msdn® training

Module 2: Using Web Controls

Contents

Microsoft®

Overview

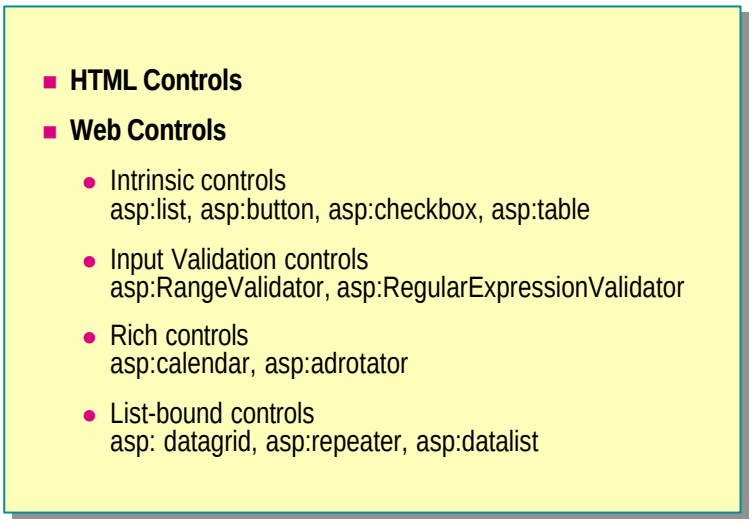
- What Are Web Controls?
- Using Intrinsic Controls
- Using Input Validation Controls
- Selecting Controls for Applications

Web controls are designed to provide a simple way to add functionality, such as displaying data or selecting dates, to a Web page. Web controls include traditional form controls, such as buttons and text boxes, and complex controls, such as tables and the calendar control.

After completing this module, you will be able to:

- Add Web controls to a Microsoft® ASP.NET page.
- Use properties, methods, and events of Web controls.
- Validate user input on an ASP.NET page by using input validation controls.
- Bind two controls together.

What Are Web Controls?



ASP.NET uses server controls extensively to simplify the task of programming Web pages. You can change many Hypertext Markup Language (HTML) elements into server controls by adding the **runat=server** attribute to the element tag. However, ASP.NET includes new controls, called Web controls, which have built-in functionality and are the preferred controls to use in an ASP.NET page.

Web controls include traditional form controls, such as buttons and text boxes, and complex controls, such as tables. They also include controls that provide commonly used form functionality such as displaying data in a grid, choosing dates, and so on.

There are four sets of Web controls in ASP.NET:

- Intrinsic controls map to simple HTML elements.
- Validation controls provide a variety of data validation.
- Rich controls provide rich user interface and functionality. Currently ASP.NET ships with two rich controls, the **Calendar** and the **AdRotator** controls.
- List-bound controls provide data flow across a page.

  The **DataGrid**, **DataList**, and **Repeater** controls constitute an interrelated set of Web controls. These controls render HTML to display the contents of a list or data source to which they are bound. Hence, they are collectively referred to as list-bound controls.

Note For more information about displaying data from databases, see Module 3, " Using Microsoft ADO.NET to Access Data," in Course 2063B, *Introduction to Microsoft ASP.NET*.

Web controls exist within the **System.Web.UI.WebControls** namespace. You create them by using the **asp:** namespace. You do not need to import this namespace into your .aspx page to use these controls. For example, the following sample code creates a button Web control:

```
<asp:Button runat="server" />
```

Note All controls must be well-formed and must not overlap. Unless otherwise noted, elements must be closed, either with an ending slash within the tag or with a closing tag. For example, <asp:Button runat="server" /> or < asp:Button runat="server" > </asp:Button>.

For more information about the Web controls available with ASP.NET, see " ASP.NET Syntax for Web Controls" in the .NET Framework SDK documentation.

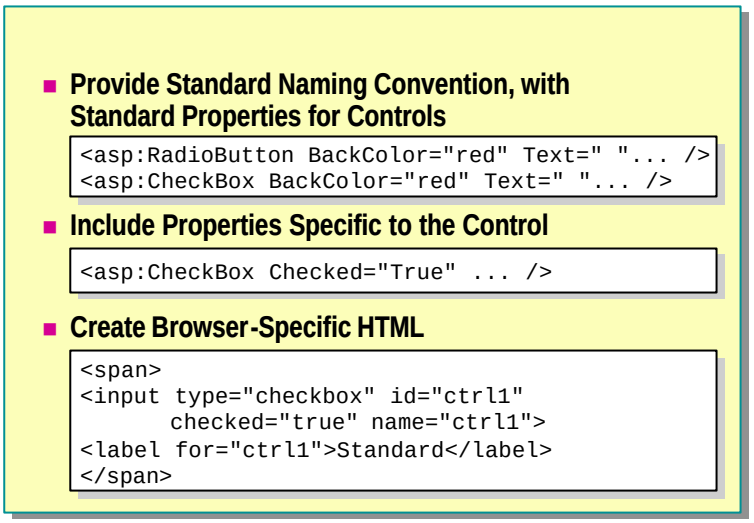
◆ Using Intrinsic Controls

- What Are Intrinsic Controls?
- List of Intrinsic Controls
- Handling Intrinsic Control Events
- Demonstration: Adding Intrinsic Controls to a Page
- Linking Two Controls Together
- Demonstration: Linking Two Controls Together

Intrinsic controls are designed to replace the standard set of HTML controls.

This section describes intrinsic controls and explains how to use them in an ASP.NET page.

What Are Intrinsic Controls?



Intrinsic controls are used as an alternative to HTML controls. You use Web controls in the same way that you use HTML controls. The most important difference is the prefix tag **asp:** that is added before every Web control. For example, you can add a **Label** control to a form by using the following line of code:

```
<asp:Label runat="server" Text="Label1"></asp:Label>
```

Intrinsic Web controls provide the following benefits:

- Provide a standard naming convention for similar controls
- Provide common properties for all controls
- Include strongly typed properties that are specific to the control
- Create browser-specific HTML

Standard Naming Convention

One of the problems with HTML controls has been the lack of a consistent naming convention for similar controls. Intrinsic controls eliminate this problem, as shown in the following examples.

Example 1

The following lines of code declare a radio button, a check box, and a button.

```
<input type ="radio">
<input type ="checkbox">
<input type ="submit">
```

All of these controls are input controls; however, they all behave in different ways.

Intrinsic controls provide an alternative way to declare different controls. The following lines of code use intrinsic Web controls to declare a radio button, a check box, and a button:

```
<asp:RadioButton>
<asp:CheckBox>
<asp:Button>
```

Include Common Properties for Controls

In ASP.NET, common properties for intrinsic controls have the same name. For example, when you want to set the background color for a control, you always use the same attribute irrespective of the control. For example, all controls have a **Text** property that sets the text corresponding to the control. Controls also have **BackColor** and **ForeColor** properties. Setting the BackColor="red" property sets the background color to red for any control.

```
<asp:textbox id="txtName" Text="Jane" BackColor="red"
runat="server" />

<asp:Button id="btnSubmit" Text="Submit" BackColor="cyan"
runat="server" />
```

Control-Specific Properties

Intrinsic Web controls also have a set of properties that relate specifically to the control.

Check Box

For example, a check box has a **Checked** property that indicates whether the check box is checked or not.

The following sample code creates a check box labeled Standard that is initially checked.

```
<asp:CheckBox Checked="True" Text="Standard" runat="server" />
```

List Box

Some properties of Web controls are available only at run time. For example, the list box and drop-down list boxes have the **SelectedItem** property that returns either the value or text of the item selected in the list box.

Create Browser-Specific HTML

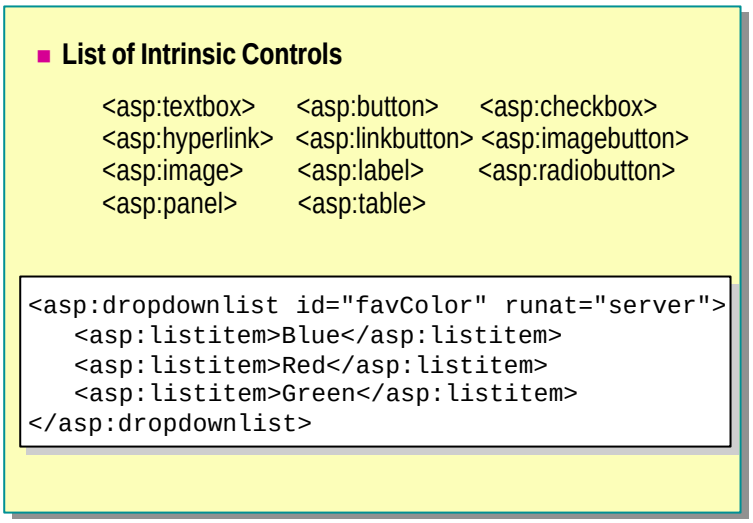
Although Web controls are easily added to a page, they are powerful controls. When a page is rendered for a browser, the Web controls determine which browser is requesting the page and then deliver the appropriate HTML.

For example, if the requesting browser supports client-side scripting, such as Microsoft Internet Explorer version 4.0 or later, the controls create client-side script to implement their functionality. But, if the requesting browser does not support client-side script, the controls create server-side code and require more round trips to the server to obtain the same functionality.

For example, the HTML generated for the above check box in Internet Explorer 5.5 is:

```
<span>
<input type="checkbox" id="ctrl1" checked="true" name="ctrl1">
<label for="ctrl1">Standard</label>
</span>
```

List of Intrinsic Controls



ASP.NET intrinsic controls include traditional form controls, such as buttons and text boxes, and complex controls, such as tables.

Examples of Intrinsic Controls

The following table lists some of the intrinsic Web controls and their corresponding HTML controls.

Web control	HTML control
<asp:textbox>	<input type=text>
<asp:button>	<input type=submit>
<asp:imagebutton>	<input type=image>
<asp:checkbox>	<input type=checkbox>
<asp:radiobutton>	<input type=radiobutton>
<asp:listbox>	<select size="5"> </select>
<asp:dropdownlist>	<select> </select>
<asp:hyperlink>	
<asp:image>	<img src="…">
<asp:label>	
<asp:panel>	
<asp:table>	<table> </table>

Adding Intrinsic Controls

Drop-Down List Box

The **DropDownList** control is used to allow a single selection from a number of choices displayed as a drop-down list:

You can add a list box to a form by using the following code:

```
<asp:dropdownlist id="favColor" runat="server">
  <asp:listitem>Blue</asp:listitem>
  <asp:listitem>Red</asp:listitem>
  <asp:listitem>Green</asp:listitem>
</asp:dropdownlist>
```

Label

You can add a label control to a form as follows:

```
<asp:Label runat="server" Text="Label1" Font-
Italic="true"></asp:Label>
```

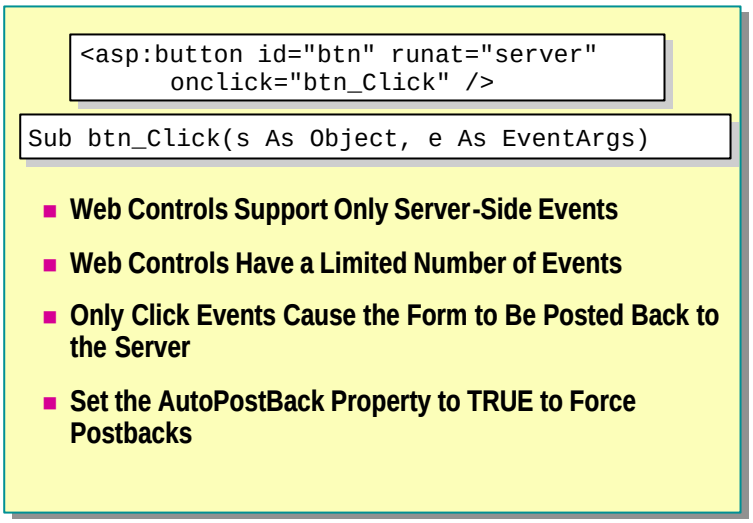
Check box

The **CheckBox** control is used to generate a check box that can be toggled between selected and cleared states:

```
<asp:CheckBox runat="server" Text="CheckBox1"
Checked="True"></asp:CheckBox>
```

For more information about the properties and events associated with the intrinsic controls, see the .NET Framework SDK documentation.

Handling Intrinsic Control Events



Events in Web forms work differently than events in traditional client forms or in client-based Web applications. In client-based applications, events are raised and handled on the client. In Web forms, all events are raised on the client but handled on the Web server.

Web controls support server-side events only. Web controls have a limited number of events because all events are server-side events and cause a round trip to the server. They support click-type events. Some server controls also support a special version of an **onChange** event, such as **SelectionChanged**, which is raised when the control's value changes. However, mouse-key events are not supported.

Using Event Procedures

As with HTML control events, writing event handlers for Web controls is a two-step procedure. You need to set up the event handler in the control's tag, and then write the event procedure itself.

For example, you set up a click event for a button control as follows:

```
<asp:button id="btn" runat="server" onclick="btn_Click" />
```

Note The button Web control has only an **OnClick** event, not an **OnServerClick** event.

You then write the event procedure as follows:

```
Sub btn_Click(s As Object, e As EventArgs)

End Sub
```

For more information on the events available for intrinsic controls, see the Microsoft .NET Framework SDK documentation.

Note Web controls do not support client-side events. HTML controls have both server-side and client-side events.

Using AutoPostBack

In Web controls, by default, only click events cause the form to be posted to the server. Change events are captured, but do not immediately cause a postback. Instead, they are cached by the control until the next time a post occurs. Once the form is posted to the server, the associated event procedures will run.

During server page processing, all change events are processed first, in no particular order. When all change events have been processed, the click event that caused the form to be posted is processed.

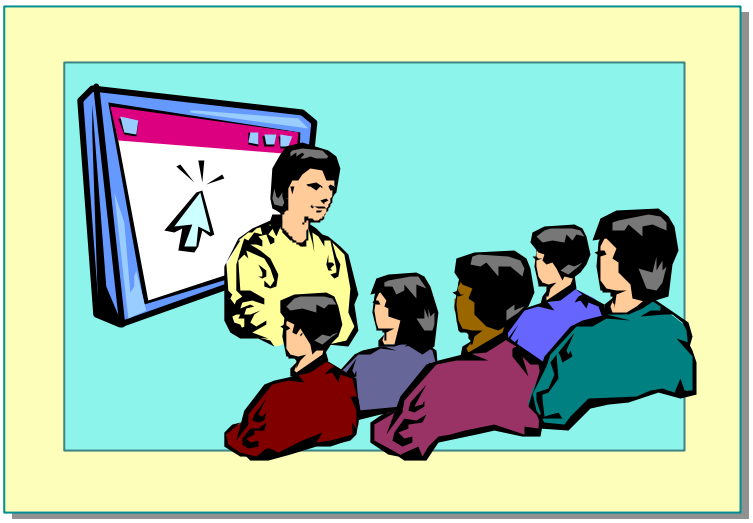
You can specify a change event to cause a form postback, and thus run the event procedure immediately, by setting the **AutoPostBack** property to **True**.

For example, you can set up a change event for a list box with the following tag:

```
<asp:dropdownlist id="lstTitle" runat="server"
  onSelectedIndexChanged="lst_change" >
```

However, until you set the **autoPostBack** property to **True**, the **lst_change** event procedure will run only when the form is posted to the server in response to a click event of a button.

Demonstration: Adding Intrinsic Controls to a Page



In this demonstration, you will learn how to add intrinsic Web controls to an ASP.NET page. You will see the HTML generated by this page for Internet Explorer and understand what happens when you add intrinsic controls to a form. Next, you will learn how to add event procedures for the controls.

The file <*install folder*>\Democode\Mod02\demo1.aspx has the completed code for this demonstration.

↙ To run the demonstration

1. Open intrinsic.aspx from the <*install folder*>\Democode\Mod02 folder.

   There are three intrinsic controls in this file: a text box, a list box, and a button, plus two elements. There are also event procedures created for the button and the list box.

2. View the page in Internet Explorer. Enter text in the text box and click **Save**.

   Nothing happens when you click the button or select an item in the list box. Events are fired only for controls on forms.

3. Edit the page and add a form with the **runat="server"** attribute to the page to hold the controls.

4. View the page in Internet Explorer again. Enter text in the text box and click **Save**.

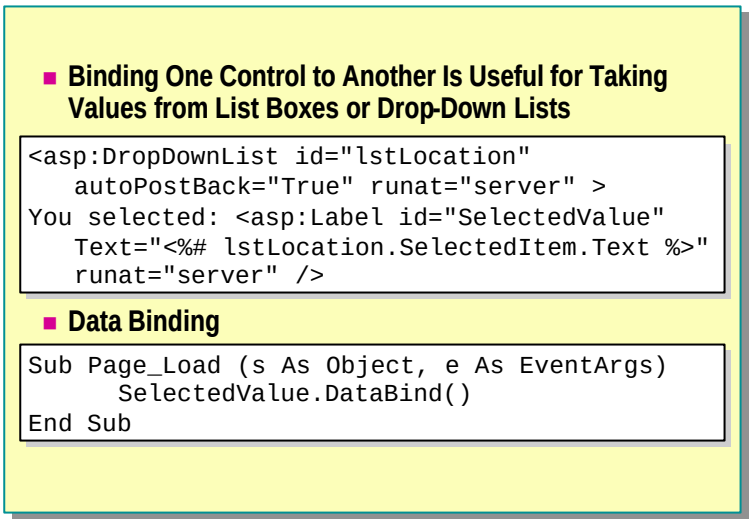
   View state is now preserved for the text box and the list box. The button procedure runs when you click the button, but the list box event procedure does not fire until the button is clicked.

5. Edit the page and set the **AutoPostBack** attribute of the list box to **True**.

6. View the page in Internet Explorer again. Enter text in the text box and click **Save**.

   The list box event procedure fires immediately when you select an item.

Linking Two Controls Together



You can bind one control to the contents of another. This is particularly useful for taking values from list boxes or drop-down lists.

The following example demonstrates how to bind a **Label** control to the contents of a drop-down list. You set the **Text** attribute of the **Label** control to the **Selected Item** of the **lstLocation** list by using the binding tags **<%#** and **%>**:

```
<asp:Label id="SelectedValue" runat="server"
  Text="<%# lstLocation.SelectedItem.Text %>" />
```

In the following code, notice that the **autoPostBack** property of the drop-down list is set to **True**, which causes automatic post back whenever the value of the list box changes.

```
<script language="VB" runat="server">
Sub Page_Load (Sender As Object, E As EventArgs)
  SelectedValue.DataBind()
End Sub
</script>

<form runat="server">
  <asp:DropDownList id="lstLocation"
      autoPostBack="true" runat="server" >
      <asp:listitem>Boston</asp:listitem>
      <asp:listitem>New York</asp:listitem>
      <asp:listitem>London</asp:listitem>
      <asp:listitem>Amsterdam</asp:listitem>
  </asp:dropdownlist>
  <p>You selected: <asp:Label id="SelectedValue"
      Text="<%# lstLocation.SelectedItem.Text %>"
      runat="server" />
  </p>
</form>
```

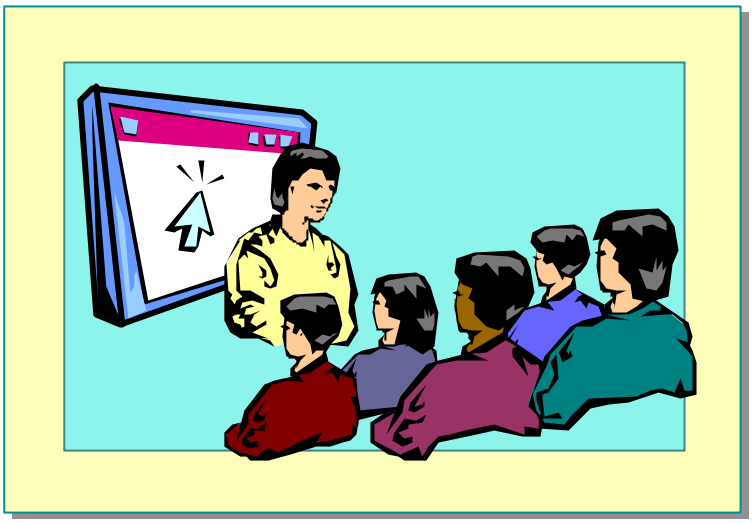
You can use **Page.DataBind()** if you want the page to data bind all elements on the page. The preceding example binds only the label control to data, and therefore uses the *SelectedValue*.DataBind() syntax, where **SelectedValue** is the **id** attribute of the label control.

While this lesson describes using the binding tags **<%#** and **%>**, it is often a better practice to assign values from one control directly to another, rather than using data binding. The following line of code demonstrates assigning values from one control to another:

```
SelectedValue.Text=lstLocation.SelectedItem.Text
```

For more information about data binding, see Module 3, " Using Microsoft ADO.NET to Access Data," in Course 2063B, *Introduction to Microsoft ASP.NET*.

Demonstration: Linking Two Controls Together



In this demonstration, you will see how to bind a label control to a list box.

The completed code for this demonstration is in the <*install folder*>\Democode\Mod02\binding.aspx page.

↙ To run the demonstration

1. Edit the page you created in the last demo, <*install folder*>\Democode\Mod02\intrinsic.aspx
2. Add a label control to the page.

```
<asp:label id="lblListbox" runat="server"/>
```

3. Bind the label control to the list box by setting its **Text** attribute.

```
<asp:label id="lblListbox"
Text="<%# lstTitle.SelectedItem.Text %> " runat="server"/>
```

4. Create a Page_Load event and call lblListbox.DataBind()

```
Sub Page_Load(s As Object, e As EventArgs)
   lblListbox.DataBind()
End Sub
```

5. View the page in Internet Explorer.

   When you select an item in the list box, the value is reflected in the label control.

◆ Using Input Validation Controls

- Input Validation Controls
- Adding Input Validation Controls to a Form
- Validating an Entire Page
- Demonstration: Adding Input Validation Controls
- Advantages of Using Input Validation Controls

In HTML 3.2, validating data is a difficult process. You can run validation code on the client, on the server, or both. Each time you receive a request, you not only need to write code that checks the input, you need to write error messages to help the user to correctly fill in the form. This is a taxing process for users, developers, and servers.

The introduction of input validation controls in ASP.NET makes the task of validating input easier than it has been in the past.

In this section, you will learn about the advantages of using input validation controls. You will also learn how to add these controls to an ASP.NET page.

Input Validation Controls

- RequiredFieldValidator
- CompareValidator
- RangeValidator
- RegularExpressionValidator
- CustomValidator
- ValidationSummary

Validation controls are added to an ASP.NET page like other server controls. There are controls for specific types of validation, such as range checking or pattern matching, plus a **RequiredFieldValidator** control that ensures that a user does not skip an entry field. Validation controls work with all server controls.

The validation controls have both uplevel and downlevel client support. Uplevel browsers perform validation on the client (using JavaScript and dynamic HTML [DHTML]). Client-side validation enhances the validation scheme by checking user input as the user enters data. This allows errors to be detected on the client before the form is submitted, preventing the round trip necessary for server-side validation. In uplevel checking, if there is an error in user input, a trip is not made to the server; instead, the associated error messages are shown on the client.

Downlevel browsers perform validation on the server. Both scenarios use the same programming model.

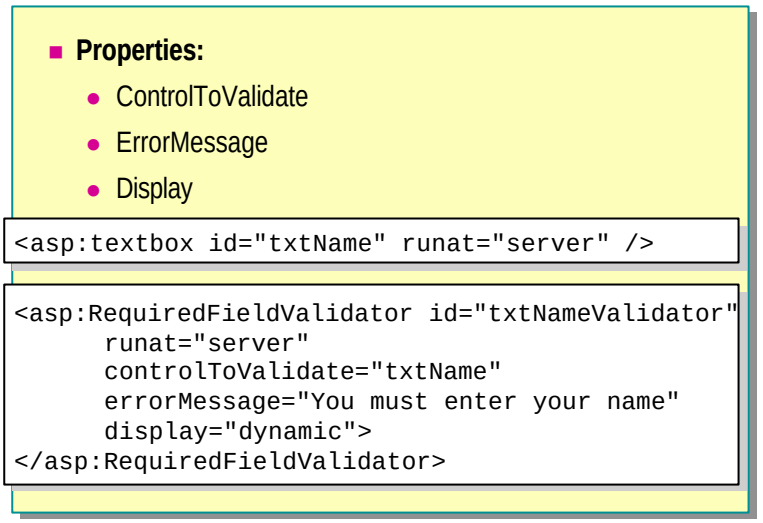
Note More than one input validation control can be associated with a Web control. For example, there can be required and compare validators for the same text box Web control.

The following table lists the validation controls included in the ASP.NET Framework.

Validation control	Function
RequiredFieldValidator	Checks whether value has been entered into a control.
CompareValidator	Compares an input control to a fixed value or another input control. It can be used for password verification fields, for example. It is also possible to do typed date and number comparisons.
RangeValidator	Much like **CompareValidator**, but can check that the input is between two values or the values of other input controls.
RegularExpressionValidator	Checks that the entry matches a pattern defined by a regular expression. This type of validation allows you to check for predictable sequences of characters, such as those in social security numbers, e-mail addresses, telephone numbers, postal codes, and so on.
CustomValidator	Allows you to write your own code to take part in the validation framework.
ValidationSummary	Displays a summary of all of the validation errors for all of the validation controls on the page.

For more information about using the input validation controls, see "Validation Server Controls Overview" in the .NET Framework SDK documentation.

Adding Input Validation Controls to a Form



The standard code used to add input validation controls to a form is as follows:

```
<asp:type_of_validator id="validator_id" runat="server"
  controlToValidate="control_id"
  errorMessage="error_message_for_summary"
  display="static|dynamic|none">
</asp:type_of_validator>
```

In the preceding code, **controlToValidate** is the ID of the control whose input is being validated (a text box, in most cases). The **errorMessage** property is the error message that is displayed if the input is not valid, and the **display** property indicates how the validation control will be rendered in the page. Setting the **display** property to static implies that each validation control occupies space even when no error message text is visible, thereby allowing you to define a fixed layout for the page. Setting the **display** property to dynamic means that the layout of the page changes, sometimes causing controls to move as error messages are displayed. Validation controls cannot occupy the same space on the page, so you must give each control a separate location on the page.

In addition to the above-mentioned required properties, each validation control has more specific properties that define how the control should behave. The following examples show how to add validation controls.

RequiredFieldValidator Control

Use the **RequiredFieldValidator** control to require input in a field.

The following code shows how to use the **RequiredFieldValidator** control.

```
<asp:textbox id="txtName" value="Enter your name"
  runat="server" />
<asp:RequiredFieldValidator id="txtNameValidator"
  runat="server"
  controlToValidate="txtName"
  errorMessage="You must enter your name"
  display="dynamic">
</asp:RequiredFieldValidator>
```

RangeValidator Control

To test whether an input value falls within a given range, use the **RangeValidator** control.

The **RangeValidator** control uses three additional properties to perform its validation—**MinimumValue**, **MaximumValue**, and **type**. **MinimumValue** and **MaximumValue** properties define the minimum and maximum values of the valid range. The **type** property specifies the data type that that is used in comparing values.

In the following example, the **txtAge** field must have a value between 18 and 50. If the value does not fall within the range, an error message is displayed and no trip to the server occurs.

```
<asp:textbox id="txtAge" value="Enter your age" runat="server"
/>

<asp:RangeValidator id="txtAgeValidator" runat="server"
  controlToValidate="txtAge"
  type="Integer"
  minimumValue="18"
  maximumValue="50"
  errorMessage="Applicants must be between 18 and 50"
  display="dynamic">
</asp:RangeValidator>
```

CompareValidator Control

To use one control to test the validity of a second control, or to test input against a specific value, use the **CompareValidator** control.

The **CompareValidator** control uses the following additional properties to perform its validation:

- **ValueToCompare** or **ControlToCompare**

  Set **ValueToCompare** to compare to a constant value. Set **ControlToCompare** to the ID of another control to compare against. If you set both **ValueToCompare** and **ControlToCompare**, **ControlToCompare** takes precedence.

- **Type**

  The data type of the two values to be compared.

- **Operator**

  The comparison operator to use. Operators are specified with the name of the comparison operators, such as **Equal**, **NotEqual**, **GreaterThan**, **GreaterThanEqual**, and so on.

The following code shows an example in the .aspx file of a Web form **Textbox** control with required field validation. A table is used to control layout. In the following example, the **txtAge** field must have a value greater than or equal to zero.

```
<asp:textbox id="txtAge" runat="server" />
<asp:CompareValidator id="txtAgeCompareValidator"
  runat="server"
  ControlToValidate="txtAge"
  ValueToCompare="0"
  Type="Integer"
  Operator="GreaterThanEqual"
  ErrorMessage="Please enter a whole number zero or greater.">
</asp: CompareValidator >
```

RegularExpressionValidator Control

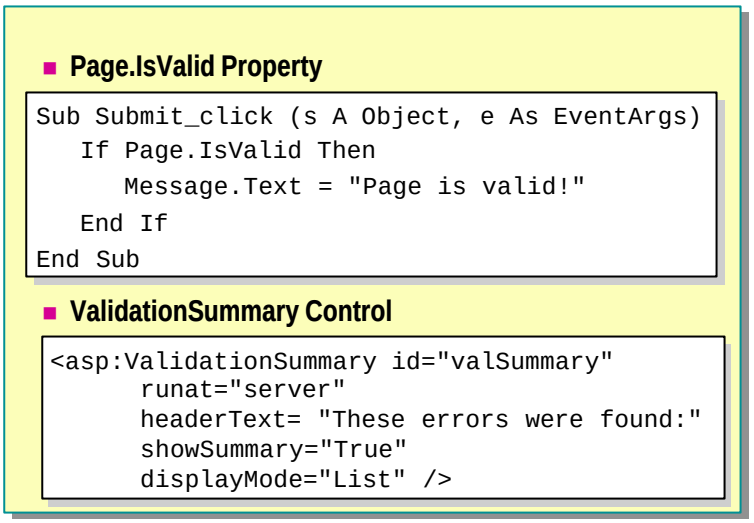
You can check whether a user's entry matches a pre-defined pattern, such as a phone number, postal code, e-mail address, and so on, with the **RegularExpressionValidator** control.

Set the **ValidationExpression** property of the **RegularExpressionValidator** control to a regular expression.

The following example shows how you can use a **RegularExpressionValidator** control to check whether users enter valid social security numbers. The control checks for the pattern: three digits, a hyphen, two digits, a hyphen, and four more digits.

```
<asp:textbox id="txtSSN" runat="SERVER" />
<asp:RegularExpressionValidator id="txtZIP_validation"
  runat="server"
  ControlToValidate="txtSSN"
  ErrorMessage="Use the format 123-12-1234."
  ValidationExpression="[0-9]{3}-[0-9]{2}-[0-9]{4}">
</asp:RegularExpressionValidator>
```

Validating an Entire Page



Sometimes you need to know if all the controls on a page are valid before performing some other action. In fact, in client-side validation, values are sent across to the server only when all controls are valid.

Page.IsValid Property

ASP.NET developers can check the **Page.IsValid** property at run time to determine whether all validation server controls on a page are currently valid. This provides a short and simple way to determine whether to proceed with business logic.

```
Sub Submit_click (s A Object, e As EventArgs)
  If Page.IsValid Then
     Message.Text = "Page is valid!"
  End If
End Sub
```

Note The validation can be either client-side or server-side, depending on the browser.

Validation Summary Control

If a validation control is in error, an error message may be displayed in the page by the validation control or in the **ValidationSummary** control elsewhere on the page. The **ValidationSummary** control is displayed when the **IsValid** property of the page is false. Each of the validation controls on the page is polled and the text messages exposed are aggregated by the **ValidationSummary** control.

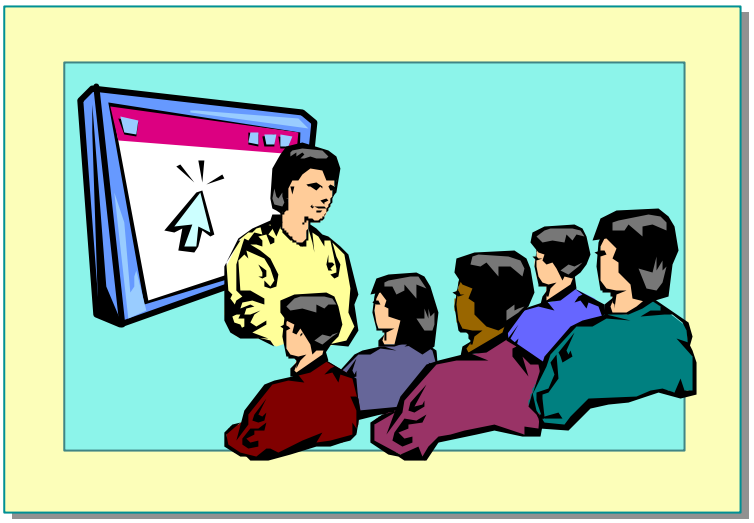
- The **ValidationSummary** updates itself without posting back if it detects errors.
- The **ValidationSummary** can optionally display a message box to the user if there are errors.

```
<asp:ValidationSummary id="valSummary" runat="server"
   headerText="These errors were found:"
   showSummary="True"
   displayMode="List" />
```

There are two places where you can display error messages to the user: in the input validation control or in the **ValidationSummary** control. If there is a **ValidationSummary** control on the form, it gathers and displays all **ErrorMessage** properties of input validation controls on the form, and the validation controls themselves display any error text that is contained between their start and end tags.

If there is no **ValidationSummary** control, the input validation controls display the text in the **ErrorMessage** property.

Demonstration: Adding Input Validation Controls



In this demonstration, you will see how to add input validation controls to text controls on a form.

The completed code for this demonstration is in the <*install folder*>\Democode\Mod02\validation.aspx and <*install folder*>\Democode\Mod02\validate_all.aspx pages.

↙ To run the demonstration

1. Edit the page you created in the last demonstration <*install folder*>\Democode\Mod02\intrinsic.aspx.
2. Add a **RequiredField** validator to the **txtName** text box.

```
<asp:RequiredFieldValidator id="txtNameValidator"
runat="server"
   controlToValidate="txtName"
   errorMessage="You must enter your name"
   display="dynamic">
</asp:RequiredFieldValidator>
```

3. View the page in Internet Explorer. Leave the text box blank and click **Save**.

   The error message from the **errorMessage** attribute appears.
4. View the source of the page. The input validation control generated client-side script to do the validation.

5. Add a **RangeValidator** control to limit the entries to names that start with "A".

```
<asp:RangeValidator id="txtNameRngValidator" runat="server"
   controlToValidate="txtName"
   errorMessage="Please enter a name that begins with 'A'"
   type="String"
   minimumValue="A"
   maximumValue="B"
   display="dynamic">
</asp:RangeValidator>
```

6. Set the **display** attribute of the **RequiredFieldValidator** to **static**.
7. View the page in Internet Explorer. Enter invalid text in the text box and click **Save**.

   The error message from the **errorMessage** attribute of the **RangeValidator** control appears, but it is offset by the area required by the **RequiredFieldValidator**.
8. Open the page validate_all.aspx.

   There are a number of input validation controls used on this page, and a validation summary control.
9. View the page validate_all.aspx in Internet Explorer.

   When you click the **Validate** button, the **ValidationSummary** control displays all the **errorMessage** attributes of controls that are not valid.

Advantages of Using Input Validation Controls

- Provide Immediate Feedback to the User
- Create Client-Side and Server-Side Validation
- Uses DHTML and JScript Library to Write Client Logic
- Allow Enhancement of Client-Side Validation and Behavior

Most Web sites perform their validation checks on the server. However, most applications also need client-side validation checks. Writing two versions of validation code for these validation checks can be time-consuming for developers.

Validation controls eliminate this problem because almost all the duplicated logic is encapsulated in the controls. Validation controls check controls for a specific type of error condition and display a description of that problem. In addition, the controls create browser-specific code, so that, if a client with Internet Explorer 4.0 or later uses the page, it can perform the same input validation that takes place on the server without client script.

Using validation controls has the following advantages:

- Provides immediate feedback to the user.

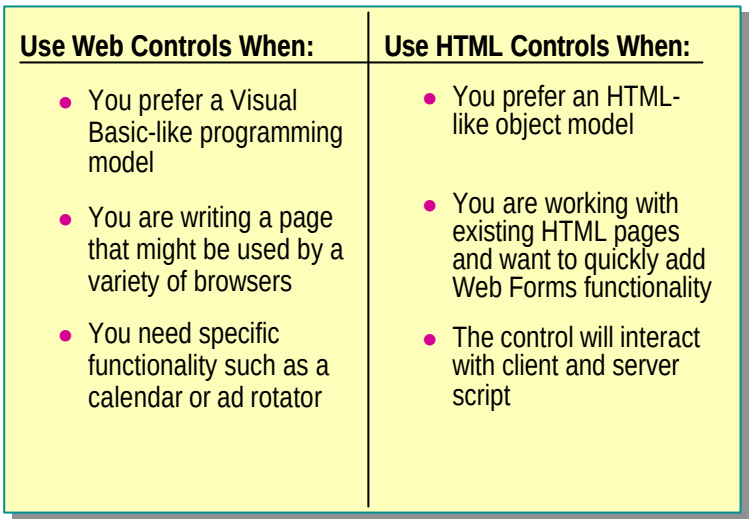
  The user gets immediate feedback on whether the data entered is valid or not. The error messages appear and disappear immediately after the bad input is entered or corrected.
- Creates client-side and server-side validation.

  If the browser supports client-side script, the controls do error checking before posting the data to the server. This saves the user time and reduces hits on the server.
- Uses DHTML and a Microsoft JScript® library to write client logic.

  The client logic is contained in a JScript library, so no Microsoft ActiveX® components or Java applets are used.
- Allows enhancement of client-side validation and behavior.

  An object model is exposed on the client to allow enhancement of client-side validation and behavior.

Selecting Controls for Applications



When creating ASP.NET pages, you can use different classes of controls.

You can mix control types on the same page. For example, your ASP.NET page might include a form made up of Web controls plus an HTML control converted from an HTML element.

The following table summarizes some of the situations in which you might decide to use a particular control.

Type of control	Use it when
Web control	You prefer a Microsoft Visual Basic®-like programming model.
	You are writing a Web Forms page that might be viewed by a variety of browsers.
	You need specific functionality, such as a calendar or ad rotator, that is available only as a Web Forms control.
	You are creating applications with nested controls and want to be able to catch events at the container level.
HTML control	You prefer an HTML-like object model.
	You are working with existing HTML pages and want to quickly add Web Forms functionality. Because HTML controls map exactly to HTML elements, any HTML design environment can support them.
	The control will also interact with client and server script.

As a general rule, Web controls are more capable and have a richer object model than HTML controls. If you want all the processing of an ASP.NET page to occur on the server, Web controls are a good choice.

Lab 2: Using Web Controls

Objectives

After completing this lab, you will be able to:

- Use Web controls in an ASP.NET page.
- Use input validation controls in an ASP.NET page.
- Display the summary of validation controls in a form.
- Test if a page is valid or not.
- Add a Calendar control to an ASP.NET page.

Prerequisite

Before working on this lab, you must know how to create ASP.NET event procedures.

Lab Setup

There are starter and solution files associated with this lab. The starter files are in the folder <*install folder*>\Labs\Lab02\Starter and the solution files for this lab are in the folder <*install folder*>\Labs\Lab02\Solution.

Scenario

When users want to reserve a room in a hotel for a conference, they need to select the type of room and specify the check-in and check-out dates. This is done on the hotel_reservation.aspx page. In exercises 1 and 2 of this lab, you will add Web controls to this page.

Before registering for a conference on the Conference Web site, a user needs to register as an individual. This is done on the register.aspx page. In exercises 3 and 4 of this lab, you will add input validation controls to the register.aspx page to validate the input for the controls on this page.

Estimated time to complete this lab: 60 minutes

Exercise 1
Adding Web Controls to a Page

In this exercise, you will add three different controls to the ASP.NET page hotel_reservation.aspx: a **button** control, a **label** control, and a **drop-down list** control.

↙ To add a button

1. Open the file hotel_reservation.aspx in the folder InetPub\wwwRoot\ASPNET.
2. Locate the following comment:

```
<!-- TO DO: add a reservation button -->
```

3. Create a **button** Web control.
   a. Set the **ID**, **Text** and **Runat** attributes.
   b. Set the **OnClick** event to call the **cmdSubmit_Click** procedure.

   Your code should look like the following:

```
<asp:button
   id="cmdOK"
   Text="Reserve"
   OnClick="cmdSubmit_Click"
   runat="server"
/>
```

↙ To add a label

1. Locate the following comment:

```
<!-- TO DO: add a confirmation label -->
```

2. Create a **label** Web control.
   - Set the **ID** and **Runat** attributes.

   Your code should look like the following:

```
<asp:Label
   id="lblReservationConfirmation"
   runat="server"
/>
```

↙ To add a drop down list

1. Locate the following comment:

```
<!-- TO DO: add a second DropDownList -->
```

2. Create a second **drop-down list** control to contain the available room types.
   a. Set the **ID** attribute to **lstRoomType2**.
   b. Create two drop-down list items: **NoSmoking** and **Smoking**. Set the **No Smoking** item to be the default selection.

   Your code should look like the following:

```
<asp:DropDownList
    id="lstRoomType2"
    runat="server">
    <asp:ListItem selected="true">No Smoking</asp:ListItem>
    <asp:ListItem>Smoking</asp:ListItem>
</asp:DropDownList>
```

3. Save your work.

Exercise 2
Adding a Calendar Control to a Page

In this exercise, you will add a calendar control to the ASP.NET page hotel_reservation.aspx.

↙ To add a calendar control

1. In the file hotel_reservation.aspx, locate the following comment:

```
<!-- TO DO: add a calendar control for the ending date -->
```

2. Create a second calendar control, named **calEndingDate**, with the same attributes as the **calStartingDate** calendar control.

   Your code should look like the following:

```
<asp:Calendar id="calEndingDate" runat="server"
   BackColor="#9FC89F" ForeColor="#313179"
   BorderWidth="3"
   BorderStyle="Solid" BorderColor="#313179"
   CellSpacing="2" CellPadding="2"
   ShowGridLines="true"
   TitleStyle-BackColor="white"
   TitleStyle-ForeColor="black"
/>
```

↙ To validate start and end date selections

1. Locate the following comment:

```
'TO DO: validate start and end date selections
```

2. Write an event procedure named **cmdSubmit_Click** for the **Reserve** button that validates the selections in the two calendar controls.

   a. If the end date is greater than the start date, display a confirmation message in the **lblReservationConfirmation** label.

   b. If the end date is less than the start date, display an error message in the **lblReservationConfirmation** label.

Note You use the **SelectedDate** method to obtain the selected date of a calendar control and you use the **SelectedItem.Text** method to obtain the selected item of a drop-down list.

Your code should look like the following:

```
' Test if the date range is valid
If calEndingDate.SelectedDate <= _
   calStartingDate.SelectedDate Then
       lblReservationConfirmation.Text = _
       "The end date cannot be before the start date."
Else
       lblReservationConfirmation.Text = _
       "Congratulations. There is an available room " & _
       "for you.<br>" & _
       "Room type: " & _
       lstRoomType1.SelectedItem.Text & ", " & _
       lstRoomType2.SelectedItem.Text & "<br>" & _
       "from: " & _
       calStartingDate.SelectedDate.ToShortDateString() & _
       " to " & _
       calEndingDate.SelectedDate.ToShortDateString()
End If
```

↙ To save and test your work

1. Save your changes to the file hotel_reservation.aspx.
2. Using Internet Explorer, go to the hotel reservation page of the Conference Web site by viewing **http://localhost/ASPNET/hotel_reservation.aspx**.
3. Select a room type, a start date and an end date, and then click **Reserve**.

   You should see a confirmation message.
4. Select a start date that is later than the end date, and then click **Reserve**.

   You should see the following message: “The end date cannot be before the start date.”

Exercise 3
Validating User Input

In this exercise, you will add four different input validation controls to the ASP.NET page register.aspx: a **RequiredValidator** control, a **RangeValidator** control, a **RegularExpressionValidator** control, and a **CompareValidator** control.

↙ To validate the Full Name text box

1. Open the file register.aspx in the folder InetPub\wwwRoot\ASPNET.
2. Locate the following comment:

```
<!-- TO DO: validate txtFullName control-->
```

3. Create a **RequiredFieldValidator** input validation control to validate the **txtFullName** control.
   a. Set the **ID** and **Runat** attributes.
   b. Set the remaining attributes as shown in the following table.

Property	Value
ErrorMessage	" Full Name is a required field."
InitialValue	""
Display	Static

   c. Set the default text of the control to "*".

   Your code should look like the following:

```
<asp:RequiredFieldValidator
   id="valFullName"
   runat="server"
   controlToValidate="txtFullName"
   errorMessage="Full Name is a required field."
   initialValue=""
   display="static">
   *
</asp:RequiredFieldValidator>
```

↙ To validate the e-mail text box

1. Locate the following comment:

```
<!-- TO DO: validate txtEmail control -->
```

2. Create a **RegularExpressionValidator** input validation control to validate the **txtEmail** control. Set the **ID**, **ControlToValidate**, **Runat**, **ErrorMessage**, and **Display** attributes.
3. Set the **validationExpression** attribute of the **RegularExpressionValidator** control to a regular expression for the syntax of an e-mail address.

   Use the characters in the following table to create the regular expression.

Character	Meaning
.	Any character
+	One or more occurrences of the preceding expression
@	Character " @"
\.	Character " ."

   An e-mail address must meet the following requirements:

   - One or more characters representing the account name preceding the " at" (@) separator between the account name and the domain name address.
   - The "at" (@) separator.
   - One or more characters representing the domain name following the "at" (@) separator.
   - One period following the conclusion of the domain name.
   - One or more characters following the period, representing the top-level domain.

   Your code should look like the following:

```
<asp:RegularExpressionValidator
   id="valEmailExpr"
   runat="server"
   validationExpression=".+@.+\..+"
   controlToValidate="txtEmail"
   errorMessage="E-mail syntax is incorrect."
   display="dynamic">
   *
</asp:RegularExpressionValidator>
```

↙ To compare the values in the txtPassword and txtConfirmPassword text boxes

1. Locate the following comment

```
<!-- TO DO: password comparison -->
```

2. Create a **CompareValidator** input validation control to compare the **txtConfirmPassword** control with the **txtPassword** control.
3. Set the **ID**, **Runat**, **ControlToValidate**, **ControlToCompare**, **ErrorMessage**, and **Display** properties.

   Your code should look like the following:

```
<asp:CompareValidator
   id="valComparePassword"
   runat="server"
   controlToValidate="txtConfirmPassword"
   controlToCompare="txtPassword"
   errorMessage="Password fields do not match."
   Display="dynamic">
   *
</asp:CompareValidator>
```

↙ To save and test your work

1. Save your changes to the file register.aspx.
2. Using Internet Explorer, go to the registration page of the Conference Web site by viewing **http://localhost/ASPNET/register.aspx**.
3. Enter invalid information in the **Name**, **E-mail**, **Password**, and **Confirm Password** text boxes, and then click **Submit**. The following table gives examples of invalid information.

Field	Value
Full Name	“”
E-mail	“ Email”
Password	“abc”
Confirm Password	“def”

   The page should display an asterisk next to each invalid field.

4. Enter valid information in the **Name**, **E-mail**, **Password**, and **Confirm Password** text boxes, and then click **Submit**.

Exercise 4
Summarizing Validation Errors

In this exercise, you will add a **ValidationSummary** control to the register.aspx page. You will also programmatically test the validity of the controls on the form in the **OnClick** event procedure of the **Submit** button.

↙ To add a validation summary control

1. In the register.aspx file, locate the following comment:

```
<!-- TO DO: validation summary -->
```

2. Create a **ValidationSummary** control to summarize the validity of all validation controls on the form.

   Your code should look like the following:

```
<asp:validationsummary
   id="valSummary"
   runat="server" />
```

↙ To save and test your work

1. Save your changes to the register.aspx file.
2. Open the page in Internet Explorer and enter invalid values in some of the fields, and then click **Submit**.

   You will see an error message for each incorrect field in a summarization at the bottom of the page.

↙ To display another page if all the fields are valid

1. In the **cmdValidation_Click** event procedure, go to the comment:

```
'TO DO: test if the page is valid
```

2. Add code to test if the page is valid.
   a. If the page is valid, redirect to the validOK.aspx page using the **Redirect** method of the **Response** object.
   b. If the page is invalid, do nothing.

   Your code should look like the following:

```
If Page.IsValid Then
   Response.Redirect("validOK.aspx")
End If
```

↙ To save and test your work

1. Save your changes to the file register.aspx.
2. Open the page in Internet Explorer.
3. Enter invalid values in the fields, and then click **Submit**.

   You should see error messages.
4. Enter valid values in the fields, and then click **Submit**.

   You should receive the confirmation page.

Review

- What Are Web Controls?
- Using Intrinsic Controls
- Using Input Validation Controls
- Selecting Controls for Applications

---

1. What is the prefix tag that you must use to create Web controls?

2. How do you create an event procedure for a button?

3. List the input validation controls included with ASP.NET.

4. What property of the input validation controls determines the error text displayed to the user?

5. What validation control would you use to validate a phone number?

6. Why would you use the **ValidationSummary** control?