

```
string country = txtCountry.Text.ToLower();
if (country == "canada")
    labRegion.Text = "Province";
else if (country == "united states" || country == "usa")
    labRegion.Text = "State";
else
    labRegion.Text = "Region";
}
```

The result of this listing is shown in Figure 3.5.

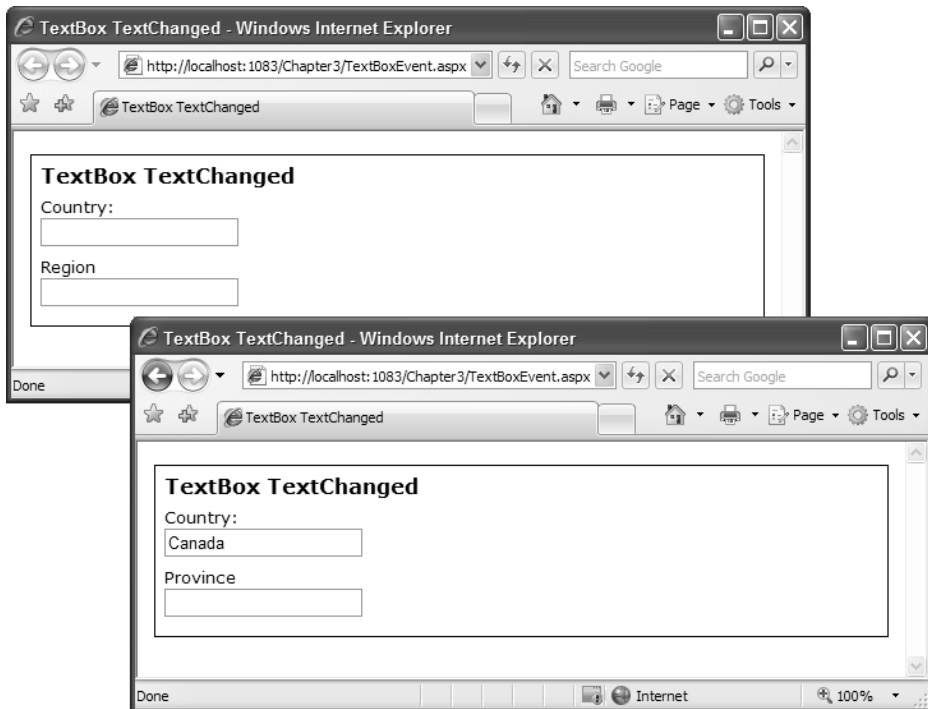


Figure 3.5 TextBoxEvent.aspx

Button-Style Controls

There are three kinds of button controls in ASP.NET.

- Button—Displays a standard HTML submit button.

- `LinkButton`—Displays a hyperlink that works as a button. Unlike a regular HTML `<a>` element or the `HyperLink` control, the `LinkButton` control causes a postback (and thus allows for post-click processing on the server).
- `ImageButton`—Displays an image that works as a button and that allows you to determine the image coordinates of the location that was clicked.

All three button controls submit the form to the server when clicked; that is, they cause a postback to the server when clicked.

Button controls can be either *submit* or *command* buttons. By default, buttons are submit buttons. This means that they cause a postback to the server when pushed; this event can then be handled on the server by writing a `Click` event handler. Command buttons also cause a postback when pushed. However, they also have a command name associated with them. This allows you to use a single event handler (a `Command` event handler) for multiple buttons on a form. Although the three buttons do not share a common base class (`Button` and `LinkButton` inherit from `WebControl`, whereas `ImageButton` inherits from `Image`), they nonetheless share several unique properties and events, as shown in Tables 3.8 and 3.9.

Table 3.8 Unique Properties of the `Button`, `ImageButton`, and `LinkButton` Classes

Property	Description
<code>CausesValidation</code>	Indicates whether validation is performed when button is clicked. Validation is covered in Chapter 5.
<code>CommandArgument</code>	Specifies optional argument to be used in conjunction with the <code>CommandName</code> property.
<code>CommandName</code>	Specifies the command name for the button.
<code>OnClickClick</code>	Specifies the client-side script to be run when the button's click event is triggered.
<code>PostBackUrl</code>	Specifies the URL that the page posts back to when the button is clicked. Normally, a page posts back to the same page as the button. This property allows you to post back to a different page.
<code>Text</code>	The text caption in the button (<code>Button</code> and <code>LinkButton</code> only).
<code>UseSubmitBehavior</code>	Indicates whether the control uses the client browser's submit mechanism or the ASP.NET postback mechanism (<code>Button</code> only). Default is <code>true</code> (use the browser's submit mechanism). If set to <code>false</code> , client-side script is added to the page that posts the form to the server.

Table 3.9 Unique Events of the Button, ImageButton, and LinkButton Classes

Event	Description
Click	Raised when the button is clicked. Used to define a handler for a submit-style button.
Command	Raised when the button is clicked. Used to define a handler for a command-style button.

Listings 3.6 and 3.7 illustrate the use of the three different button controls. The result in the browser is shown in Figure 3.6.

Listing 3.6 ButtonTest.aspx

```

<p>
  <asp:ImageButton ID="imgbtnTest" runat="server"
    ImageUrl="images/navigation.gif"
    AlternateText="Navigation Menu"
    OnClick="imgbtnTest_Click" />
  <br />
  <asp:Label ID="labMessage1" runat="server"></asp:Label>
</p>
<p>
  <asp:Button ID="btnTest" runat="server" Text="Click Me"
    OnClick="btnTest_Click" />
  <br />
  <asp:Label ID="labMessage2" runat="server"></asp:Label>
</p>
<p>
  <asp:LinkButton ID="lnkbtnTest" runat="server"
    OnClick="lnkbtnTest_Click">
    Link to click
  </asp:LinkButton>
  <br />
  <asp:Label ID="labMessage3" runat="server"></asp:Label>
</p>

```

Listing 3.7 ButtonTest.aspx.cs

```

/// <summary>
/// Handler for the image button
/// </summary>
protected void imgbtnTest_Click(object sender,
    ImageClickEventArgs e)

```

```
{
    labMessage1.Text = " ImageButton Clicked Coordinates: " +
        e.X.ToString();
    labMessage1.Text += ", " + e.Y.ToString();
}

///summary
///Handler for the (regular) button
///summary
protected void btnTest_Click(object sender, EventArgs e)
{
    labMessage2.Text = "Button was clicked";
}

///summary
///Handler for the link button
///summary
protected void lnkbtnTest_Click(object sender, EventArgs e)
{
    labMessage3.Text = "LinkButton was clicked";
}
```

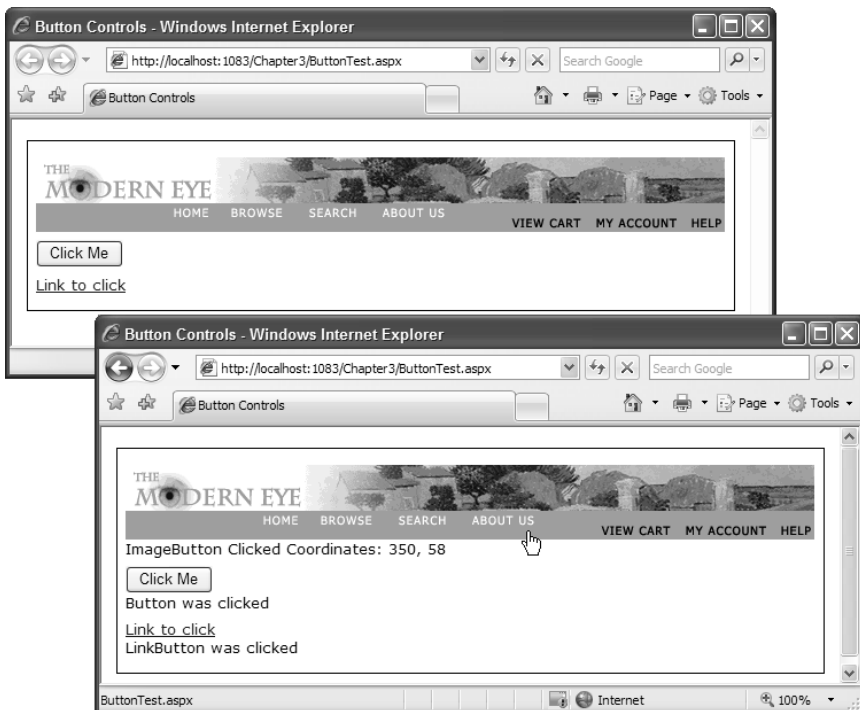


Figure 3.6 ButtonTest.aspx

Command Event

It is not uncommon for a page to contain multiple buttons that are similar in some way. An example is multiple link buttons in a list—for instance, Add to Cart buttons in a list of products. In such a situation, you do not want separate event handlers for each button, but a single event handler that handles the `Click` event for all the Add to Cart buttons.

The issue with using a single event handler for multiple buttons is that the event handler usually requires some way of determining which particular button triggered the event. The solution to this problem is the `Command` event.

When a button is clicked, the form is submitted to the server and both the `Click` and `Command` events are raised (`Click` is raised first). The only difference between the two events is that additional information (such as information that can be used to identify which button triggered the event) can be passed to the `Command` event via the `CommandName` and the `CommandArgument` attributes/properties.

Listings 3.8 and 3.9 use the `Command` event in conjunction with the `CommandName` attribute to implement a simple calculator that uses one event handler for four different buttons. The result in the browser can be seen in Figure 3.7.

Listing 3.8 `CommandTest.aspx`

```
<h1>Simple Calculator</h1>
<div class="box">
  <asp:TextBox ID="txtValue1" runat="server" />
  <asp:Button runat="server" ID="btnAdd"
    Width="30px" Text="+"
    OnCommand="Button_Command" CommandName="add" />
  <asp:Button runat="server" ID="btnSubtract"
    Width="30px" Text="-"
    OnCommand="Button_Command" CommandName="subtract" />
  <br />
  <asp:TextBox ID="txtValue2" runat="server" />
  <asp:Button runat="server" ID="btnMultiply"
    Width="30px" Text="*"
    OnCommand="Button_Command" CommandName="multiply" />
  <asp:Button runat="server" ID="btnDivide"
    Width="30px" Text="/"
    OnCommand="Button_Command" CommandName="divide" />
</div>
<p><asp:Label ID="labMessage" runat="server" /></p>
```

Listing 3.9 CommandTest.aspx.cs

```
/// <summary>
/// Single Command event handler for all the calculator buttons
/// </summary>
protected void Button_Command(Object o, CommandEventArgs e)
{
    double dVal1 = 0.0;
    double dVal2 = 0.0;

    // Try converting user input into double values
    bool val1okay = Double.TryParse(txtValue1.Text, out dVal1);
    bool val2okay = Double.TryParse(txtValue2.Text, out dVal2);

    if (val1okay && val2okay)
    {
        double result = 0;

        string op = "";
        switch (e.CommandName)
        {
            case "add":
                op = "+";
                result = dVal1 + dVal2;
                break;
            case "subtract":
                op = "-";
                result = dVal1 - dVal2;
                break;
            case "multiply":
                op = "*";
                result = dVal1 * dVal2;
                break;
            case "divide":
                op = "/";
                result = dVal1 / dVal2;
                break;
        }
        labMessage.Text = txtValue1.Text + op + txtValue2.Text
            + "=" + result;
    }
    else
    {
        labMessage.Text =
            "Unable to compute a value with these values";
    }
}
```

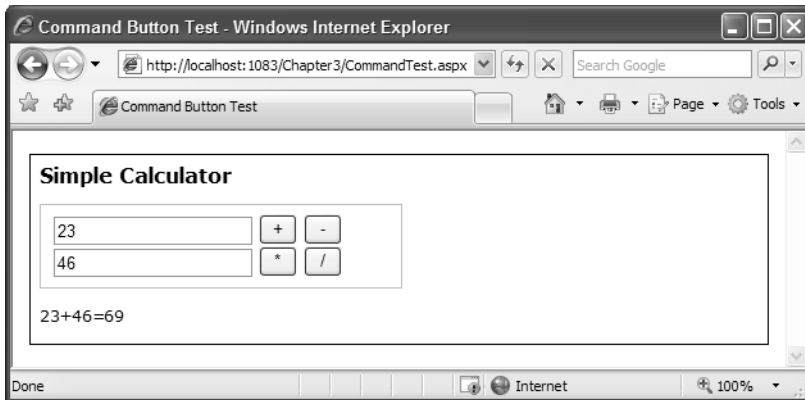


Figure 3.7 CommandTest.aspx

CORE NOTE

Notice the use of the `Double.TryParse` method. This method is used to convert the contents of the `TextBox` controls into `double` values. The `out` keyword indicates that the variable is being passed by reference. This means that the `TryParse` method returns the converted `double` value within the passed variable.



Working with Client-Side Events

There may be times when you want to associate client-side script events with your buttons. For instance, you want to display a client-side alert box when a button is clicked, or change the CSS or image of a button in response to the mouse's movement. ASP.NET allows you to add client-side attributes to any server control; if the control is not recognized by ASP.NET, it is simply passed to the client. For instance, in the following `ImageButton` control, the `OnMouseOver` attribute is not defined for the `ImageButton` control in ASP.NET.

```
<asp:ImageButton ... OnMouseOver="this.src='images/hot.gif'" />
```

Because the attribute is not recognized by ASP.NET, it is simply passed to the client and rendered as an attribute in the destination element. In this case, the resulting code sent to the browser is

```
<input type="image" ... OnMouseOver="this.src='images/hot.gif'" >
```

What do you do if the client attribute you want to work with has the same name as a supported property in ASP.NET? For instance, what if you want to display a Javascript alert box in response to a button click? Because both the server-side and the client-side event share the same name (`OnClick`), you cannot pass an attribute such as `OnClick="alert('message')"` to the client. The solution to this problem has typically required programmatically adding client-code to the attributes collection, such as

```
myButton.Attributes.Add("onclick",  
    "alert('Posting information...')");
```

However, with ASP.NET 2.0, there is a nonprogramming solution to the problem of passing code to the client for the `OnClick` event of buttons. The different ASP.NET button controls now have an `OnClientClick` property that allows you to attach Javascript code, which is run when the client event is triggered on the client. You can thus use this property instead of the `Attributes` collection, as in

```
<asp:ImageButton ... OnClientClick="alert(  
    'Posting Information...') " />
```

Listing 3.10 illustrates the use of some client-side events in conjunction with a `Button` control. In the listing, the control uses both an `OnClientClick` as well as two client-side attributes (`OnMouseOver` and `OnMouseOut`) to switch the CSS class of the button when the user moves the mouse over the button. The result in the browser can be seen in Figure 3.8.

Listing 3.10 ClientEvents.aspx

```
<head runat="server">  
  <title>Client Events</title>  
  <link href="chapterStyles.css" type="text/css"  
    rel="stylesheet" />  
  <style type="text/css">  
    .normal {  
      border: 3px double #999999;  
      padding: 0.25em;  
      background-color: aliceblue;  
      color: DimGray;  
      font: bold 12px Verdana,Helvetica,Arial,sans-serif;  
    }  
    .over {  
      border: 3px double DarkRed;  
      padding: 0.25em;  
      background-color: lightcyan;  
      color: DarkRed;  
      font: bold 12px Verdana,Helvetica,Arial,sans-serif;  
    }  
  </style>  
</head>
```

```
</style>
</head>
<body>
  <form id="form1" runat="server">
    <div id="container">
      <h1>Using client-side events</h1>
      <asp:Button ID="btnTest" runat="server"
        Text="Click Me" CssClass="normal"
        OnClientClick="alert('posting to server, please wait');"
        OnMouseOver="this.className='over';"
        OnMouseOut="this.className='normal';" />
    </div>
  </form>
</body>
```

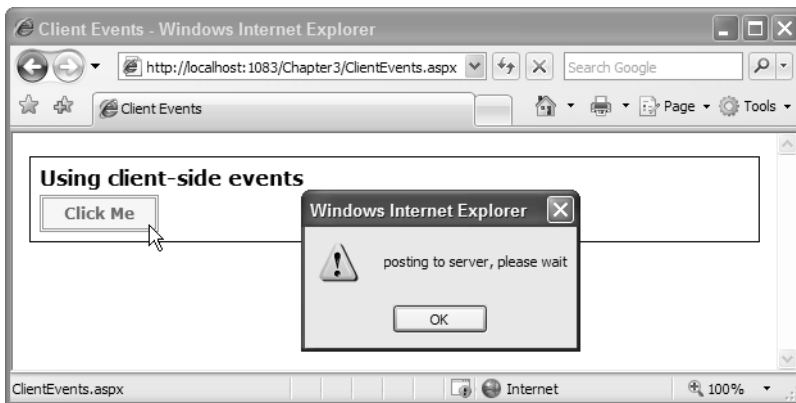


Figure 3.8 ClientEvents.aspx

CheckBox Control

The `CheckBox` control allows a user to choose between a true or false state. It is rendered on the browser as a check box form element. The `CheckBox` is usually just a single element within a form; it can, however, be used to cause a postback to the server. The unique properties and events of the check box control are shown in Tables 3.10 and 3.11.

Table 3.10 Unique Properties of the CheckBox Control

Property	Description
CausesValidation	Indicates whether validation is performed (true/false) when button is clicked. Validation is covered in Chapter 5.
Checked	Indicates whether the check box is checked (true).
InputAttributes	Collection of attributes to be rendered by (passed to) the client for the check box part of the control.
LabelAttributes	Collection of attributes to be rendered by (passed to) the client for the label part of the control.
Text	Specifies the text of the label part of the control.
TextAlign	Specifies the alignment (right or left) of the text label in relation to the check box itself.
ValidationGroup	Specifies the name of the validation group for which the control causes validation.

Table 3.11 Unique Events of the CheckBox Class

Event	Description
CheckedChanged	Raised when the check box is checked or unchecked. Note that this event is only raised with a postback.

Listings 3.11 and 3.12 illustrate the usage of the CheckBox control. Notice that Listing 3.11 uses the CheckBox control both as a static element within the form as well as a control that causes a postback. In the later usage, the control's event handler (shown in Listing 3.12) toggles the visibility of other controls on the form (see Figure 3.9).

Listing 3.11 CheckBoxTest.aspx

```

<h1>Check Box Test</h1>
<div class="box">
  <p>Delivery:
  <asp:CheckBox ID="chkDelivery" runat="server"
    OnCheckedChanged="CheckChanged" AutoPostBack="True" />
  </p>
  <p>
    <asp:Label ID="labAddress" runat="server"
      Text="Customer Address: " Visible="false" /><br />

```

```

        <asp:TextBox ID="txtAddress" runat="server"
            Columns="60" Visible="False" />
    </p>
    <p>
        Pizza Styles: <br />
        <asp:CheckBox ID="chkThin" runat="server"
            Text="Thin Crust" />
        <br />
        <asp:CheckBox ID="chkExtra" runat="server"
            Text="Extra Sauce" />
    </p>
</div>

<asp:Button ID="btnOrder" runat="server"
    Text="Order Pizza" OnClick="btnOrder_Click" />
<p><strong><asp:Label ID="labMessage" runat="server" />
</strong></p>

```

Listing 3.12 CheckBoxTest.aspx.cs

```

/// <summary>
/// Called when user changes the delivery check box
/// </summary>
protected void CheckChanged(object sender, System.EventArgs e)
{
    if (chkDelivery.Checked)
    {
        txtAddress.Visible = true;
        labAddress.Visible = true;
    }
    else
    {
        txtAddress.Visible = false;
        labAddress.Visible = false;
    }
}

/// <summary>
/// Called when user clicks order button
/// </summary>
protected void btnOrder_Click(object sender, EventArgs e)
{
    labMessage.Text = "Pizza Order Styles: <br/>";
    if (chkThin.Checked)
        labMessage.Text += chkThin.Text + "<br/>";
    if (chkExtra.Checked)
        labMessage.Text += chkExtra.Text + "<br/>";
}

```

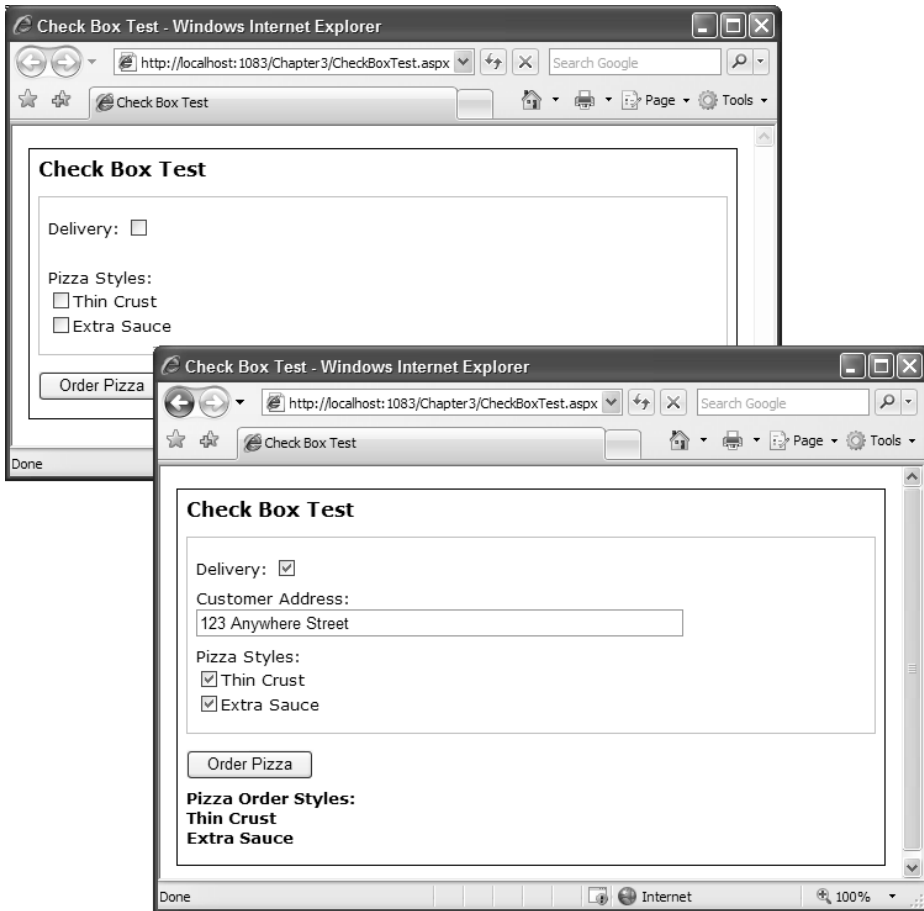


Figure 3.9 CheckBoxTest.aspx

RadioButton Control

The `RadioButton` control represents a radio button in a form. `RadioButton` controls can be logically grouped together using the `GroupName` property. Alternately, multiple radio buttons can be grouped together by using the `RadioButtonList` (covered separately in the next section).

Programming the `RadioButton` control is almost the same as programming the `CheckBox` control, because `RadioButton` is a subclass of the `CheckBox` class. The only property that is unique to the `RadioButton` control is the `GroupName` property, which is used to logically group separate radio button controls so that they form a set

of mutually exclusive buttons (that is, only one radio button in the group can be selected), as in

```
Select a philosopher: <br/>
<asp:RadioButton ID="radPhil1" Runat="server"
  GroupName="phil" Text="Aristotle" />
<asp:RadioButton ID="radPhil2" Runat="server"
  GroupName="phil" Text="Plato"/>
```

List-Style Controls

The `DropDownList`, `ListBox`, `CheckBoxList`, `RadioButtonList`, and `BulletedList` controls are list controls with fundamentally similar functionality. All of these classes share the same base class: the `ListControl` class. The `ListControl` class in turn inherits from the `DataBoundControl` class, as shown in Figure 3.10. (Data binding is covered in Chapter 8.)

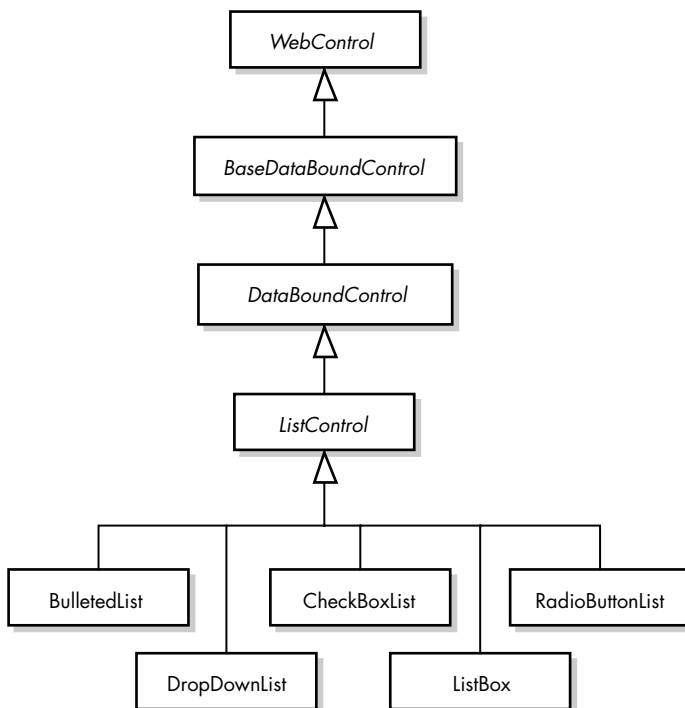


Figure 3.10 List controls class hierarchy

The unique properties and events of the `ListControl` class are shown in Tables 3.12 and 3.13.

Table 3.12 Unique Properties of the `ListControl` Class

Property	Description
<code>AppendDataBoundItems</code>	Specifies whether the list is cleared of existing items before data binding. The default is <code>false</code> , which means the list is cleared; setting this value to <code>true</code> preserves existing list values and appends the data bound items to the end of the list.
<code>DataTextField</code>	Specifies the field name of the data source that will provide the textual content for the list.
<code>DataTextFormatString</code>	Specifies the formatting string (see Chapter 10, page 585) that controls the visual display of the list content.
<code>DataValueField</code>	Specifies the field name of the data source that will provide the value for each list item.
<code>Items</code>	The collection of <code>ListItems</code> in the control.
<code>SelectedIndex</code>	The index (starting with 0) of the selected list item(s). For multiple selections, this property returns the lowest ordinal index that was selected. A value of -1 indicates that no item was selected in the list.
<code>SelectedItem</code>	The list item that was selected. For multiple selections, this property returns the item with the lowest index in the control.
<code>SelectedValue</code>	The value of the item that was selected.

Table 3.13 Unique Events of the `ListControl` Class

Event	Description
<code>SelectedIndexChanged</code>	Raised when the selection of the list control changes. Note that this event is only raised with a postback.

Notice that the `ListControl` class has a collection of `ListItems` controls. A `ListItems` encapsulates both the text and value of an item in a list. Table 3.14 lists the principle properties of the `ListItems`.

Table 3.14 Properties of the ListItem Class

Property	Description
Attributes	Specifies a collection of attributes to be rendered in browser. Note: These attributes are not in fact rendered in the browser.
Enabled	Indicates whether the list item is enabled. Disabled list items are not visible.
Selected	Indicates whether a list item is selected.
Text	The display text of the list item.
Value	The value of the list item.

You add `ListItem` objects to any of the list controls either declaratively or programmatically. For instance, the following adds list items to a `DropDownList` control declaratively.

```
<asp:DropDownList ID="myDrop" runat="server">
  <asp:ListItem Selected="True">Choose a color</asp:ListItem>
  <asp:ListItem>Red</asp:ListItem>
  <asp:ListItem>Blue</asp:ListItem>
  <asp:ListItem Value="#00FF00">Green</asp:ListItem>
</asp:DropDownList>
```

To add list items programmatically, you must add `ListItem` objects to the control's `Items` collection, as in

```
ListItem li = new ListItem("Choose a color ");
li.Selected = true;
myDrop.Items.Add(li);
myDrop.Items.Add(new ListItem("Red"));
myDrop.Items.Add(new ListItem("Blue"));
// Specify both display text and value
myDrop.Items.Add(new ListItem("Green", "#00FF00"));
```

This code snippet illustrates the different ways that you can add items to a list control. Notice that the `ListItem` constructor is overloaded; you can specify just the display text of the item or both the display text and its value.

To programmatically process a multiselection list control, you typically need to iterate through the collection. Listings 3.13 and 3.14 illustrate the markup and code for displaying and iterating through a multiselection list box. Figure 3.11 illustrates the result in the browser window.

Listing 3.13 ListControlProgrammatic.aspx

```

<h1>Choose multiple colors (use ctrl or shift): </h1>
<asp:ListBox ID="myList" runat="server"
    SelectionMode="Multiple" Rows="6">
    <asp:ListItem Value="#FF0000">Red</asp:ListItem>
    <asp:ListItem Value="#FF00FF">Magenta</asp:ListItem>
    <asp:ListItem Value="#FFFF00">Yellow</asp:ListItem>
    <asp:ListItem Value="#00FF00">Green</asp:ListItem>
    <asp:ListItem Value="#00FFFF">Cyan</asp:ListItem>
    <asp:ListItem Value="#0000FF">Blue</asp:ListItem>
</asp:ListBox>
<p><asp:Button ID="myBtn" runat="server" Text="Submit"/></p>
<asp:Label ID="labResult" runat="server" />

```

Listing 3.14 ListControlProgrammatic.aspx.cs

```

/// <summary>
/// Called each time the page loads
/// </summary>
protected void Page_Load(object sender, EventArgs e)
{
    if (IsPostBack)
    {
        labResult.Text = "Colors chosen: <br/>";
        foreach (ListItem li in myList.Items)
        {
            if (li.Selected)
            {
                labResult.Text += "<span style='color:";
                labResult.Text += li.Value + "'> " + li.Text;
                labResult.Text += "</span><br/>";
            }
        }
    }
}

```

List items can also be added implicitly using data binding. Chapter 8 illustrates the use of data binding with list controls.

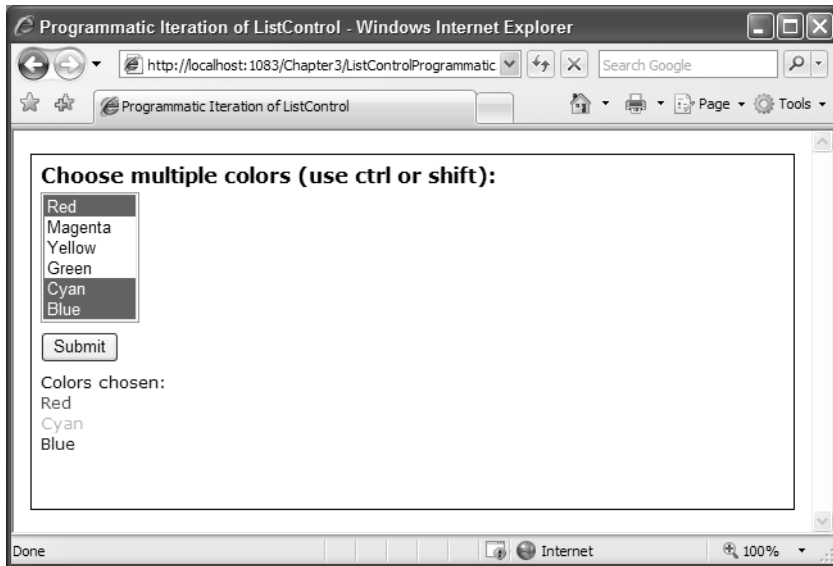


Figure 3.11 ListControlProgrammatic.aspx

BulletedList

The `BulletedList` control displays a bulleted list of items. This list can be ordered (a numbered list) or unordered (a bulleted list). The style of the bullet or the numbers can be set via properties of the control (see Table 3.15).

Table 3.15 Unique Properties of the `BulletedList` Control

Property	Description
<code>BulletImageUrl</code>	The path of the image to be used for the bullet if the <code>BulletStyle</code> is <code>CustomImage</code> .
<code>BulletStyle</code>	The style of the bullets in the list. Possible values are described by the <code>BulletStyle</code> enumeration (<code>Numbered</code> , <code>LowerAlpha</code> , <code>UpperAlpha</code> , <code>LowerRoman</code> , <code>UpperRoman</code> , <code>Disc</code> , <code>Circle</code> , <code>Square</code> , or <code>CustomImage</code>).
<code>DisplayMode</code>	The mode to display the list items. Possible values are described by the <code>DisplayMode</code> enumeration (<code>HyperLink</code> , <code>LinkButton</code> , or <code>Text</code>).

Table 3.15 Unique Properties of the BulletedList Control (*continued*)

Property	Description
FirstBulletNumber	Specifies the numeric value of the first item in an ordered (numbered) bulleted list.
Target	When a hyperlink in a bulleted list item is clicked, the Web page content is displayed in the window or a frame with this name.

Like the `CheckBoxList`, `RadioButtonList`, `DropDownList`, and `ListBox` controls, the `BulletedList` control has the `ListControl` class as its base class. Like in these other classes, individual items can be added declaratively to the list via the `<asp:ListItem>` element, as in the following.

```
<asp:BulletedList id="aList" runat="server"
    BulletStyle="LowerRoman">
    <asp:ListItem>Item 1</asp:ListItem>
    <asp:ListItem>Item 2</asp:ListItem>
    <asp:ListItem>Item 3</asp:ListItem>
</asp:BulletedList>
```

The list's appearance can be changed via the `DisplayMode` property. The default value for this property is `Text`, which simply displays the list items in a bulleted list. Other possible values for this property are `HyperLink` and `LinkButton`. Although both of these settings display the list as a series of hyperlinks (see Figure 3.12), they differ in their behavior after the user clicks the link. `HyperLink` navigates with no postback to the URL specified via the `Value` attribute, whereas `LinkButton` posts back to the server when the link is clicked.

Listings 3.15 and 3.16 illustrate how to use the `BulletedList` and how the `DisplayMode` changes the list's appearance and behavior. It uses a `DropDownList` that lets the user select the `DisplayMode`. The handler for the `DropDownList` then sets the `DisplayMode` programmatically (see Figure 3.12 for the final result).

Listing 3.15 BulletedListTest.aspx

```
<p>Choose a display mode:
<asp:DropDownList ID="drpDisplayMode" runat="server"
    OnSelectedIndexChanged="drpDisplayMode_SelectedIndexChanged"
    AutoPostBack="True">
    <asp:ListItem Selected="True">Text</asp:ListItem>
    <asp:ListItem>HyperLink</asp:ListItem>
    <asp:ListItem>LinkButton</asp:ListItem>
</asp:DropDownList>
</p>
```

```
<asp:BulletedList ID="blCompanies" runat="server"
    DisplayMode="Text"
    BulletStyle="Square" OnClick="blItems_Click">

    <asp:ListItem Value="http://www.microsoft.com">
        Microsoft</asp:ListItem>
    <asp:ListItem Value="http://www.adobe.com">
        Adobe</asp:ListItem>
    <asp:ListItem Value="http://www.oracle.com">
        Oracle</asp:ListItem>

</asp:BulletedList>
<p><asp:Label ID="labMsg" runat="server"></asp:Label></p>
```

Listing 3.16 BulletedListTest.aspx.cs

```
/// <summary>
/// Handler for bullet list click
/// (only called when DisplayMode = LinkButton)
/// </summary>
protected void blItems_Click(object sender,
    BulletedListEventArgs e)
{
    labMsg.Text = "The bullet item index you selected was "
        + e.Index;
}

/// <summary>
/// Handler for drop-down list of display modes
/// </summary>
protected void drpDisplayMode_SelectedIndexChanged(object sender,
    EventArgs e)
{
    // Get the selected value in the drop-down list
    string sMode = drpDisplayMode.SelectedItem.Text;

    // Convert the string to the display mode enum
    BulletedListDisplayMode mode = (BulletedListDisplayMode)
        Enum.Parse(typeof(BulletedListDisplayMode),
            sMode, true );

    // Change the display mode of bulleted list to selected value
    blCompanies.DisplayMode = mode;
}
```

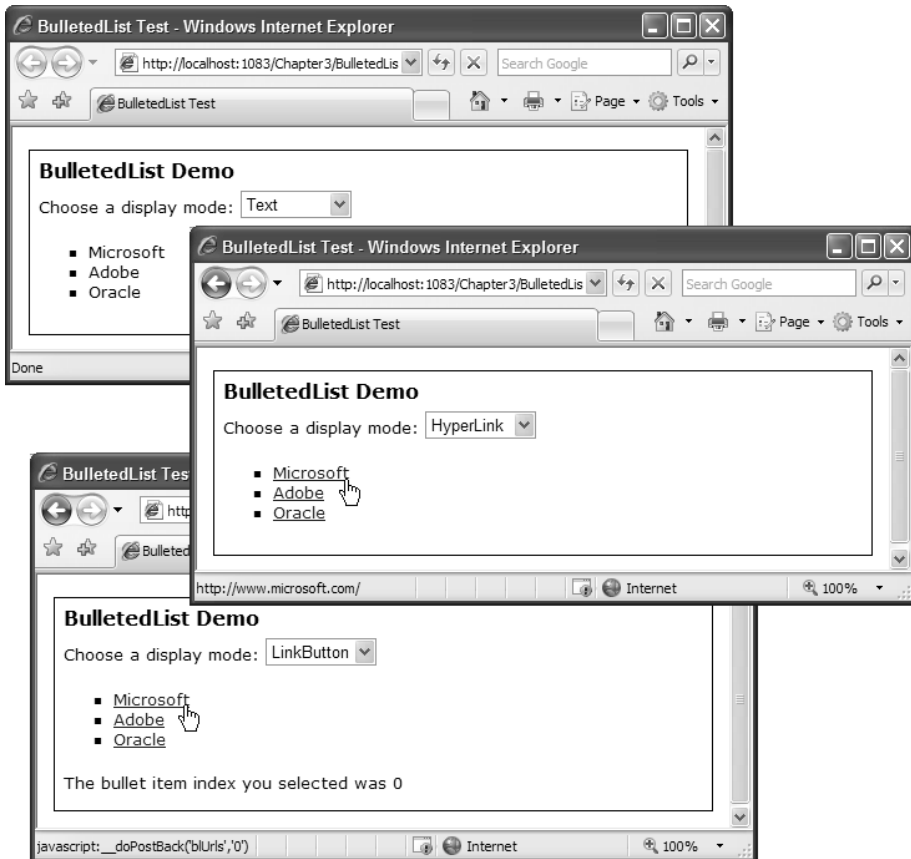


Figure 3.12 BulletedListTest.aspx

Notice that the `drpDisplayMode_SelectedIndexChanged` method in Listing 3.16 uses the `Parse` method of the `Enum` class to convert the string from the selected `DropDownList` to an enumerated value of type `BulletedListDisplayMode`. The more verbose alternative to this approach is to use conditional logic such as

```
string sMode = drpDisplayMode.SelectedItem.Text;
if (sMode == "Text")
    blCompanies.DisplayMode = BulletedListDisplayMode.Text;
else if (sMode == "HyperLink")
    blCompanies.DisplayMode = BulletedListDisplayMode.HyperLink;
else if (sMode == "LinkButton")
    blCompanies.DisplayMode = BulletedListDisplayMode.LinkButton;
```

RadioButtonList and CheckBoxList

The `RadioButtonList` and `CheckBoxList` controls encapsulate a list of radio buttons and check boxes. The `RadioButtonList` allows the user to select one item from a list of mutually exclusive radio buttons. The `CheckBoxList` allows the user to select multiple items from a list of check boxes.

Table 3.16 lists the unique properties of the `RadioButtonList` and `CheckBoxList` controls.

Table 3.16 Unique Properties of the `RadioButtonList` and `CheckBoxList` Controls

Property	Description
<code>CellPadding</code>	Specifies the pixel distance between the border and the contents of the table cell (if <code>RepeatLayout</code> is set to <code>Table</code>).
<code>CellSpacing</code>	Specifies the pixel distance between table cells (if using a table for layout).
<code>RepeatColumns</code>	Specifies the number of columns to use to display the list items.
<code>RepeatDirection</code>	Specifies that the list items are displayed horizontally in rows from left to right (<code>Horizontal</code>) or vertically in columns from top to bottom (<code>Vertical</code>). Possible values are described by the <code>RepeatDirection</code> enumeration.
<code>RepeatLayout</code>	Specifies whether the list is displayed within a table (<code>Table</code>) or not within a table (<code>Flow</code>). If the list is not displayed in a table, the list items are separated by line breaks (<code>
</code>). Possible values are described by the <code>RepeatLayout</code> enumeration. Default value is <code>Table</code> .
<code>TextAlign</code>	Specifies the horizontal alignment for items within the list. Possible values are described by the <code>TextAlign</code> enumeration (<code>Left</code> , <code>Right</code>).

The example in Listings 3.17 and 3.18 illustrates the use of both `RadioButtonList` and `CheckBoxList` controls. It also demonstrates the runtime setting of the layout and direction of these list controls.

Listing 3.17 `CheckAndRadioLists.aspx`

```
<div class="layout">
  Repeat Direction:
  <asp:DropDownList ID="drpDirection" runat="server">
```

```

        <asp:ListItem>Horizontal</asp:ListItem>
        <asp:ListItem Selected="True">Vertical</asp:ListItem>
    </asp:DropDownList><br />
    Repeat Layout:
    <asp:DropDownList ID="drpLayout" runat="server">
        <asp:ListItem Selected="True">Table</asp:ListItem>
        <asp:ListItem>Flow</asp:ListItem>
    </asp:DropDownList><br />
    Repeat Columns:
    <asp:DropDownList ID="drpColumns" runat="server">
        <asp:ListItem>1</asp:ListItem>
        <asp:ListItem>2</asp:ListItem>
        <asp:ListItem>3</asp:ListItem>
    </asp:DropDownList>
    <br />
    <asp:Button ID="btnChange" runat="server"
        Text="Change Properties" OnClick="btnChange_Click" />
</div>
<div class="box">
<p>
Crust:
<br>
<asp:RadioButtonList ID="rlstCrust" runat="server" >
    <asp:ListItem>Thin</asp:ListItem>
    <asp:ListItem>Medium</asp:ListItem>
    <asp:ListItem Selected="True">Thick</asp:ListItem>
</asp:RadioButtonList>
</p>
<p>
Toppings:<br>
<asp:CheckBoxList ID="clstToppings" runat="server"
    RepeatColumns="2"
    RepeatDirection="Vertical" RepeatLayout="Table" >
    <asp:ListItem>Ham</asp:ListItem>
    <asp:ListItem>Mushrooms</asp:ListItem>
    <asp:ListItem>Pepperoni</asp:ListItem>
    <asp:ListItem>Tomatoes</asp:ListItem>
    <asp:ListItem>Green Peppers</asp:ListItem>
    <asp:ListItem>Shrimp</asp:ListItem>
    <asp:ListItem>Extra Cheese</asp:ListItem>
    <asp:ListItem>Anchovies</asp:ListItem>
</asp:CheckBoxList>
</p>
<asp:Button ID="btnOrder" runat="server" Text="Order"
    OnClick="btnOrder_Click" />
</div>
<p><asp:Label ID="labMessage" runat="server" /></p>

```

Listing 3.18 CheckAndRadioLists.aspx.cs

```
/// <summary>
/// Handler for change properties button
/// </summary>
protected void btnChange_Click(object sender, EventArgs e)
{
    // Apply the layout settings to the two lists

    // First get the number of columns and apply to lists
    int columns = Convert.ToInt32(drpColumns.SelectedValue);
    rlstCrust.RepeatColumns = columns;
    clstToppings.RepeatColumns = columns;

    // Now get the layout and apply to lists
    string sLayout = drpLayout.SelectedValue;
    RepeatLayout layout = (RepeatLayout)Enum.Parse(
        typeof(RepeatLayout), sLayout, true);
    rlstCrust.RepeatLayout = layout;
    clstToppings.RepeatLayout = layout;

    // Finally get the repeat direction and apply to lists
    string sDirect = drpDirection.SelectedValue;
    RepeatDirection direct = (RepeatDirection)Enum.Parse(
        typeof(RepeatDirection), sDirect, true);
    rlstCrust.RepeatDirection = direct;
    clstToppings.RepeatDirection = direct;
}

/// <summary>
/// Handler for order button
/// </summary>
protected void btnOrder_Click(object sender, EventArgs e)
{
    // Get the crust from the radio list and display
    labMessage.Text = "<b>Pizza Ordered: </b><br>";
    labMessage.Text += rlstCrust.SelectedItem.Text;
    labMessage.Text += " Crust<br/><b>Toppings:</b><br/>";

    // Get all the toppings selected in list and display
    foreach (ListItem topping in clstToppings.Items)
    {
        if (topping.Selected)
        {
            labMessage.Text += topping.Text + "<br/>";
        }
    }
}
}
```

Figure 3.13 illustrates the result in the browser window. Be sure to use the browser's View Source option and compare how the `RepeatLayout` property changes the rendered HTML.

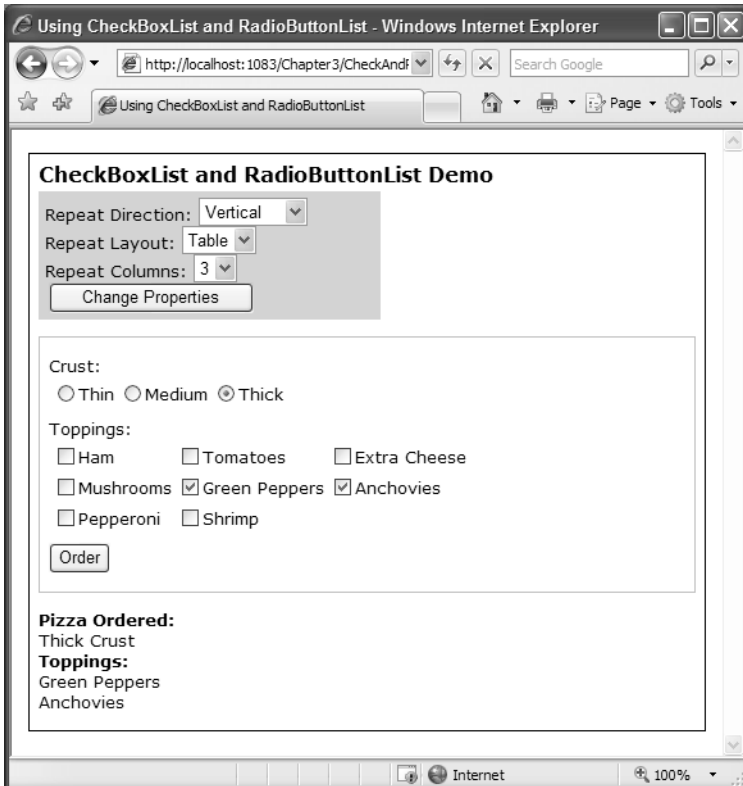


Figure 3.13 CheckAndRadioLists.aspx

Image Control

The `Image` control is used to display an image on the page. It provides the same functionality as the HTML `IMG` element, but allows you to programmatically set image properties in server code. For instance, the image URL can be set at design time or runtime and can even be bound to a value from a database.

Note that the `Image` control simply displays an image; it does not act like a button. If you want an image to have the behavior of a button, you must use the `ImageButton` control. If you need to determine the coordinates of where the image was clicked, use the `ImageMap` control.