Notes sourced from 'ASP.NET 4 Unleashed' by Steven Walther, et al, 2010

Understanding ASP.NET Controls

ASP.NET controls are the heart of the ASP.NET Framework. An ASP.NET control is a .NET class that executes on the server and renders certain content to the browser.

The ASP.NET framework includes over 70 controls, which enable you to do everything from displaying a list of database records to displaying a randomly rotating banner advertisement.

In this session, you are provided with an overview of the controls included in the ASP.NET Framework. You also learn how to handle events that are raised by controls and how to take advantage of View State.

Overview of ASP.NET Controls

The ASP.NET Framework (version 4.0) 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.
- **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. These controls are discussed in *Session 2*.
- **Rich Controls** The rich controls enable you to render things such as calendars, file upload buttons, rotating banner advertisements, and multi-step wizards. These controls are discussed in *Session 3*.
- **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. These controls are discussed in detail in later.
- **Navigation Controls** The navigation controls enable you to display standard navigation elements such as menus, tree views, and bread crumb trails. These controls are discussed later
- **Login Controls** The login controls enable you to display login, change password, and registration forms. These controls are discussed in later.
- **Web Part Controls** The Web Part controls enable you to build personalizable portal applications.

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.

Note

You should always assign an ID attribute to every control even when you don't need to program against it. If you don't provide an ID attribute, then certain features of the ASP.NET Framework (such as two-way databinding) won't work.

The declaration also includes a `runat="Server"` attribute. This attribute marks the tag as representing a server-side control. If you neglect to include this attribute, then the `TextBox` tag would be passed, without being executed, to the browser. The browser would simply ignore the tag.

Finally, notice that the tag ends with a forward slash. The forward slash is shorthand for creating a closing `</asp:TextBox>` tag. You can, if you prefer, declare the `TextBox` control like this:

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

In this case, the opening tag does not contain a forward slash and an explicit closing tag is included.

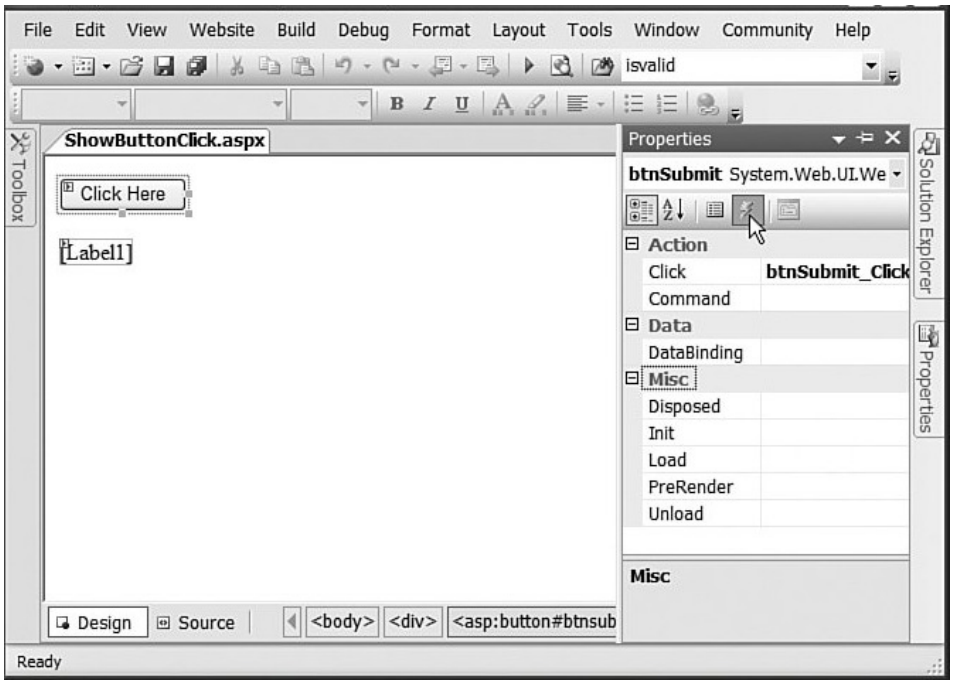
Understanding and Handling Control Events

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.

You can add an event handler automatically to a control in multiple ways when using VisualStudio 2012. In Source view, add a handler by selecting a control from the top-left drop-down list and selecting an event from the top-right drop-down list

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 (see below).



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.

Understanding 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 ASP.NET Framework uses a trick called View State. If you open an aspx page with some controls added on it, and view this page in a browser and select View Source, you'll notice that the page includes a hidden form field named `__VIEWSTATE` that looks something 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.

Sometimes, you might want to disable View State even when you aren't concerned with the size of the `__VIEWSTATE` hidden form field. For example, if you are using a Label control to display a form validation error message, you might want to start from scratch each time the page is submitted. In that case, simply disable View State for the Label control.

Note
The ASP.NET Framework since version 2.0 includes a new feature called Control State. Control State is similar to View State except that it is used to preserve only critical state information. For example, the GridView control uses Control State to store the selected row. Even if you disable View State, the GridView control remembers which row is selected.

Handling Page Events

Whenever you request an ASP.NET page, a particular set of events is raised in a particular sequence. This sequence of events is called the page execution lifecycle.

For example, we have already seen the `Page Load` event demonstrated by the lecturer in class. You normally use the `Page Load` event to initialize the properties of controls contained in a page. However, the `Page Load` event is only one event supported by the `Page` class.

Here is the sequence of events that are raised whenever you request a page:

1. `PreInit`
2. `Init`
3. `InitComplete`
4. `PreLoad`
5. `Load`
6. `LoadComplete`
7. `PreRender`
8. `PreRenderComplete`
9. `SaveStateComplete`
10. `Unload`

Why so many events? Different things happen and different information is available at different stages in the page execution lifecycle.

For example, View State is not loaded until after the `InitComplete` event. Data posted to the server from a form control, such as a `TextBox` control, is also not available until after this event.

Ninety-nine percent of the time, you won't handle any of these events except for the `Load` and the `PreRender` events. The difference between these two events is that the `Load` event happens before any control events and the `PreRender` event happens after any control events.

The other thing you should notice about an aspx page is the way the event handlers are wired to the `Page` events. ASP.NET pages support a feature named `AutoEventWireUp`, which is enabled by default. If you name a subroutine `Page_Load()`, the subroutine automatically handles the `Page Load` event; if you name a subroutine `Page_PreRender()`, the subroutine automatically handles the `Page PreRender` event, and so on.

Warning- *`AutoEventWireUp` does not work for every page event. For example, it does not work for the `Page_InitComplete()` event.*

Using the `Page.IsPostBack` Property

The `Page` class includes a property called the `IsPostBack` property, which you can use to detect whether the page has already been posted back to the server.

Because of View State, when you initialize a control property, you do not want to initialize the property every time a page loads. Because View State saves the state of control properties across page posts, you typically initialize a control property only once, when the page first loads.

In fact, many controls don't work correctly if you re-initialize the properties of the control with each page load. In these cases, you must use the `IsPostBack` property to detect whether or not the page has been posted.

Displaying Information

The ASP.NET Framework includes two controls you can use to display text in a page: the `Label` control and the `Literal` control. Whereas the `Literal` control simply displays text, the `Label` control supports several additional formatting properties.

Using 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.

By default, a `Label` control renders its contents in an HTML `<span>` tag. Whatever value you assign to the `Text` property is rendered to the browser enclosed in a `<span>` tag.

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.)

In general, I recommend that you avoid using the formatting properties and take advantage of Cascading Style Sheets to format the rendered output of the `Label` control. More about this later.

When you provide a `Label` control with an `AssociatedControlID` property, the `Label` control is rendered as an HTML `<label>` tag instead of an HTML `<span>` tag.

Always use a `Label` control with an `AssociatedControlID` property when labeling form fields. This is important when you need to make your website accessible to persons with disabilities.

If someone is using an assistive device, such as a screen reader, to interact with your website, the `AssociatedControlID` property enables the assistive device to associate the correct label with the correct form field.

A side benefit of using the `AssociatedControlID` property is that clicking a label when this property is set automatically changes the form focus to the associated form input field.

Web Standards Note

Both the WCAG 1.0 and Section 508 accessibility guidelines require you to use the `<label for>` tag when labeling form fields. For more information, see http://www.w3.org/wai and http://www.Section508.gov.

Using the `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. However, unlike the `Label` control, the `Literal` control does not render its content inside of a `<span>` tag.

Because the contents of a `Literal` control are not contained in a `<span>` tag, the `Literal` control does not support any of the formatting properties supported by the `<span>` tag. For example, the `Literal` control does not support either the `CssClass` or `BackColor` properties.

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.

Accepting User Input

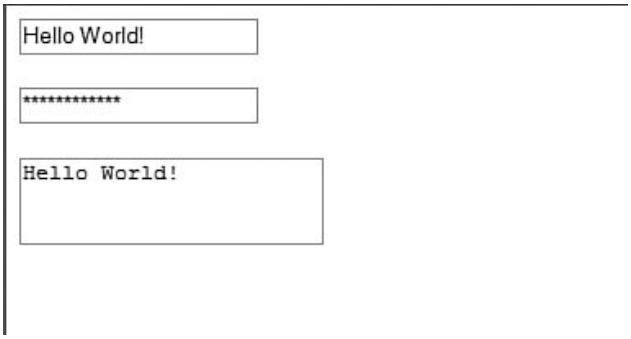
The ASP.NET Framework includes several controls that you can use to gather user input. In this section, you learn how to use the `TextBox`, `CheckBox`, and `RadioButton` controls. These controls correspond to the standard types of HTML input tags.

Using the `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.

The page below contains three `TextBox` controls that illustrate all three of the `TextMode` values.



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.

And, the `TextBox` control supports the following event:

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

When the `AutoPostBack` property has the value `true`, the form containing the `TextBox` is automatically posted back to the server when the contents of the `TextBox` changes

Web Standards Note

You should avoid using the `AutoPostBack` property for accessibility reasons. Creating a page that automatically reposts to the server can be very confusing to someone using an assistive device such as a screen reader. If you insist on using the `AutoPostBack` property, you should include a value for the `ToolTip` property that warns the user that the page will be reloaded.

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 numberin 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).

Note

When using Internet Explorer, you can configure `AutoComplete` by selecting Tools, Internet Options, Content, and clicking the AutoComplete button. The ASP.NET Framework does not support `AutoComplete` for other browsers such as FireFox or Opera.

Finally, the `TextBox` control supports the `Focus()` method. You can use the `Focus()` method to shift the initial form focus to a particular `TextBox` control. By default, no form field has focus when a page first opens. If you want to make it easier for users to complete a form, you can set the focus automatically to a particular `TextBox` control contained in a form.

Using the `CheckBox` Control

The `CheckBox` control enables you to display, well, a check box.

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.

And, the `CheckBox` control supports the following event:

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

Notice that the `CheckBox` control, like the `TextBox` control, supports the `AutoPostBack` property.

Note
Framework also includes the CheckBoxList control that enables you to display a list of check boxes automatically.

Using the `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.

Note
The ASP.NET Framework also includes the RadioButtonList control, which enables you to display a list of radio buttons automatically.

Submitting Form Data

The ASP.NET Framework includes three controls you can use to submit a form to the server: the `Button`, `LinkButton`, and `ImageButton` controls. These controls have the same function, but each control has a distinct appearance.

In this section, you learn how to use each of these three types of buttons in a page. Next, you learn how to associate client-side scripts with server-side `Button` controls. You also learn how to use a button control to post a form to a page other than the current page. Finally, you learn how to handle a button control's Command event.

Using the `Button` Control

The `Button` control renders a push button that you can use to submit a form 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.

Using the `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.

Using the `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

Web Standards Note

Always include alternate text for any image. The accessibility guidelines require it. Furthermore, remember that some people turn off images in their browsers for a faster surfing experience.

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.

Web Standards Note

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 `ImageMap` control is discussed in the next section of this chapter.

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.
-

Specifying a Default Button

You can specify a default button for a form by using the `DefaultButton` property of the server-side `Form` control. If you specify a default button, then pressing the keyboard Enter key invokes the button.

Displaying Images

The ASP.NET framework includes two controls for displaying images: the `Image` and `ImageMap` controls. The `Image` control simply displays an image. The `ImageMap` control enables you to create a client-side, clickable, image map.

Using the `Image` Control

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

- `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 complexsuch as a bar chart, pie graph, or company organizational chartthen 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.

Using the `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.

More about ImageMaps and Hotspots later.