessing Data Bound Information

The Items property of the Repeater control is a collection of RepeaterItem objects, each representing an individual row in the list. To define the template structure for normal and alternating items, you need a way to access the data source fields for the particular row being processed. This information is exactly what the expression Container.DataItem returns when you use Container.DataItem in a data-binding expression for the normal or alternate templates.

Container.DataItem represents the nth object bound to the RepeaterItem objects in a list. It evaluates to the same type of element as the caller placed in the Repeater control's data source. Let's look at some examples. If you place a DataTable object in the DataSource property of the Repeater control, then Container.DataItem evaluates to a DataRow object. It evaluates to

DataRow because in .NET, a DataTable is perceived as a collection of rows and each row is implemented through a DataRow object. If you use a DataView object, the individual item is evaluated to a DataRowView object. If you use, say, an array of strings, Container.DataItem evaluates to a simple text string. Table 1-3 shows how to obtain the value of a field for the current row.

Different Ways to Retrieve Data Item Values in Iterative Controls

Data Source	Data Item
Data Table	<%# ((DataRow) Container.DataItem)["Field"] %>
Data View	<%# ((DataRowView) Container.DataItem)["Field"] %>
Array of Strings	<%# ((String) Container.DataItem) %>
Dictionary	<%# ((Dictionary) Container.DataItem)["entry"] %>

Because .NET objects are strongly typed, coercing types is an absolute necessity. If you find the syntax in Table 1-3 somewhat quirky, you can alternatively resort to the DataBinder class. This class features a static method named Eval that has an easier syntax to remember than that of standard data-binding expressions. When you use DataBinder.Eval to get the current data item in the expressions in Table, the code looks like this:

```
<%# DataBinder.Eval(Container.DataItem, "Field") %>
```

As you can see, DataBinder.Eval blurs the distinction between the element types in the control's data source. However, because of automatic type inference and conversion, the housekeeping code for DataBinder.Eval can result in a slightly slower server response.

The DataBinder.Eval method has two prototypes:

```
public static Object Eval(Object container, String expr);
public static String Eval(Object container, String expr, String format);
```

The first argument identifies the data provider to which the next expression must be applied. The third argument, which is optional, specifies a format string that converts the results of evaluating the data-binding expression to String. When you need to apply formatting, DataBinder.Eval helps considerably to build a more readable statement. The following two instructions obtain the same result—formatting a numeric column as a currency value—but the first is easier to read and understand, albeit slightly slower:

```
<%# DataBinder.Eval(Container.DataItem, "Price", "{0:c}") %>
<%# String.Format("{0:c}", ((DataRowView) Container.DataItem)["Price"]) %>
```

> Using DataBinder.Eval does not give you more programming power. It just makes programming easier because it shields you from many of the casting and formatting details. In return for this, it adds a little more overhead to your code, so try to use it only when necessary, such as when you need formatting.

Repeater Control Events

In addition to firing the standard events supported by all .NET controls (such as Init, Load, PreRender, and DataBinding), the Repeater control can fire three of its own events, which are described in Table.

Event	Description
ItemCreated	Fires whenever a new item is created, regardless of its type (header, footer, item, or separator) or its ordinal position. `ItemCreated(Object s, RepeaterItemEventArgs e)` The event structure makes available to your code the current item (the Item property), the index (the ItemIndex property), the type (the ItemType property), and the DataItem associated with it (the DataItem property).
ItemCommand	Fires whenever a user clicks a link button or a push button embedded in the control's template. `ItemCommand(Object s, RepeaterItemEventArgs e)` The event structure provides the CommandSource property to let you know about the button that triggered the event.
ItemDataBound	Fires when an item is data bound, always before it is rendered. `ItemDataBound(Object s, RepeaterItemEventArgs e)`

The ItemCommand event interface deserves a little more explanation because it is a peculiarity of iterative control and, like the Repeater control, is widely used with the DataList and DataGrid controls. ItemCommand fires whenever the user clicks any button in the Repeater control. You retrieve the current instance of the clicked object by using the CommandSource property. You can give each button control a name that identifies its action. You retrieve this name by using the CommandName property of RepeaterItemEventArgs.

Figure 1-4 shows an HTML table that was generated by using the Repeater control. This figure displays a few fields selected from the Northwind Employees database.

When the user clicks the Search button on the screen in Figure 1-4, the page runs a query against SQL Server and stores the default view of the retrieved table as the data source of the Repeater control.

Repeater has the following structure:

```
<asp:repeater runat="server" id="Repeater1">
    <HeaderTemplate> ... </HeaderTemplate>
    <ItemTemplate> ... </ItemTemplate>
    <AlternatingItemTemplate> ... </AlternatingItemTemplate>
    <SeparatorTemplate> ... </SeparatorTemplate>
    <FooterTemplate> ... </FooterTemplate>
</asp:repeater>
```

For each template, you provide the ASP.NET code that will be evaluated dynamically during the execution of the DataBind method. You don't need to specify all the templates. However, for a reasonably complex control, you need to specify at least the header and the item. Let's take a look at each of the templates for the example in Figure 1-4.

The Header template is processed only once and is responsible for opening the table and defining the caption.

```
<table style="border:1px solid black;" class="stdtext">
   <thead bgcolor="blue" style="color:white">
      <td><b>ID</b></td>
      <td><b>First Name</b></td>
      <td><b>Last Name</b></td>
   </thead>
```

If you don’t define the same number of headers as expected columns, the graphical effect will be poor but you won’t get run-time errors.

Items and alternating items are element types that the control can represent with different graphical settings. Suppose your goal is to build an HTML table. As the following code illustrates, the item templates can add only the necessary number of rows and cells for the overall buffer.

```
<ItemTemplate>
   <tr>
      <td bgcolor="white"> <%# DataBinder.Eval(Container.DataItem,
         "EmployeeID") %> </td>
      <td bgcolor="white"> <%# DataBinder.Eval(Container.DataItem,
         "FirstName") %> </td>
      <td bgcolor="white"> <%# DataBinder.Eval(Container.DataItem,
         "LastName") %> </td>
   </tr>
</ItemTemplate>

<AlternatingItemTemplate>
   <tr>
      <td bgcolor="lightblue"> <%# DataBinder.Eval(Container.DataItem,
         "EmployeeID") %> </td>
      <td bgcolor="lightblue"> <%# DataBinder.Eval(Container.DataItem,
         "FirstName") %> </td>
      <td bgcolor="lightblue"> <%# DataBinder.Eval(Container.DataItem,
         "LastName") %> </td>
   </tr>
</AlternatingItemTemplate>
```

For any of the fields, an alternative approach would be to use more direct syntax and avoid DataBinder.Eval, as shown here:

```
<%# ((DataRowView) Container.DataItem)["LastName"] %>
```

Like the Header template, the Footer template is drawn only once and only when all items are rendered.

```
<FooterTemplate>
   <tfoot>
      <td bgcolor="silver" colspan="3">
         <%# "<b>" + ((DataView) Repeater1.DataSource).Count +
            "</b> employees found." %>
      </td>
   </tfoot>
</table>
</FooterTemplate>
```

The Footer template defines the final row by using a <tfoot> HTML tag and makes sure that the final row spans all columns. Typically, you use the footer to display summary information. To

obtain the total number of employees displayed, the code resorts to the Count property of the DataView class. To get the same information with a DataTable set as the data source, you would use the following expression:

```
<%# ((DataTable) Repeater1.DataSource).Rows.Count +
   " employees found." %>
```

If DataSource is set but no data is returned, the Repeater control attempts to render only the header and footer templates, if either is specified. No items and separators are rendered. If the DataSource property is not set, the Repeater control is not rendered.

The DataList Control

If in the previous example you absentmindedly replaced Repeater with DataList, your code would run perfectly and function in the same way. However, the two controls are not identical. DataList, which is a data bound control, offers you a full bag of new features, mostly in the area of graphical layout—in fact, its set of features begins where the set of features supported by the Repeater control ends. For example, it supports directional rendering in which items can flow horizontally or vertically, depending on the specified number of columns. DataList provides facilities to retrieve a key value associated with a current row in the data source and has built-in support for selection and editing. DataList also supports more templates and can fire more events than Repeater.

Basic data-binding operations that use DataList are similar to those that use Repeater. You use the DataSource property to bind the control to data and the DataBind method to refresh the user interface.

```
DataList1.DataSource = ds.Tables["EmpTable"];
DataList1.DataBind();
```

In template code, you can access data bound information using both the DataBinder object and the Container.DataItem expression.

Templates Specific to DataList

The DataList control supports two templates that we haven't yet discussed: SelectedItemTemplate and EditItemTemplate. SelectedItemTemplate controls how the selected item is displayed. You should use it only when you need to display other controls or apply a particular logic. When you want to change only the appearance of the selected item, resort to a simpler cascading style sheet (CSS) style set via the SelectedItemStyle property.

EditItemTemplate specifies the graphical template for the row currently being edited. If you don't specify your own template, the standard template is used, which renders any data bound element by using text boxes. Note that the template defines the layout of the control only when in edit mode. Managing the user interaction, saving the edit, or canceling the edit operation must be handled separately. For each template, you can use special style tags to set all the CSS attributes that you need via a declaration. These style tags are <AlternatingItemStyle>, <ItemStyle>, <HeaderStyle>, <FooterStyle>, <SeparatorStyle>, <SelectedItemStyle>, and <EditItemStyle>. The following code assigns a white background to the items and a thick black border:

```
<ItemStyle BorderColor="black" BorderWidth=3 BackColor="white" />
```


```
<ItemStyle BorderColor="black" BorderWidth=3 BackColor="white" />
```


You can also use style properties that have the same name as the style tags, as shown in this code:

```
DataList1.ItemStyle.BackColor = Color.White;
```

In ASP.NET pages, you indicate colors by using strings. The ASP.NET run time silently converts the color name to a valid .NET type. To assign a color programmatically, you must use the predefined values of the Color enumeration in the System.Drawing namespace. If you

don’t know the .NET name of a given color, use any of the following expression formats:

```
Color.FromName("white");
Color.FromName("#00ee00");
```

Events Specific to DataList

All iterative controls fire a generic ItemCommand event when a user clicks a link button or a push button, and this button is associated with a command name that is specified by the programmer. You use the command name within the ItemCommand event to recognize which button has actually been clicked. If these links and buttons are automatically generated during the execution of DataBind, it is especially difficult for you to assign a unique value to the OnClick event for each control. The next code example clarifies my point. The DataList control generates buttons based on the content of an array of strings.

```
<asp:DataList runat="server" id="DataList1"
    RepeatDirection="Horizontal"
    OnItemCommand="HandleItemCommand">

<ItemTemplate>
    <asp:linkbutton runat="server" CommandName="<%# Container.DataItem %>"
        Text="<%# Container.DataItem %>" />    
</ItemTemplate>

</asp:DataList>
```

The data source returned by the CreateDataSource function generates the output shown in Figure 1-5.

```
public ArrayList CreateDataSource()
{
    ArrayList a = new ArrayList();
    a.Add("Search");
    a.Add("Execute");
    a.Add("Load");
    return a;
}
```

Within the body of the HandleItemCommand function, you can analyze the command name and decide the proper action to take.

```
void HandleItemCommand(Object sender, DataListCommandEventArgs e)
{
    switch (e.CommandName)
    {
        case "Search":
            statusbar.Text = "searching...";
            return;
        case "Execute":
            statusbar.Text = "executing...";
            return;
        case "Load":
            statusbar.Text = "loading...";
            return;
    }
}
```

The full source for the CommandButtons.aspx application is available on the companion CD.

Special Command Names

The DataList control provides special support for five predefined command names: edit, update, delete, cancel, and select. Any link or button in a DataList control whose command name matches one of these five keywords receives special treatment from the control. The predefined events that fire when the user clicks buttons associated with these strings are listed in Table 1-5.

Table 1-5 Predefined Events of the DataList Control

Event	Description
CancelCommand	Fires when a button named Cancel is clicked.
DeleteCommand	Fires when a button named Delete is clicked.
EditCommand	Fires when a button named Edit is clicked. When a user clicks the Edit button, the item automatically enters edit mode. Edit mode implies use of the EditItem template.
UpdateCommand	Fires when a button named Update is clicked.
SelectedIndexChanged	Fires when a button named Select is clicked. Clicking a Select button automatically deselects the current item and selects the new item.

Edit mode and select mode are graphically rendered through their specific templates and CSS style.

Clicking a button named with a predefined keyword generates two events: first the generic ItemCommand event with the CommandName property set to the keyword and then the button-specific event (UpdateCommand).

Relating Graphical and Data Elements

To enable item selection and in-place editing, the DataList control has an internal mechanism that associates graphical items with corresponding elements in the data source. To retrieve this association, you set the DataKeyField property to the name of one data source field. The selected field must have content that allows you to uniquely identify the data item (a primary key). For example, if your data source is a table with information about a company's employees, you could assign the employeeid field to DataKeyField, as shown here:

```
<asp:DataList runat="server" DataKeyField="employeeid">
```

After DataKeyField is set, the control ensures that the DataKeys collection contains all the field's values for each item the control is currently displaying. You retrieve the key value based on the ordinal position of the item. For example, to obtain the key value of the currently selected item, you would do this:

```
int nEmployeeID = DataKeys[DataList1.SelectedIndex];
```

The number of displayed items does not necessarily match the number of items in the data source. In some situations, to handle a special application's requirements, you might need to programmatically hide some of the items. For example, you have to do this with DataGrid controls when pagination is enabled. (We'll talk more about DataGrid controls later in this chapter.)

The following code example illustrates how to take advantage of the DataList control to create a “smart” list box, and Figure 1-6 shows the results. (The full source code for the SmartList.aspx application is available on the companion CD.) All items in the data source are rendered as a link button with the command name select. As you know, the DataList control provides special support for select. When the user clicks any of the selectable links, ASP.NET fires a SelectedIndexChanged event and updates the style of the clicked item according to the attributes set through the SelectedItemStyle tag. The appearance of the button is refreshed to reflect the new state. In response to the selection event, additional information about the employee is retrieved and displayed. A new button appears at the bottom of the page to let the user deselect any selected item.

When the page first loads, it retrieves only the fields needed to prepare the list of employees—that is, employeeid, firstname, and lastname. When the user clicks a link button, the DataList control is re-bound and an informative label appears with a message.

```
public void OnSearch(Object sender, EventArgs e)
{
    DataSet ds = new DataSet();

    String sConn = "server=localhost;uid=sa;Initial Catalog=Northwind;";
    String sText = "SELECT employeeid, firstname, lastname FROM Employees";
    sText += " WHERE employeeid >=" + tbEmpID.Text;
    SqlDataAdapter cmd = new SqlDataAdapter(sText, sConn);
    cmd.Fill(ds, "EmpTable");

    DataList1.DataSource = ds.Tables["EmpTable"].DefaultView;
    DataList1.DataBind();

    theLabel.Visible = true;
    theLabel.Text = "Click to read more.";
}
```

The DataList control has the following structure:

```
<asp:DataList runat="server" id="DataList1"
    DataKeyField="employeeid"
    OnSelectedIndexChanged="HandleSelection">

<SelectedItemStyle BackColor="lightblue" />

<HeaderTemplate>
    <h3>Northwind's Employees</h3>
</HeaderTemplate>

<ItemTemplate>
    <asp:linkbutton runat="server" commandname="select"
        Text='<%# ((DataRowView)Container.DataItem)["employeeid"] + " - " +
             ((DataRowView)Container.DataItem)["lastname"]  + ", " +
             ((DataRowView)Container.DataItem)["firstname"] %> ' />
</ItemTemplate>

<FooterTemplate> <hr> </FooterTemplate>
</asp:DataList>
```

To retrieve additional information about the clicked item, you need to issue another query based on a key value. You retrieve the key value associated with the selected item by using the DataKeys array.

```
void HandleSelection(Object sender, EventArgs e)
{
```

```
    int nEmpID = (int) DataList1.DataKeys[DataList1.SelectedIndex];

    String sConn = "server=localhost;uid=sa;Initial Catalog=Northwind;";
    SqlConnection cn = new SqlConnection(sConn);

    String sText = "SELECT title, hiredate, country, notes FROM Employees";
    sText += " WHERE employeeid = " + nEmpID.ToString();
    SqlCommand cmd = new SqlCommand(sText, cn);
    cn.Open();

    SqlDataReader dr = cmd.ExecuteReader();
    dr.Read();

    theLabel.Text =  "<b>" + dr["title"] + "</b><br>";
    DateTime dtime = Convert.ToDateTime(dr["hiredate"]);
    theLabel.Text += "Hired on " + dtime.ToShortDateString() + " from " +
                     dr["country"] + "<hr>";
    theLabel.Text += "<i>" + dr["Notes"] + "</i>";

    btnUnselect.Visible = true;
    dr.Close();
    cn.Close();
}
```

Using the SqlDataReader control instead of the DataSet control to retrieve data is more efficient when you have just one row to fetch. After the data has been successfully read, it gets properly formatted and the Unselect button is displayed.

The user interface of the application changes significantly when an employee name is selected. Additional information about the employee is retrieved and displayed, and a new button appears to let you deselect the currently selected item.

To enable the user to deselect an item, set the SelectedIndex property to -1.

```
void RemoveSelection(Object sender, EventArgs e)
{
    DataList1.SelectedIndex = -1;
    theLabel.Text = "Click to read more.";
    btnUnselect.Visible = false;
}
```

The DataList control and the panel that displays additional employee information are two cells of the same all-encompassing table. By using different settings for the layout properties of DataList, you can obtain a significantly altered design for the links without affecting the core of the code.

As the preceding code example illustrates, the DataList control is much more powerful than the Repeater control, but it is far from being perfect. It still lacks pagination, sorting, and powerful column-based rendering capabilities.

The DataGrid Control

The DataGrid control renders a multi-column, fully templated grid and is by far the most versatile of all data bound controls. The user interface it provides closely resembles a Microsoft Excel worksheet. Despite its rather advanced programming interface and full set of attributes, DataGrid simply generates an HTML table with interspersed links to provide interactive functionality such as sorting and pagination commands.

With the DataGrid control, you can create simple data bound columns that show data retrieved from a data source, template columns that let you design the layout of cell contents, and last but not least, command-based columns that allow you to add specific functionality to a grid. In the next chapters, I'll delve more deeply into the implementation and the customizable features of the DataGrid control. But I think a discussion of the features that differentiate DataGrid from DataList is useful here.

The DataGrid control uses templates extensively but differently from the DataList and Repeater controls. DataGrid renders tables of data organized in columns, so the templates don't apply to the control as a whole but rather to specific columns. DataList and Repeater work on a per-item basis and make you responsible for controlling the final layout of the data. DataGrid has just one possible layout—a sequence of columns—and supports the same events as the DataList control plus two more: PageIndexChanged and SortCommand.

Now let's take a look at the DataGrid control in action. As you've learned, the DataGrid control works by displaying columns of data. In most cases, you need to specify which columns you want and how you want them displayed, but when the data to be rendered is uncomplicated, you can leave the control free to automatically generate the columns based on the structure of the data source. When you leave the control in charge, however, you cannot control the heading of each column, which defaults to the column name. The following code arranges a DataGrid control that uses alternating rows and automatically generates the columns.

```
<asp:DataGrid id="grid" runat="server"
    AutoGenerateColumns="true"
    CellPadding="2" CellSpacing="2" GridLines="none"
    BorderStyle="solid" BorderColor="black" BorderWidth="1"
    font-size="x-small" font-names="verdana">

    <AlternatingItemStyle BackColor="palegoldenrod" />
    <ItemStyle BackColor="beige" />
    <HeaderStyle ForeColor="white" BackColor="brown" Font-Bold="true" />
</asp:DataGrid>
```

Notice that you can use the style properties to declare CSS styles.

The next code example illustrates how to fill the grid and refresh the user interface.

```
void OnLoadData(Object sender, EventArgs e)
{
    SqlConnection conn = new SqlConnection(txtConn.Text);
    SqlDataAdapter da = new SqlDataAdapter(txtCommand.Text, conn);
    DataSet ds = new DataSet();
    da.Fill(ds, "MyTable");

    grid.DataSource = ds.Tables["MyTable"];
    grid.DataBind();
}
```