**Lecture 3**

# ASP.NET Web Controls - I

## Objectives

In this lecture you will learn the following

- Knowing about Web Controls
- Learning Web Controls and its hierarchy

# Coverage Plan

## Lecture 3

3.1 Snap Shot

3.2 Web Controls

3.3 Web Controls Hierarchy

3.4 Short Summary

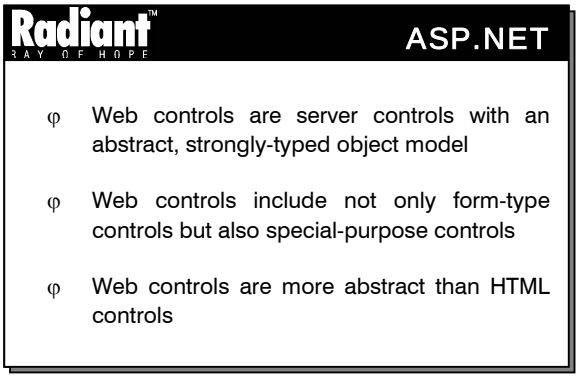
3.5 Brain Storm

## 3.1 Snap Shot

This session is designed to enlighten the readers about web controls, its properties and events. It throws light on the common properties shared by all the web controls. This session enables the learners to know about the properties and events of each control with an example in detail.

Web controls are Server-Side controls, which can be used in the same way as HTML controls. The only difference is that they must have the **runat = "server"** attribute set. This attribute makes the control available for server-side programming. Each ASP.NET Server Control is capable of exposing an object model containing properties, methods and events. This object model can be utilized by the ASP.NET developers to modify and interact with the Web page.

## 3.2 Web Controls




Web controls are nothing but the server controls with an abstract, strongly-typed object model. Web controls include not only form-type controls such as buttons and text boxes, but also special-purpose controls such as a calendar. Web controls are more abstract than HTML controls. In web controls, the object model does not necessarily reflect the HTML syntax. There are some common properties shared by all the Web controls which are as follows.

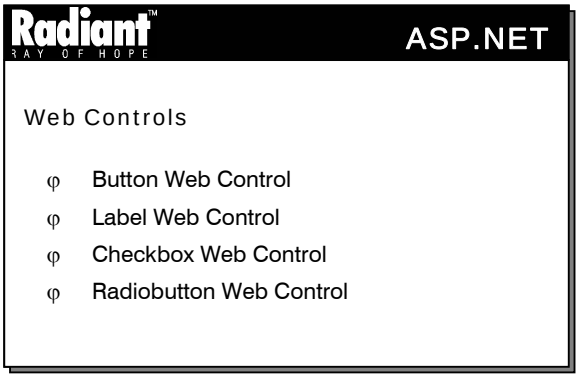
- **AccessKey** - This is the control's keyboard accelerator key. It specifies a single letter or number that the user should use while pressing ALT. For example, if the user wants to press ALT+K to access the control, "K" should be specified. Access keys are not supported on all browsers
- **Attributes -** Attributes denote the complete set for the control's persistent format. This property is only used while programming and cannot be set while declaring the control
- **BackColor -** This property indicates the color behind the text of the control
- **BorderWidth -** This indicates the size of the control's border in pixels
- **BorderStyle** - This indicates the border style of the control in pixels
- **CSS Class -** This class can be assigned to the control
- **CSS Style -** This is a collection of text attributes that will be rendered as a CSS style attribute on the outer tag of the control. This property is only used while programming and cannot be set while declaring the control
- **Enabled -** This property enables the control if set to true (the default) or disables when it is set to false

- **ForeColor -** ForeColor is the text color of the control. This property may not work in some of the earlier version browsers
- **TabIndex -** This property sets the order of the control when the user tabs between controls. If it is not set, the control's index is zero. Controls with the same tab index will be navigated in the order in which they are declared in the content
- **ToolTip -** ToolTip is indicated in the form of the text that appears when the mouse is placed on the control. ToolTip does not work in all browsers

## 3.3 Web Controls Hierarchy

All Web controls (except **Repeater)** derive directly or indirectly from the base class **System.Web.UI. WebControls.WebControl**. The controls shown on the left in the above figure map to HTML elements. The controls on the right provide data binding support. The controls in the middle are for validating form input. Additionally, in the center, are controls that provide rich functionality, such as the **Calendar** and the **AdRotator** controls.

It is possible to develop a custom Web control by extending an existing Web control, by combining existing Web controls, or by creating a control that derives from the base class **System.Web.UI.WebControls.WebControl.**




The following figure shows the Web Controls Hierarchy in the **System.Web.UI.WebControls** namespace.

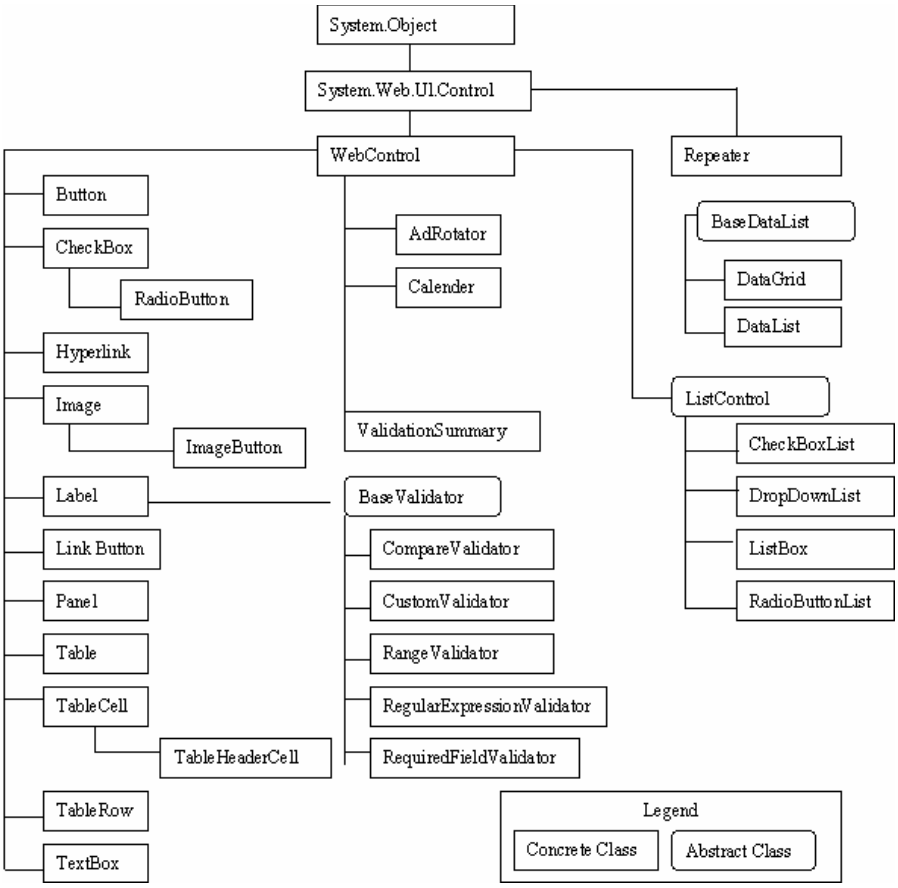



*Figure 3.1*

### Button

The **Button** Web control causes the Web Forms page to be posted back to the server.

### Syntax

```
<asp:Button runat="server"
   id="accessID"
   Text="label"
   Command="command"
   CommandArgument="commandArgument"
   OnClick="OnClickMethod"
/>
```

The properties of the button control are given below.

- **Command -** This property will be invoked when a button embedded in a container control is clicked
- **CommandArgument -** This is an optional command argument that is used in combination with the value of the Command property
- **Text -** This is the button caption

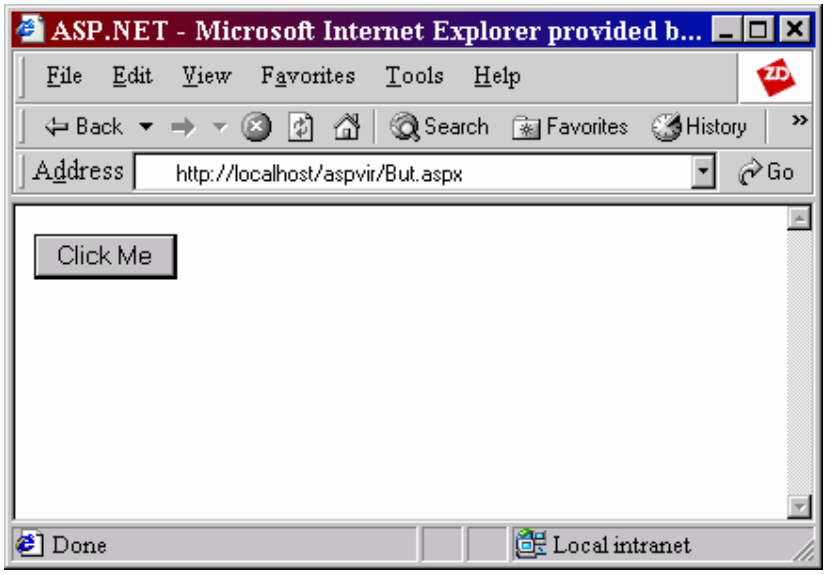
This control has only one event:

- **OnClick -** This event will be raised when the user clicks the button. This event always causes the page to be posted to the server

### Practice 3.1

The following example shows the code for displaying a button.

```
<HTML>
<TITLE> ASP.NET </TITLE>
<HEAD>
  <form runat="server">
     <p><asp:button id="Button1" text="Click Me"  runat="server" />
  </form>
</HEAD>
</HTML>
```




In the above example a button control is created. The id of the button is Button1 and the text "click me" is the caption that appears on the button.

### Label

This control allows to display static text to be displayed on the page and to manipulate it programmatically.

### Syntax

```
<asp:Label runat="server"
   id="accessID"
   Text="label"
/>
```

The main property of the Label control is given below:

- **Text** - This the text to be displayed on the label

### Textbox

This control generates single- and multi-line text boxes.

### Syntax

```
<asp:TextBox runat="server"
   id=accessID
   AutoPostBack="True|False"
   Colums="characters"
   MaxLength="characters"
   Rows="rows"
   Text="text"
   TextMode="Single | Multiline | Password"
   Wrap="True|False"
   OnTextChanged="OnTextChangedMethod"
>
</asp:TextBox>
```

The properties of this control are:

- **AutoPostBack -** If this property is used in the control, it will automatically cause a postback to the server when it is true; otherwise false. The default is false
- **Columns -** The column width of the control is in characters. This property differs from the Width, which sets the absolute width of the control independent of the character spacing
- **MaxLength** - The maximum number of characters are allowed within the text box. This property has no effect unless the TextMode property is set to SingleLine or Password
- **Rows -** Rows indicates the number of rows within the text box. This property has no effect unless the TextMode property is set to MultiLine
- **Text -** Implies the text that the user has entered into the box
- **TextMode -** Indicates whether the text box is in single-line, multi-line, or password mode. Possible values are Single, MultiLine and Password
- **Wrap -** Indicates whether text should wrap around as users type text into a multi line control. This property has no effect unless the TextMode property is set to Multi line

This Control has only one event

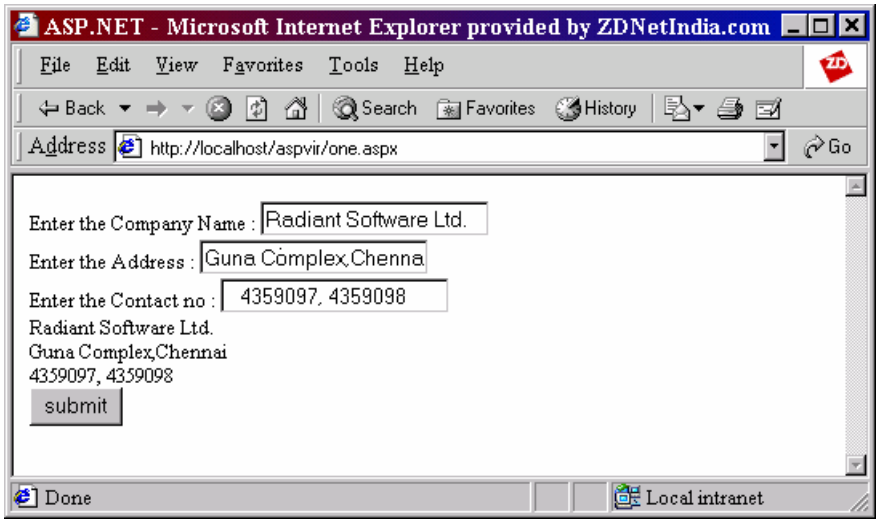
- **OnTextChanged -** This event is raised on the server when the contents of the text box change. This event does not cause the Web Forms page to be posted to the server unless the AutoPostBack property is set to true.

### Practice 3.2

The following example illustrates the usage of label, Textbox controls.

```
<html>
<title> ASP.NET </title>
```

```
<head>
<script language=c# runat=server>
 void button_click(Object sender,EventArgs e)
 {
     label1.Text=text1.Text;
     label2.Text=text2.Text;
     label3.Text=text3.Text;
 }
</script>
<body>
 <form runat=server>
     Enter the name : <asp:Textbox id=text1 runat=server/> <br>
     Enter the address : <asp:Textbox id=text2 runat=server/> <br>
   Enter the Contact no:  <asp:Textbox id=text3 runat=server/> <br>
     <asp:Label id="label1" runat=server/> <br>
     <asp:Label id="label2" runat=server/> <br>
     <asp:Label id="label3" runat=server/> <br>
     <asp:button OnClick="button_click" text="submit" runat=server/>
 </form>
</body>
</html>
```




In the above example, Textbox, label, button controls are used. It can be seen from the above example that the names entered in the textboxes are displayed in the label controls when the submit button is clicked.

### CheckBox

This control creates a check box on a Web Forms page, allowing users to set a true or false value for the item associated with the control.

### Syntax

```
<asp:CheckBox runat="server"
   id="accessID"
```

```
   AutoPostBack="True|False"
   Text="label"
   TextAlign="Right|Left"
   Checked="True|False"
   OnCheckedChanged="OnCheckedChangedMethod"
/>
```

The properties of the CheckBox are:

- **Checked -** This is true if the check box is checked, otherwise false. The default is false
- **TextAlign -** TextAlign is the position of the caption. The possible values are Right and Left. The default is Right
- **Text -** This is the check box caption

The CheckBox has the following event:

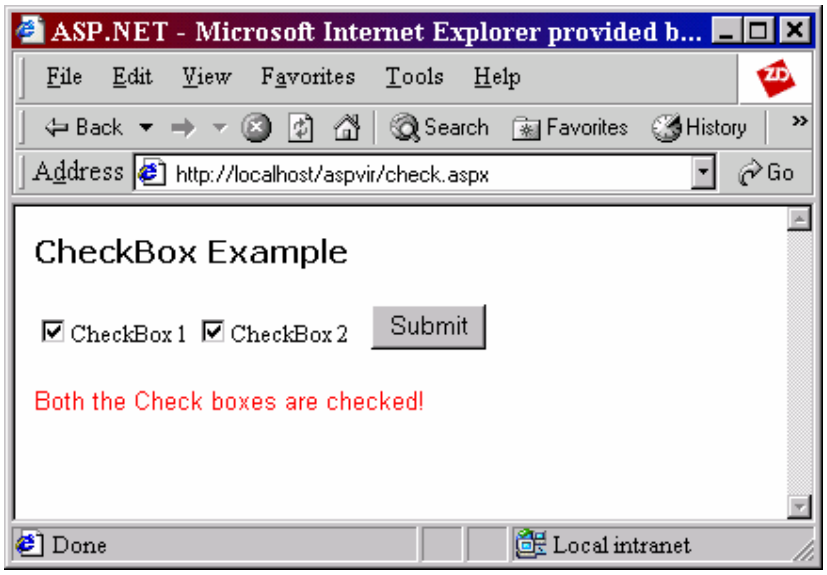
- **OnCheckedChanged -** This event is raised when the user clicks the checkbox. It does not cause the Web Forms page to be posted to the server unless the AutoPostBack property is set to true

## Practice 3.3

The following example illustrates the usage of the CheckBox Control.

```
<html>
<title> ASP.NET </title>
<head>
    <script language="C#" runat="server">
       void SubmitBtn_Click(Object Sender, EventArgs e) {
              if (Check1.Checked == true && Check2.Checked == true )
          {
             Label1.Text = "Both the Check boxes are checked!";
          }
         else if(Check1.Checked == false && Check2.Checked == true )
          {
             Label1.Text = "Check box 2 is  checked!";
          }
          else if(Check1.Checked == true && Check2.Checked == false )
          {
             Label1.Text = "Check box 1 is  checked!";
          }
          else
          {
             Label1.Text = "Both are not checked!";
          }
       }
    </script>
</head>
<body>
    <h3><font face="Verdana">CheckBox Example</font></h3>
    <form runat=server>
        <asp:CheckBox id=Check1 Text="CheckBox 1" runat="server" />
```

```
        <asp:CheckBox id=Check2 Text="CheckBox 2" runat="server" />
          
  <asp:button text="Submit" OnClick="SubmitBtn_Click" runat=server/>
        <p>
  <asp:Label id=Label1 font-name="arial" font-size="10pt" forecolor=red
                  runat="server"/>
    </form>
</body>
</html>
```




In the above example it can be seen that controls such as button, checkbox and labels are used. When the user checks the check box and clicks the "**submit**" button, a message is displayed in the label depending on the number of checks boxes that are checked.

### RadioButton

This control creates a single radio button on a Web Forms page.

### Syntax

```
<asp:RadioButton runat="server"
   id="accessID"
   AutoPostBack="True|False"
   Checked="True|False"
   GroupName="GroupName"
   Text="label"
   TextAlign="Right|Left"
   OnCheckedChanged="OnCheckedChangedMethod"
/>
```

The properties of the RadioButton are as follows:

- **AutoPostBack -** This returns true if client-side changes in the control automatically cause a postback to the server; otherwise false. The default is false
- **Checked -** This returns true if the radio button is checked, otherwise false. The default is false
- **GroupName -** The name of a group to which the radio button belongs. Radio buttons with the same group name are mutually exclusive
- **TextAlign -** This property indicates the position of the caption. The possible values are Right and Left. The default is Right
- **Text -** Text is the radio button caption

The RadioButton has the following event:

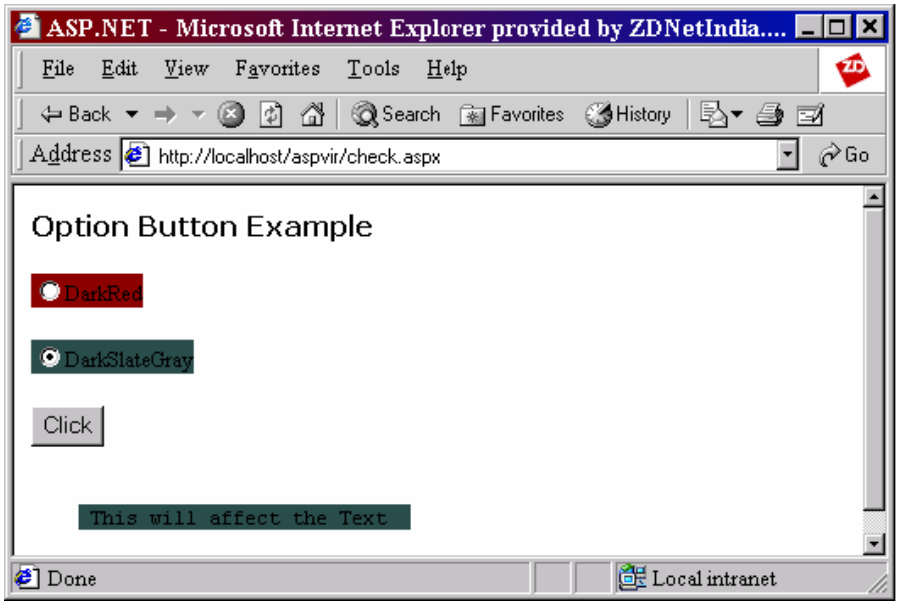
- **OnCheckedChanged -** This event will be raised when the user clicks the radio button and does not cause the Web Forms page to be posted to the server unless the AutoPostBack property is set to true

## Practice 3.4

The following example illustrates the usage of Radio Button.

```
<html>
<title> ASP.NET </title>
<head>
    <script language="C#" runat="server">
  void Button1_Click(Object sender, EventArgs e)
  {
    if (Radio1.GroupName=="Dark")
    {
      if (Radio1.Checked==true)
      {
          Span1.Style["background-color"] = Radio1.Text;
      }
      else if (Radio2.Checked==true)
      {
          Span1.Style["background-color"] = Radio2.Text;
      }
   }
      }
    </script>
</head>
<body>
    <h3><font face="Verdana">Option Button Example</font></h3>
    <form runat=server>
    <asp:RadioButton id="Radio1" GroupName="Dark"
           Text="DarkRed" BackColor="DarkRed" runat="server"/>
    <p>
    <asp:RadioButton id="Radio2" GroupName="Dark"
     Text="DarkSlateGray" BackColor="DarkSlateGray" runat="server"/>
    <p>
    <asp:Button id="Button1" OnClick="Button1_Click"
        Text="Click" runat="server"/>
  <pre>
    <span id=Span1 runat=server > This will affect the Text  </span>
</pre>
    </form>
```

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




This program uses two radio buttons and a button control. When the user selects one of the Options and clicks the "**Click**" button, the corresponding color will be the background color of the text area.

## 3.4 Short Summary

- Web controls are mkserver controls with an abstract, strongly-typed object model.
- Web controls are more abstract than HTML controls, in that their object model does not necessarily reflect in the HTML syntax.
- All Web controls (except Repeater) derive directly or indirectly from the base class System.Web.UI. WebControls.WebControl.
- The Button Web control causes the Web Forms page to be posted back to the server.
- Label Web control allows to display static text on the page and manipulate it programmatically.
- Check box control creates a check box on a Web Forms page, allowing users to set a true or false value for the item associated with the control.
- Radio button control creates a single radio button on a Web Forms page.

## 3.5 Brain Storm

1. What are Web controls?
2. What are the various Web controls available in ASP.NET?

**Lecture 4**

# ASP.NET Web Controls - II

## Objectives

In this lecture you will learn the following

+ Knowing about Image Controls & List Controls
+ Knowing about Link Button and Pannel
+ Knowing about AdRotator & Calendar

# Coverage Plan

## Lecture 4

4.1 Snap Shot

4.2 ASP.NET Image Controls

4.3 ASP.NET List Controls

4.4 Link Button

4.5 Panel

4.6 AdRotator

4.7 Calendar

4.8 Short Summary

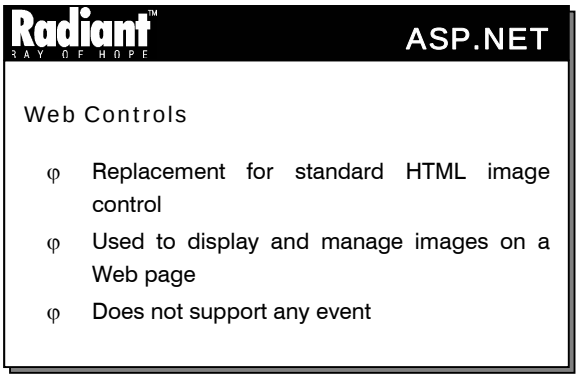
4.9 Brain Storm

## 4.1 Snap Shot

This session is aimed at making the learner familiar with advanced Web controls namely, ASP.NET Image controls, ASP.NET List controls, LinkButton, Panel, AdRotator and Calendar. This session also teaches how to apply styles to Web controls.

The ASP.NET list controls enable the users to choose one or more items from a list of items. The Panel is a container for other controls. The AdRotator control is used to display randomly selected advertisement banners on Web pages. The Calendar control displays a one-month calendar on the Web page, which can be used to view and select dates.

## 4.2 ASP.NET Image Controls




The **Image** control is a replacement for the standard HTML image control. It is used to display and manage images on a Web page. The syntax for using the Image control is:

```
<asp:Image id=imageId runat="server" ImageUrl=pathOfTheImage
AlternateText=text ImageAlign=alignment />
```

where *alignment* takes one of the values NotSet, AbsBottom, AbsMiddle, BaseLine, Bottom, Left, Middle, Right, TextTop and Top.

The properties of the Image control are listed below:

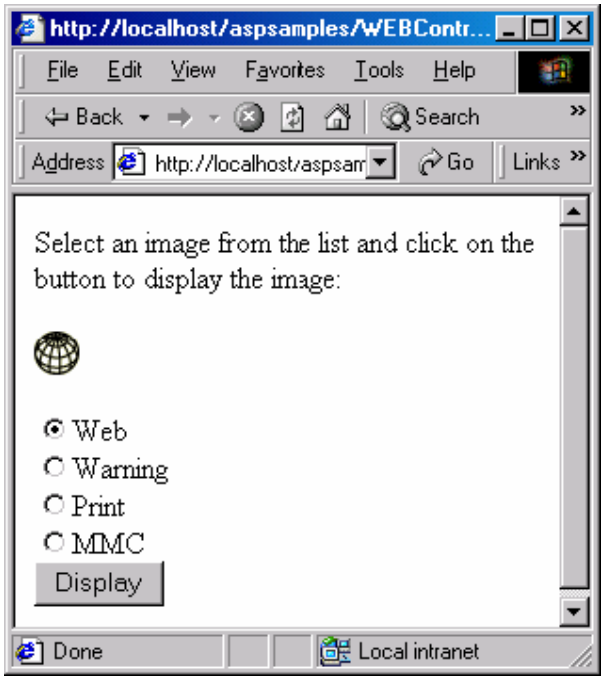
- **ImageUrl**- The ImageUrl property specifies the URL of the image to be loaded into the image control
- AlternateText - **The AlternateText property specifies the text to be displayed if the image referenced by the ImageUrl property is not available**
- **ImageAlign** - The ImageAlign property specifies the alignment of the image with the text surrounding it.

The Image control does not support any event

## Practice 4.1

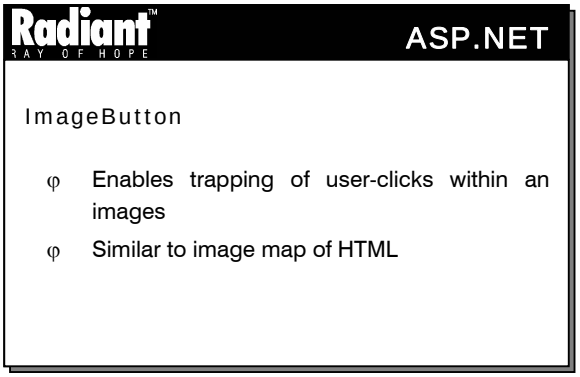
The following example displays one of the four images, based on the user's selection.

```
<html>
<head>
    <script language="C#" runat="server">
       void Btn_Click(Object sender, EventArgs e) {
            string s = "";
            if (Radio1.Checked)
               s = "web.gif";
            else if (Radio2.Checked)
               s = "warning.gif";
            else if (Radio3.Checked)
               s = "print.gif";
            else
               s = "mmc.gif";
            Img.ImageUrl="G:/Inetpub/wwwroot/" + s;
       }
    </script>
</head>

<body>
  <form runat=server>
 Select an image from the list and click on the button to display the image:
<br><br>

   <asp:Image ID="Img" ImageUrl="G:/Inetpub/wwwroot/web.gif"
            AlternateText="Sample Images" runat="server"/>
  <br><br>
  <asp:RadioButton id=Radio1 Text="Web" Checked="True" GroupName="Group1"
               runat="server" /><br>
<asp:RadioButton id=Radio2 Text="Warning" GroupName="Group1"runat="server"/>  <br>
<asp:RadioButton id=Radio3 Text="Print" GroupName="Group1" runat="server" />
  <br>
<asp:RadioButton id=Radio4 Text="MMC" GroupName="Group1" runat="server"/>
  <br>
  <asp:button text="Display" OnClick="Btn_Click" runat=server/>
 </form>

</body>
</html>
```




The example uses four radio buttons to enable the user to select his/her option. Since the default option is "Web", the corresponding image is displayed. When the user selects one of the four options and clicks the button "Display", the "OnClick" event of the button invokes the method "Btn_Click". This method checks which one among the four radio buttons is selected and based on this, it displays the corresponding image.

### ImageButton




The **ImageButton** control enables trapping of user-clicks within an image. This is similar to the image map of HTML. The syntax for using the ImageButton is:

```
<asp:ImageButton id=imageId runat="server" ImageUrl=pathOfTheImage
Command=cmdToInvoke CommandArgument=cmdargument OnClick=method />
```

The properties of the ImageButton are:

- **ImageUrl** - The ImageUrl property specifies the URL of the image to be displayed on the Web page

- **Command** - The Command property denotes the command to be executed when the button contained in a container control is clicked

- **CommandArgument** - The CommandArgument property is used along with the Command property if a value has to be supplied to the command invoked by that property

This control has only one event namely,

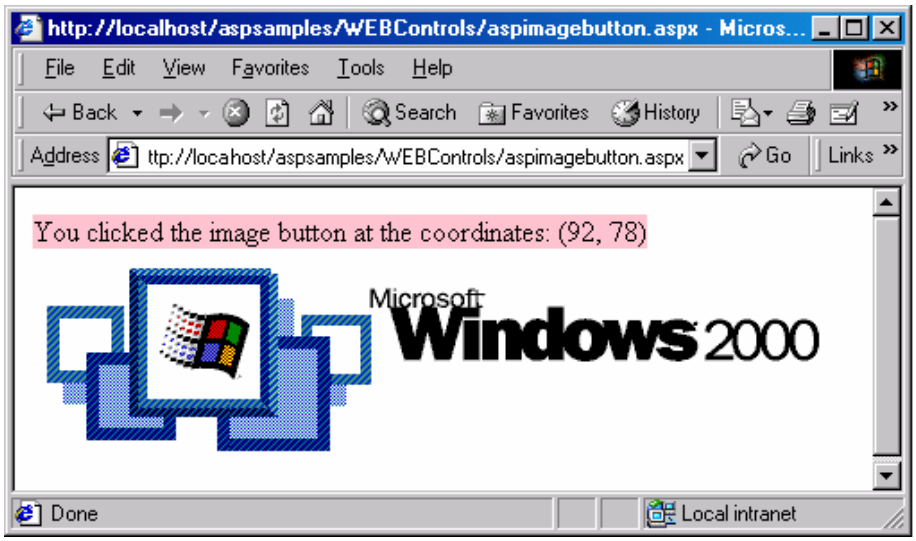
```
OnClick(Object sender, ImageClickEventArgs e)
```

that is raised when the user clicks the button and causes the page to be posted to the server. The second argument to the event specifies the position of the click.

## Practice 4.2

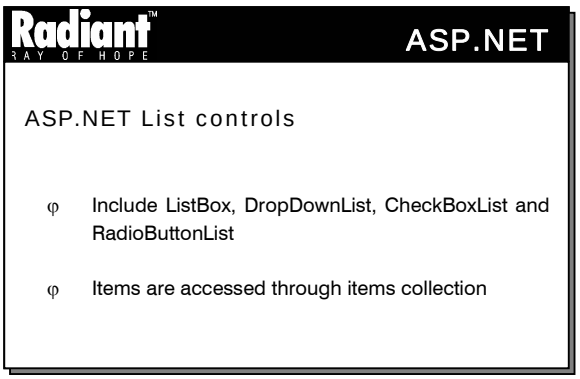
The following example displays an ImageButton and when the user clicks on the button, it displays the coordinates of the image at which the user has clicked.

```
<html>
<head>
   <script language="C#" runat="server">
      void ImgBtn_Click(object Source, ImageClickEventArgs e)
         {
     Label1.Text="You clicked the image button at the coordinates:
  (" + e.X.ToString() + ", " + e.Y.ToString() + ")";
         }
   </script>
</head>
<body>
    <form runat="server">
     <asp:label id="Label1" BackColor="Pink" runat="server"/>
     <asp:ImageButton id="imgbtn" runat="server"
        AlternateText="Windows 2000 Logo"
        ImageAlign="left"
               ImageUrl="g:/Inetpub/wwwroot/win2000.gif"
               OnClick="ImgBtn_Click"/>
        </form>
</body>
</html>
```




The example contains an **ImageButton** and a **Label**. When the user clicks on any part of the **ImageButton**, the **OnClick** event of the **ImageButton** is invoked. This event calls the method "ImgBtn_Click" that displays the message "You clicked the image button at the coordinates:" followed by the x and y coordinates of the point at which the user has clicked on the **ImageButton**.

## 4.3 ASP.NET List Controls




The ASP.NET List controls include ListBox, DropDownList, CheckBoxList and RadioButtonList. The items for the list controls are accessed through the **Items** collection. The **Items** collections can be used to add items, remove items, etc., from any control that uses the collection. Processes like adding items, determining the selection, setting the selection and responding to the changes are common to all the list controls. So, these processes will be explained after introducing these controls.

### ListBox

The **ListBox** control is used to allow the user to select one or more items from a specified list. It displays more than one item at the same time. The number of controls that should be visible can be set for the control. Each item of the list has the properties **Text**, **Value** and **Selected**. The **Text** property denotes the text to be displayed in the list and the **Value** property denotes the value associated with it. **Selected** takes a boolean value that denotes whether the item is selected or not. The following is the syntax for using the ListBox:

```
<asp:ListBox id=listboxid  runat="server"  DataSource=expression
DataTextField=source  DataValueField= source  AutoPostBack= value
Rows=numberOfRows  SelectionMode=mode  OnSelectedIndexChanged=method />
   <asp:Listitem value="value"  selected=value>  Text </asp:Listitem>
</asp:ListBox>
```

**where** AutoPostBack **and** selected **can take one of the values "true" or "false".**

The properties of the ListBox are as follows:

- **SelectionMode –** This property can either be "Single" or "Multiple" depending on whether the user wants to select a single item or multiple items. If it is set to "Single" then the SelectedItem property returns the selected item. If SelectionMode is set to "Multiple", then the SelectedItems property is used to retrieve the selected items
- **DataSource –** This property denotes the object that supports ICollection Interface
- **DataTextField -** The DataTextField property denotes the source of the Text property of the items
- **DataValueField –** The DataValueField property denotes the source of the Value property of the items
- **Rows** - The Rows property determines the number of rows to be visible in the list

The event **OnSelectedIndexChanged(Object sender, EventArgs e)** is raised on the server when the selection of the ListBox is changed.

### DropDownList

The **DropDownList** control enables users to select from a single-selection drop-down list. It is similar to the **ListBox** control, but it shows only the selected item. The syntax, properties and event for the **DropDownList** are similar to those of the **ListBox** except that **DropDownList** does not support multiple selection.

### CheckBoxList

The **CheckBoxList** control contains a group of checkbox controls. The syntax for using the CheckBoxList control is:

```
<asp:CheckBoxList  id=CheckBoxId  runat="server"  AutoPostBack=value
CellPadding= space  DataSource= expression  DataTextField=source
DataValueField=source  RepeatColumns=noOfCols  RepeatDirection=direction
RepeatLayout=layout  TextAlign=alignment  OnSelectedIndexChanged=method/>
   <asp:ListItem  text=caption  value=value  selected=value />
</asp:CheckBoxList>
```

In the above syntax, **AutoPostBack**, **DataSource**, **DataTextField**, **DataValueField**, **OnSelectedIndexChanged** and the properties of the list items are similar to those of the list box. The other properties of the CheckBoxList are:

- **CellPadding –** The CellPadding property determines the spacing between the controls

- **RepeatColumns –** The RepeatColumns property determines the number of columns in which the controls will be displayed
- **RepeatDirection** – The RepeatDirection property determines the direction in which the controls are to be displayed and can be either "Vertical" or "Horizontal"
- **RepeatLayout –** The RepeatLayout property determines whether the control renders in-line (when it takes the value "Flow") or in a table (when it takes the value "Table")
- **TextAlign –** TextAlign determines the position of the text with respect to the control and it can take one of the values "Right" or "Left". The default alignment is Right

### RadioButtonList

The **RadioButtonList** control provides a radio button group that can be dynamically generated through databinding. The radio buttons in the group are mutually exclusive, i.e., only one of them can be selected at a time.

```
<asp:RadioButtonList  id=buttonId runat="server"  AutoPostBack=value
CellPadding=space  DataSource=expression  DataTextField=source
DataValueField=source  RepeatColumns=noOfCols  RepeatDirection=direction
RepeatLayout=layout  TextAlign=align  OnSelectedIndexChanged=method />

   <asp:ListItem  text=caption value=value  selected=value />

</asp:RadioButtonList>
```

All the properties mentioned in the syntax are similar to those of the **CheckBoxList** control.

### Adding items to the ASP.NET List Control

Following are the steps to add an item programmatically to a list control:

- Create a new object of **ListItem** and set its **Text** and **Value** properties
- Add the new object to the Items collection through its **Add** method

For instance, to add a new item to a **ListBox** named **list1**, the following lines of code are written:

```
ListItem it = new ListItem();
it.Text = "New item";
it.Value = "10";
list1.Items.Add(it);
```

### Determining the selection in ASP.NET List Control

If the control allows single-selection, then the value of **SelectedIndex** will denote the index of the item that is selected. If nothing is selected then the value of SelectedIndex is –1. The contents of the selected item are retrieved using the SelectedItem property that returns an object of **ListItem**. The **Text** and **Value** properties of the **ListItem** are used to get the details about the item.

If the control allows multi-selection, then the **Selected** property of each item in the collection is checked to see if it is true. If it is true then its contents can be accessed through its **Text** and **Value** properties.

The following part of a code checks the items of a list box named **list1** to see if they are selected and displays the text of the selected items.

```
for (i = 0; i < list1.Items.count; i++)
   if(list1.Items[i].Selected == true)
      document.write(list1.items[i].Text + "  ");
```

### Setting the selection in ASP.NET List Control

To set a single selection, set the **SelectedIndex** property of the control to the index of the item to be selected. If the **SelectedIndex** property is set to –1 then no item is selected from the list. To set multiple selections, the **Selected** property of those items is set to **true**. It must be noted that multiple items can be selected only if the control's **SelectionMode** is set to **Multiple**.

The following line of code selects the second item in a list box named **list1**:

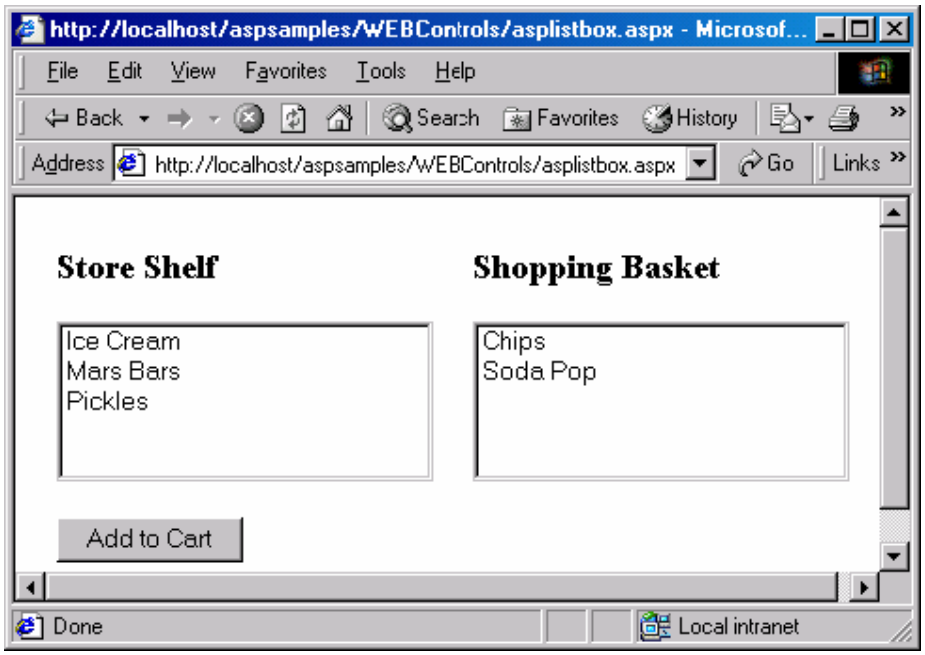
```
list1.SelectedIndex = 1;
```

### Practice 4.3

The following example moves items that are selected by the user from the "Store Shelf" ListBox to the "Shopping Basket" ListBox.

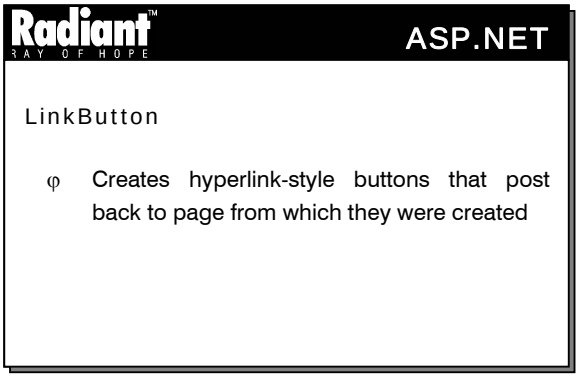
```
<html>
<head>
<Script language="c#" runat="Server">
  void addCart(Object sender,EventArgs e)
  {
     if (Store.SelectedIndex > -1)
        {
     shoppingBasket.Items.Add( new ListItem(Store.SelectedItem.Value ) );
           Store.Items.Remove( Store.SelectedItem.Value );
        }
  }
</Script>
</head>

<body>
<form runat="server">
   <table cellpadding="10">
     <tr>
         <td valign="top">
           <h3>Store Shelf</h3>
           <asp:listbox id="Store" width="200" runat="server">
              <asp:listitem>Ice Cream</asp:listitem>
                 <asp:listitem>Mars Bars</asp:listitem>
                    <asp:listitem>Chips</asp:listitem>
                       <asp:listitem>Soda Pop</asp:listitem>
                          <asp:listitem>Pickles</asp:listitem>
           </asp:listbox>
           <br>
  <asp:button id="add2Cart" text="Add to Cart"   OnClick="addCart"
                  runat="server"/>
        </td>

        <td valign="top">
           <h3>Shopping Basket</h3>
     <asp:listbox id="shoppingBasket" width="200" runat="server" />
        </td>
     </tr>
  </table>
```

```
</form>

</body>
</html>
```




The above example contains two **ListBoxes** "Store Shelf" and "Shopping Cart" and a **Button** "Add to Cart". When the user selects an item from the first **ListBox** and clicks on the **Button**, the **OnClick** event of the **Button** gets fired. This event calls the method "addCart". This method checks if any item is selected from the first ListBox by checking if its **SelectedIndex** is greater than –1. If any item has been selected in the **ListBox**, then the corresponding item is added to the second **ListBox** and then removed from the first.

## 4.4 LinkButton




The LinkButton control creates hyperlink-style buttons that post back to the page from which they were created. The syntax for using a link button is:

```
<asp:LinkButton id=buttonId runat="server" Text=caption Command=cmd
CommandArgument=argument OnClick=method />
```

The following are the properties of the LinkButton:

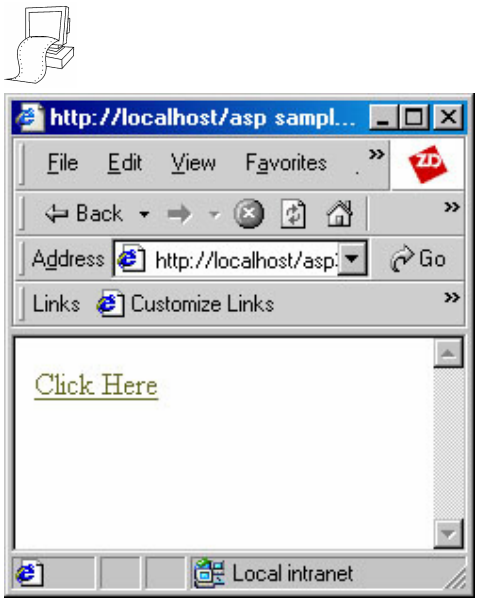
- **Text** – This property denotes the text that will be displayed in the link. It can also be specified outside the tag
- **Command** – The Command property denotes the command to be executed when the button contained in a container control is clicked
- **CommandArgument** – This property is used along with the Command property if a value has to be supplied to the command invoked by the Command property

The **`OnClick(Object sender, EventArgs e)`** event is raised when the user clicks the button and causes the page to be posted to the server.

### Practice 4.4

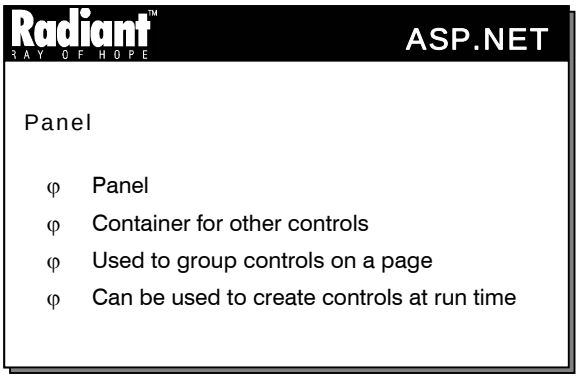
The following example uses LinkButton to enable the user to navigate to Microsoft's home page.

```
<html>
<head>
  <script language="C#" runat="server">
     void LinkBtn_Click(Object sender, EventArgs e) {
        Page.Navigate("http://www.microsoft.com");
     }
  </script>
</head>
<body>
  <form runat=server>
<asp:LinkButton Text="Click Here" onclick="LinkBtn_Click" runat="server"/>
  </form>
</body>
</html>
```




The example displays a **LinkButton** with the text "Click Here" displayed on the screen. When the user clicks on the **LinkButton**, the **OnClick** event of the **LinkButton** is invoked. This event calls the "LinkBtn_Click" method that directs the user to the home page of Microsoft.

## 4.5 Panel




The **Panel** is a container for other controls. It is used to group controls on a page. It enables the user to execute a command only on one group or certain groups of controls on a page. It can be used to create controls at run time.

```
<asp:Panel id=panelId runat="server" BackImageUrl=url HorizontalAlign=align
Wrap=value >
  ...
</asp:Panel>
```

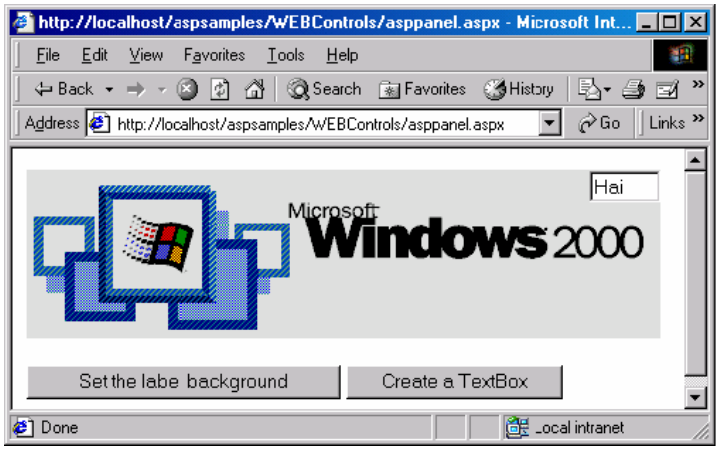
The following are the properties of the Panel control:

- **BackImageUrl –** The BackImageUrl property specifies the URL of the image to be displayed as a background of the panel
- **HorizontalAlign** – The HorizontalAlign property specifies the alignment of the panel with respect to the surrounding text. It can take one of the values "Center", "Justify", "Left", "NotSet" and "Right"
- **Wrap** - The **Wrap** property can be set to either "True" or "False". If it is set to "True" then the content can be wrapped. The default value is "True"

## Practice 4.5

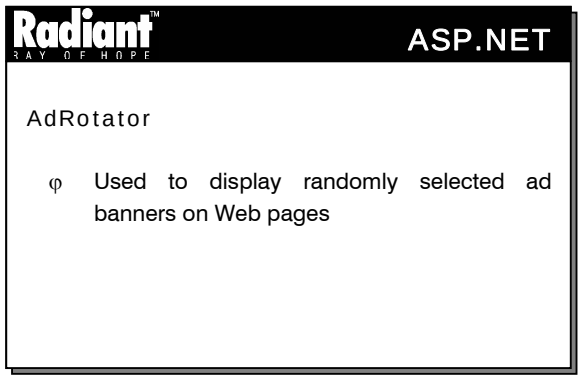
The following example sets the background for a panel and adds a textbox to it, when the user clicks the corresponding buttons.

```
<html>
<head>
  <script language="c#" runat="server">
     void Btn1_Click(Object sender, EventArgs e)
     {
               Panel1.BackImageUrl = "G:/Inetpub/wwwroot/win2000.gif";
     }
     void Btn2_Click(Object sender, EventArgs e)
     {
        TextBox t = new TextBox();
        t.ID = "Text1";
        t.Text="Hai";
        t.Width=50;
        Panel1.Controls.Add(t);
     }
  </script>
</head>
<body>
  <form runat=server>
<asp:Panel id="Panel1" Height=120 Width=450 BackColor="Gainsboro"
     Wrap="True" HorizontalAlign="Right" runat="server"/>
  <br>
<asp:Button id="Btn1" OnClick="Btn1_Click" Text="Set the label back
     ground" runat="server"/>
     <asp:Button id="Btn2" OnClick="Btn2_Click" Text="Create a
  TextBox" runat="server"/>
  </form>
</body>
</html>
```




The above example consists of a **Panel** and two **Buttons**. When the first Button, namely "Set the label background" is clicked, its **OnClick** event is invoked that calls the method "Btn1_Click". This method sets the **BackImageUrl** property of the **Panel** to display the "Windows 2000 logo". When the second Button, namely "Create a Text Box" is clicked, its **OnClick** event is invoked that calls the method "Btn2_Click". This method creates a new **TextBox** and places it in the right top of the **Panel**.

## 4.6 AdRotator




The **AdRotator** control is used to display randomly selected advertisement banners on Web pages.

```
<asp:AdRotator id=rotatorId runat="server" AdvertisementFile=file
KeywordFilter=filter Target=targetName OnAdCreated=method />

</asp:AdRotator>
```

The properties of the AdRotator are:

- **AdvertisementFile** - The AdvertisementFile property (a required property) specifies the path to the XML file containing the information about the advertisement
- **KeywordFilter –** This property indicates the category filter to pass to the source of advertisements
- **Target –** This property specifies the name of the browser in which the advertisement will be displayed

The only event of the AdRotator control is **OnAdCreated(Object sender, AdCreatedEventArgs e)**. This event is raised on the server once per round trip, after the control is created and before the page is rendered. When an AdvertisementFile is specified, this event is raised after an advertisement is selected from the XML file.

The source of the advertisement is specified in an XML file using the following syntax:

```
<Advertisements>
        <Ad>
               <ImageUrl> url </ImageUrl>
               <TargetUrl> url </TargetUrl>
               <AlternateText> toolTip </AlternateText>
               <Keyword>filter </Keyword>
               <Impressions> relativeWeight </Impressions>
        </Ad>
</Advertisements>
```

The properties of the element are:

- **ImageUrl** - The ImageUrl property contains the URL of the image file
- **TargetUrl** - The TargetUrl property contains the URL of the page that should be displayed when the user clicks on the ad
- **AlternateText** - The AlternateText property contains the text that will be displayed if the image is not loaded
- **Keyword** - The Keyword property specifies the category for the advertisement
- **Impressions** - The Impressions property denotes the weight of the advertisement with respect to the other ads in the file. It determines how often the advertisement will be displayed. The higher the value of Impressions, the more the importance given to the advertisement

## Practice 4.6

The following example uses AdRotator to display two advertisements. When the user clicks on an advertisement, he/she is directed to the corresponding Web page.

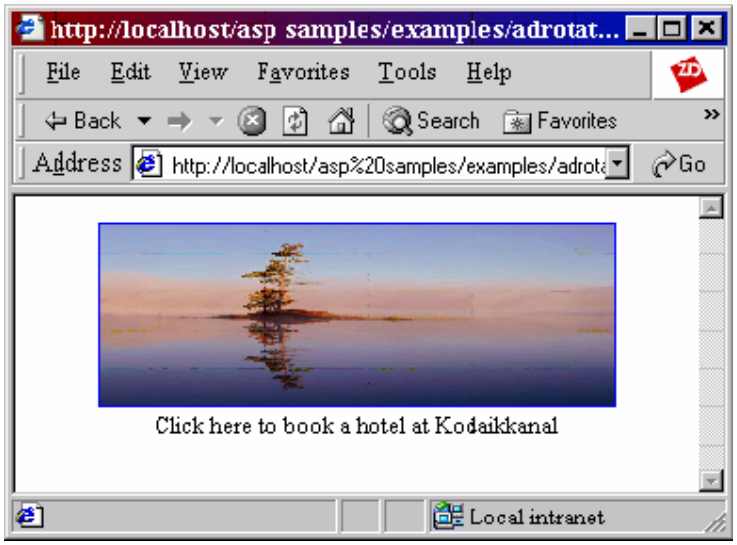
**The XML file:**

```
<Advertisements>
  <Ad>
     <ImageUrl>nat1.bmp</ImageUrl>
     <NavigateUrl>http://www.kodaihotels.com</NavigateUrl>
      <AlternateText> Click here to book a hotel at Kodaikkanal</AlternateText>
     <Keyword>websites </Keyword>
     <Impressions>10</Impressions>
```

```
  </Ad>
  <Ad>
     <ImageUrl>nat2.bmp</ImageUrl>
     <NavigateUrl>http://www.udagamandal.com</NavigateUrl>
        <AlternateText> Click here to book a hotel at Ooty </AlternateText>
     <Keyword>websites </Keyword>
     <Impressions>10</Impressions>
  </Ad>

</Advertisements>
```

**The ASPX file:**

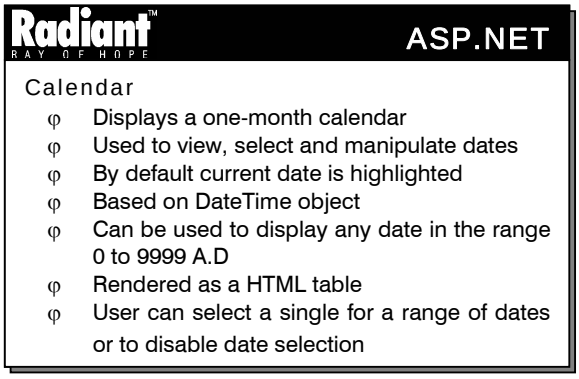
```
<html>
<head>
  <script language="c#" runat="server">
     void Ready(Object sender, AdCreatedEventArgs e) {
               label1.Text= e.AlternateText ;
     }
  </script>
</head>
<body>
  <Center>
     <asp:AdRotator Width = "80%" Height=100 id=a1
                  AdvertisementFile="adrot.xml" BorderWidth=1
               OnAdCreated = "Ready" runat=server />
     <br>
        <asp:label id="label1" runat="server"/>
  </Center>
</body>
</html>
```




The example consists of an **AdRotator** control that is centered in the form using the **<Center>** tag. The **Width** of the **AdRotator** is set to be 80% of the width of the page. The **AdvertisementFile** property

specifies the XML file that has to be referred by the ASPX file in order to display the advertisements. The XML file specifies the image to be used, its alternate text, etc.

## 4.7 Calendar




The **Calendar** control displays a one-month calendar on the Web page. It can be used to view and select dates. By default the current date is highlighted. The Calendar control can be used to display details of each day, like the daily schedule, appointments, etc. It is based on the DateTime object of the .NET framework and can be used to display any date in the range between 0 to 9999 A.D. It is actually rendered as a HTML table on the Web page. Therefore, its properties are similar to that of the table.

The user can change the current date, navigate to another month or year, and select a single date, week or month. The user can also select a range of dates using this control. By default, the control allows the user to select a single date. The settings can also be changed to disable the selection of the date. If day selection is enabled, each date acts as a LinkButton and raises an event when clicked. If the week or month selection is enabled, a column of links is added to the left side of the Calendar so that the user can select an entire week or month. The appearance of the Calendar control like its color, font, borders etc., can be modified using the style objects.

The properties of the Calendar object are listed in Table 4.1

| **Property** | **Description** |
|---|---|
| CellPadding | Determines the space between cells in pixels. |
| CellSpacing | Determines the space between the border of a cell and the text contained in the cell. |
| DayNameFormat | Determines the format of the name of the days. The value can be one of “Full”, “Short”, “FirstLetter” or “FirstTwoLetters”. The default is “Short”. |
| FirstDayOfWeek | Determines the day that will be displayed in the first column. The week will be set to start from this day. The default value depends on the local settings of the server. |
| NextMonthText | Determines the text that will be displayed in the hyperlink for the next month. This requires the **ShowNextPrevMonth** property to be set to true. |

| | |
|---|---|
| NextPrevFormat | Specifies the format in which the next month and previous month links are displayed. The default value is "Custom". |
| PrevMonthText | Determines the text that will be displayed in the hyperlink for the previous month. This requires the **ShowNextPrevMonth** property to be set to true. |
| SelectedDate | Specifies the date that will be highlighted. The default value is the value of the **TodaysDate** property. |
| SelectedDates | A collection of the dates highlighted in the Calendar. It is a read-only property. |
| SelectionMode | Specifies whether the user can select a day, week or month. It can take one of the values "None", "Date", "DateWeek" or "DateWeekMonth". The default value is "Date". Setting this property to "None" disables date selection. |
| SelectMonthText | Represents the text to be displayed for month selection in the selector column if **SelectionMode** is set to "DayWeekMonth". |
| SelectWeekText | Represents the text to be displayed for week selection in the selector column if **SelectionMode** is set to "DayWeek" or "DayWeekMonth". |
| ShowDayHeader | Determines whether or not the names of the days of the week are displayed. |
| ShowGridLines | Specifies whether or not the border is displayed around each day. |
| ShowNextPrevMonth | Specifies whether or not next and previous month hyperlinks are displayed. |
| ShowTitle | Specifies whether or not the title is displayed. |
| TitleFormat | Specifies the format of the month in the title bar. The default value is "Month". |
| TodaysDate | Determines the current date. |
| VisibleDate | Specifies the month to be displayed. The date can be any day within the month. |

***Table 4.1***

The events of the Calendar control are raised on the server. They are:

- **OnDayRender(Object sender, DayRenderEventArgs e) –** Raised when the cell for each day is created
- **OnVisibleMonthChanged(Object sender, MonthChangedEventArgs e) –** Raised when the hyperlink for the next month or the previous month is clicked
- **OnSelectionChanged(Object sender, EventArgs e) –** Raised when the SelectionMode is changed. This event causes the page to be posted to the server

**Practice 4.7**

Following is an example using the Calendar control.

```
<form runat = “server”>
<asp:Calendar id=Calendar2 runat=”server”

backcolor=”lightgreen”
bordercolor=”red”
borderwidth=10
borderstyle=”inset”

cellpadding=2
cellspacing=5
DayNameFormat=”firstletter”
DayHeaderStyle-Font-Bold=”True”
Dayheaderstyle-font-italic=”true”
Dayheaderstyle-font-oblique=”true”
Daystyle-backcolor=”white”

Firstdayofweek=”sunday”
Font-Name=”Verdana;Arial” Font-Size=”12px”
Forecolor=”red”

Height=”280px”

Nextmonthtext=”>”
NextPrevFormat=”fullmonth”
NextPrevStyle-ForeColor=”white”
NextPrevStyle-Font-Size=”10px”
OtherMonthDayStyle-ForeColor=”green”
SelectionMode=”DayWeekMonth”
SelectedDayStyle-BackColor=”#ffcc66"
SelectedDayStyle-Font-Bold=”True”
SelectorStyle-BackColor=”#99ccff”
SelectorStyle-ForeColor=”Olive”
SelectorStyle-Font-Size=”9px”
SelectWeekText = “Week”
SelectMonthText = “Month”
TodayDayStyle-Font-Bold=”True”
TodayDayStyle-ForeColor=”blue”
TitleFormat=”Monthyear”
TitleStyle-BackColor=”blue”
TitleStyle-ForeColor=”white”
TitleStyle-Font-Bold=”True”

Width=”500px”
/>
</form>
```

| April | May 2001 | | | | | | June |
|---|---|---|---|---|---|---|---|
| Month | S | M | T | W | T | F | S |
| Week | 29 | 30 | 1 | 2 | 3 | 4 | 5 |
| Week | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Week | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| Week | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| Week | 27 | 28 | 29 | 30 | 31 | 1 | 2 |
| Week | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

An important point to be noted in the above example is that the Calendar control is enclosed within the Form tag. This enables the navigation between months. If the control is not embedded in the form, it will not be submitted to the server and thus will not support the navigation between months. The example illustrates how the control can be customized by setting many of its properties.

## 4.8 Short Summary

- The Image control is a replacement for the standard HTML image control. It is used to display and manage images on a Web page
- The ImageButton control enables trapping of user-clicks within an image
- The ASP.NET List controls include ListBox, DropDownList, CheckBoxList and RadioButtonList. The items for the list controls are accessed through the Items collection
- The LinkButton control creates hyperlink-style buttons that post back to the page from which they were created
- The Panel is a container for other controls. It is used to group controls on a page and perform some action only to one group or certain groups of controls on a page
- The AdRotator control is used to display randomly selected advertisement banners on Web pages
- The Calendar control displays a one-month calendar on the Web page. It can be used to view and select dates
- The Web controls provide additional support for styles by adding properties for commonly used style settings

## 4.9 Brain Storm

1. Differentiate between Image and ImageButton controls.
2. Differentiate between ListBox and DropDownList controls.
3. What is the use of the CheckBoxList control?
4. How is an advertisement file specified in an AdRotator?
5. Which property of the AdRotator is used to specify the importance of an advertisement with respect to other advertisements?
6. Which property of the Calendar control is used to set the Text for the hyperlink for the next month?
7. Can styles be applied to ASP Web Controls?

**Lecture 5**

# Validation Controls

## Objectives

In this lecture you will learn the following

- Knowing about the Validation in ASP.NET
- What is meant by Validation Summary Controls?

# Coverage Plan

## Lecture 5

5.1 Snap Shot

5.2 Validation ASP.NET

5.3 Required Filed Validation

5.4 Compare Validator

5.5 Range Validator

5.6 Regular Expression Validation

5.7 Custom Validator

5.8 Validation Summary Controls

5.9 Short Summary

5.10 Brain Storm

## 5.1 Snap Shot

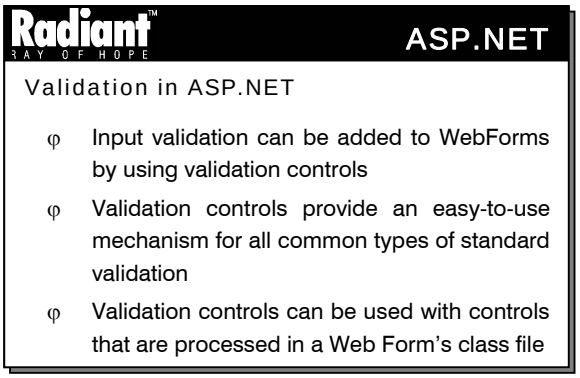
This session focusses on how input validation can be added to WebForms by means of validation controls. It also discusses the various validation controls available in ASP.NET. An important aspect of creating Web Forms pages for user is to enable Validation of user-input. The Web Forms framework provides a set of validation controls that provide an easy-to-use but powerful way to check for errors, and if necessary, display messages to the user. This session introduces to the validation controls in broader prospective.

Let us consider a typical web-based form wherein users have to provide their names and ages. A visitor to the page types his name properly, but say types in "twenty-four" for his age instead of the number "24". When the user presses the Submit button on the form, his name and age ("twenty-four") are sent to the Web server, where there is a program waiting for his input. The program expects to get proper data from the user to add it to the database. But instead, it gets a string of alphabetic characters for the age. So the Web server creates an HTML page that asks the user to enter a valid age and sends this page to the user's browser. The user might complete the field correctly the second time and send the data back to the server.

This whole process may take just half a second, or several seconds, depending on several factors such as the speed of the user's connection, the speed of the Net, the Web server's load, and so on. These trips to the server for simple validations are a waste of resources.

Moving these simple validation tasks from the server to the client is a simple task and a big advantage of client-side scripting. If the above Web page had a simple validation script, then the user would be immediately notified of his data entry error. More importantly, the page would not have been sent to the server until it was validated and complete.

## 5.2 Validation in ASP.NET




Validating user input can be a time-consuming and tedious process. ASP.NET provides developers with a set of pre-built validation controls that will help in speeding up and simplifying the task of validation. This session provides a brief overview of using the Validation controls.

Input validation can be added to WebForms by using validation controls. Validation controls provide an easy-to-use mechanism for all common types of standard validations – (eg. testing for valid dates or values within a range.) Moreover, these controls can be used to provide custom-written validations. In addition, validation controls allow to completely customize the display of error information to the user.

Validation controls can be used with any control that is processed in a Web Form's class file, including both HTML and ASP.NET server controls.

### Using Validation Controls

User input validation can be enabled by adding validation controls to the form. There are controls for different types of validation, such as range-checking or pattern-matching.

Each validation control references an input control (a server control) elsewhere on the page. When the user's input is being processed (for example, when the form is submitted), the page framework passes the user's entry to the appropriate validation control or controls. The validation controls test the user's input and set a property to indicate whether the entry has passed the test or not. After all validation controls have been called, a property on the page is set; if any of the controls show that a validation check has failed, the entire page is set to invalid.

The state of the page and that of the individual controls can be tested by the code. For example, the state of the validation controls have to be tested before updating a data record with information entered by the user. If an invalid state is detected, the the update has to be bypassed. Typically, if any validation check fails, all the processing is skipped and the page is returned to the user. Validation controls detect errors and then produce an error message that appears on the page.

### Validating for Multiple Conditions

Generally speaking, each validation control performs only one test. (eg. to establish that a field is required and limited to accepting dates within a specific range.) More than one validation control can be attached to an input field on a form. In that case, the tests performed by the controls are resolved using a logical AND. The data input by the user must pass all the tests in order to be considered as valid.

In some instances, several different entries might be valid. For example, if the code prompts for a phone number, the users may be allowed to enter a local number, a long-distance number, or an international number. This situation arises primarily where checking for specific patterns of numbers or characters. To perform this type of test, the pattern-matching validation control is used to specify the multiple valid patterns within the control.

### Displaying Error Information

Validation controls are normally not visible in the rendered form. However, if the control detects an error, it produces error message text. The error message can be displayed in a variety of ways, as listed in the following table.

| Display option | Description |
|---|---|
| In place | Each validation control can individually display an error message or appear in place (usually next to the control where the error occurred). |
| Summary | Validation errors can be collected and displayed in one place, for example, at the top of the page. (This strategy is often used in combination with displaying the error message next to the input fields with errors.) If the user is working in a browser that supports DHTML, the summary can be displayed in a pop-up message box. |
| Custom | Create custom error display by capturing the error information and designing user-defined output. |

**Note :** If in-place or summary display options are used, the error message text can be formatted using HTML.

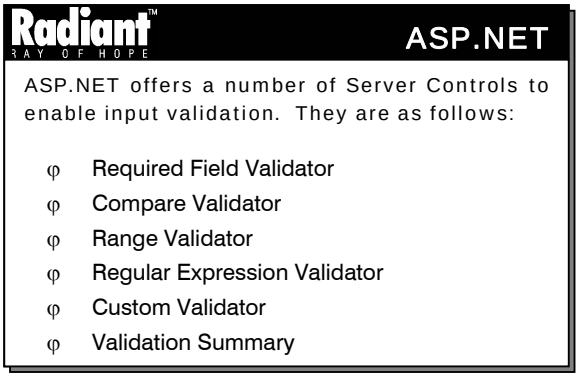
### Server-Side and Client-Side Validation

Validation controls perform input checking in server code. When the user submits a form to the server, the validation controls are invoked to review the user's input, control by control. If an error occurrs in any of the input controls, the page itself is set to an invalid state. Hence, validity has to be tested before the code runs.

If the user is working with a browser that supports DHTML, the validation controls can also perform validation using client script. This can substantially improve response time in the page Errors are detected immediately and error messages are displayed as soon as the user leaves the control containing the error. If client-side validation is available, the user has greater control over the layout of error messages and can display an error summary in a pop-up message box.

The page framework performs validation on the server even if the validation controls have already performed it on the client, so that the test for validity can be within the server-based event-handling methods. In addition, it helps prevent users from being able to bypass validation by impersonating as another user or a pre-approved transaction.

### Types of Validation




Each of these controls offers its own type of validation.

## 5.3 RequiredFieldValidator

The RequiredField Validator is used to ensure that the user does not leave a field blank. For fields where it does not matter as to what type of data the user enters, as long as they enter something, this is the validator to use. This is actually one of the most widely used validation controls. This validation ensures that the user does not skip an entry.

### Automatic Client-side or Server-side Implementation

One of the dangers of using Client-side validation is that the users could create a page of their own that does not perform validation. When they submit this to the server, it could provide invalid values. To prevent this, the validation controls that are supplied with ASP.NET automatically perform validation of posted values on the server, even if the original validation is done on the client.

Validation can be managed by a Page directive. This overrides the automatic client-side validation process. The directive:

```
<%@ Page ClientTarget="DownLevel"%>
```

forces the validation controls to only do validation on the server, while the directive:

```
<%@ Page ClientTarget="UpLevel"%>
```

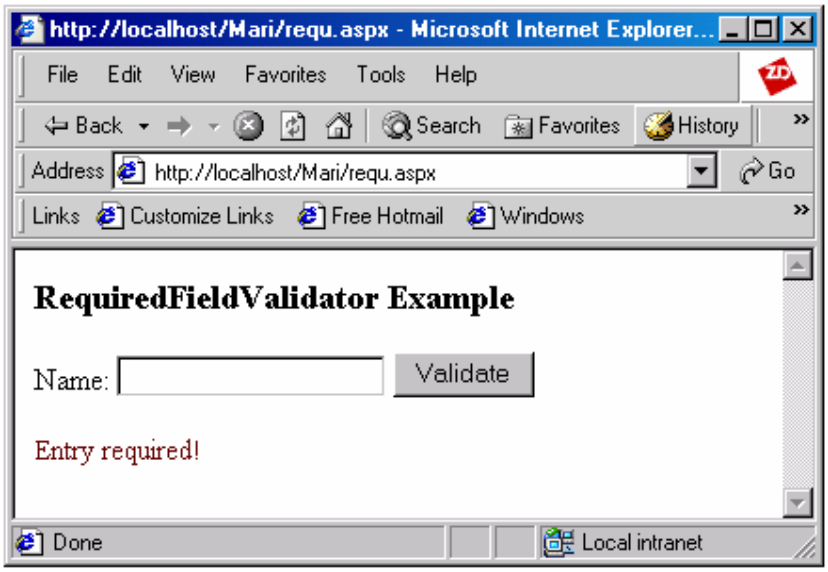
forces the controls to use Client-side validation as well as the Server-side validation of posted values.

**Note :** "UpLevel" directive will force the client to attempt to use Client-side script for the validation – even if it doesn't support this technique.

## Practice 5.1

Here is an example using RequiredField Validator, a name is required from the user as input.

```
<html>
<body>
<h3>RequiredFieldValidator Example</h3>
<form runat=server>
   Name: <asp:textbox id=text1 runat="server" />
   <asp:button id="Button1" runat="server" text="Validate" />
   <p><asp:requiredfieldvalidator id="RequiredFieldValidator1"
      controltovalidate="Text1"
      font-size="11pt"
      forecolor="#600" runat="server">
         Entry required!
   </asp:requiredfieldvalidator>
</form>
</body>
</html>
```




The RequiredFieldValidator checks if the user has made any entry. It ensures that the user is not allowed to move further without entering the field. So the attribute control to Validate is set to text1. When the

control shifts from text1, the RequiredFieldValidator checks if there is any entry for the field and if there is no entry it prints the error message "Entry Required!".

## 5.4 CompareValidator

The Compare Validator is used to compare the data entered by a user with a constant or other field on the web form. The most common use of this Validation control is with respect to the password field. Typically Web Forms would ask the user to enter a password and to confirm it . These two fields must be identical for the form to be processed. The Compare Validator would be used in this case.

### Display Property

The display property indicates how the validation controls output will be rendered in the page. The value "static" tells the control to reserve sufficient space in the page for the text content so that the page layout does not change when this content is displayed.

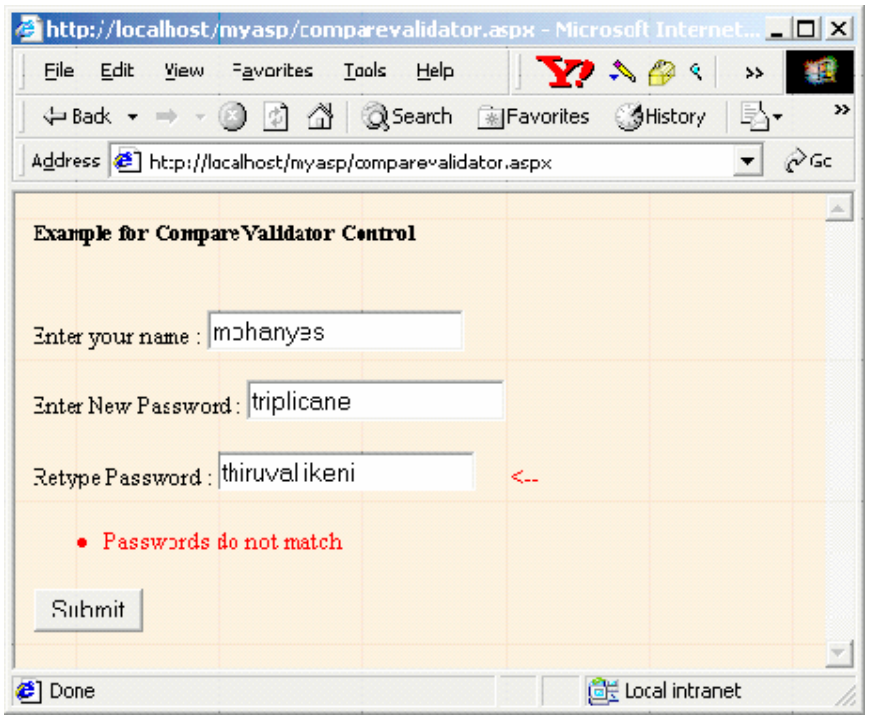
The value "dynamic" indicates that the control will only take up space in the page when the error text string is displayed. In this case, the layout of the page might change. However, this is useful when there are more than one validation control is attached to a control on the form. The value "none" tells the control that no inline output will be displayed, even if the input value is invalid. Instead, the ValidationSummary control can be used to detect if there was an error, and display the contents of the errorMessage property elsewhere in the page.

### Practice 5.2

Given below is a typical example of using the compare validator control. Two text controls are used to get the password from the user and if there is a difference in the password it is reported by the comparison control.

```
<HTML>
    <SCRIPT LANGUAGE="c#"  RUNAT="server">
    void submit_Me(Object sender,EventArgs E)
    {
        if (Page.IsValid==true)
        {
            Result.Text="Successful Login.";
        }
    }
    </SCRIPT>

    <BODY bgcolor="wheat">
    <FORM RUNAT="server">
        <h4>Example for CompareValidator Control</h4>
    <br>
    <asp:label id=Result runat="server"/>    Enter your name :
    <INPUT TYPE="text" id="UserName" Runat="server" />
        <asp:RequiredFieldValidator id="regUserName"
        runat="server"
        ControlToValidate="UserName"
        errormessage="Please Enter User Name"
    display="static">
    </asp:RequiredFieldValidator>
```

```
    <br><br>
    Enter New Password :
    <INPUT TYPE=”text” id=”Password” Runat=”server” />

    <asp:RequiredFieldValidator id=”reqPassword”
        runat=”server”
        ControlToValidate=”Password”
        errormessage=”Please Enter Password”
        display=”static”>
    <--
    </asp:RequiredFieldValidator>

    <br><br>
    Retype Password :
    <INPUT TYPE=”text” id=”vPassword” Runat=”server” />
    <asp:RequiredFieldValidator id=”reqvPassword”
        runat=”server”
        ControlToValidate=”vPassword”
        errormessage=”Please Verify Password”
    display=”static”>
       <--
    </asp:RequiredFieldValidator>

    <asp:CompareValidator id=”reqMatch”
        runat=”server”
        ControlToValidate=”Password”
        errormessage= “Passwords do not match”
        ControlToCompare = “vPassword”
        type = “string”
        operator = “Equal”
    display=”static”>
    <--
        </asp:CompareValidator>

    <asp:ValidationSummary id=”sumErrors”
        runat=”server”
        showSummary = true
    displayMode = “BulletList” />
    <asp:button text=”Submit” RUNAT=”server” onClick=”submit_Me” />
    </FORM>
    </BODY>
</HTML>
```




In the above program, three text boxes are created for Name, New Password and Retype password. The New password is for entering the new password and the retype password is used to check if the password has been typed without typographical errors. Normally, the password will be masked and it will not be visible. Here the password has not been masked to enable us to view the text.

The RequiredFieldValidator is used to check if the username, password and vpassword have any entry. If the user leaves them blank it shows the user that some entry is required for the field. The CompareValidator which checks if both the password and vpassword entries are identical. If they are not identical, the error message "Passwords do not match is displayed" otherwise "Successful Login" is displayed.

## 5.5 RangeValidator

The Range Validator checks if the data that a user enters falls within a given range of values. The minimum and maximum values can be set to constant values or they can be set to other fields on the web form.

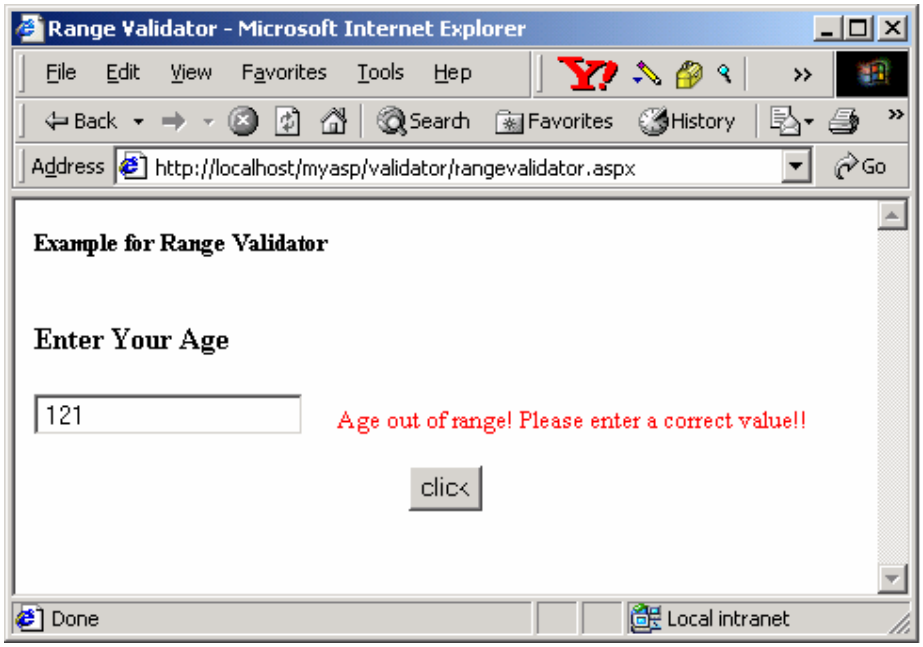
### Practice 5.3

Now let us see an example usng RangeValidiator which checks whether the given input is within the specified range.

```
<html>
<script language="c#" runat="server">
void btnclick(object sender,EventArgs e)
{
        if (Page.IsValid==true)
```

```
        {
            Result.Text=”Successful Entry.”;
        }
}
</script>
<title> Range Validator</title>
<body>
<form runat=”server”>
<h4>Example for Range Validator</h4>
<br>
<h3>Enter Your Age</h3>
<asp:textbox id=”textval” text=” “ runat=”server”/>
    
<asp:label id=Result runat=”server”/>
<asp:rangevalidator id=”checkrange”
type=”integer”
controltovalidate=”textval”
minimumvalue=1
maximumvalue=100
errormessage=”Age out of range! Please enter a correct value!!”
runat=”server”>
</asp:rangevalidator>
<center><table tablecolor=”#eeeeee” >
</table></center>
<br>
<center><asp:button id=”checkbtn” text=”click” runat=”server”
Onclick=”btnclick” /></center>
</form>
</body>
</html>
```

In the below example a text box control is used to get the age of the user. RangeValidator is used to check if the age falls within a range. So the attribute control to validate is set to textval. When the control shifts from the textval the RangeValidator checks for the range and if there is an error, prints the error message even before the user clicks the click button. The color of the error message is set to red to enable the user easily notice the error message. When the button is clicked it checks for the validation of the page and if it is valid, prints “Successful Entry!”.




## 5.6 RegularExpressionValidator

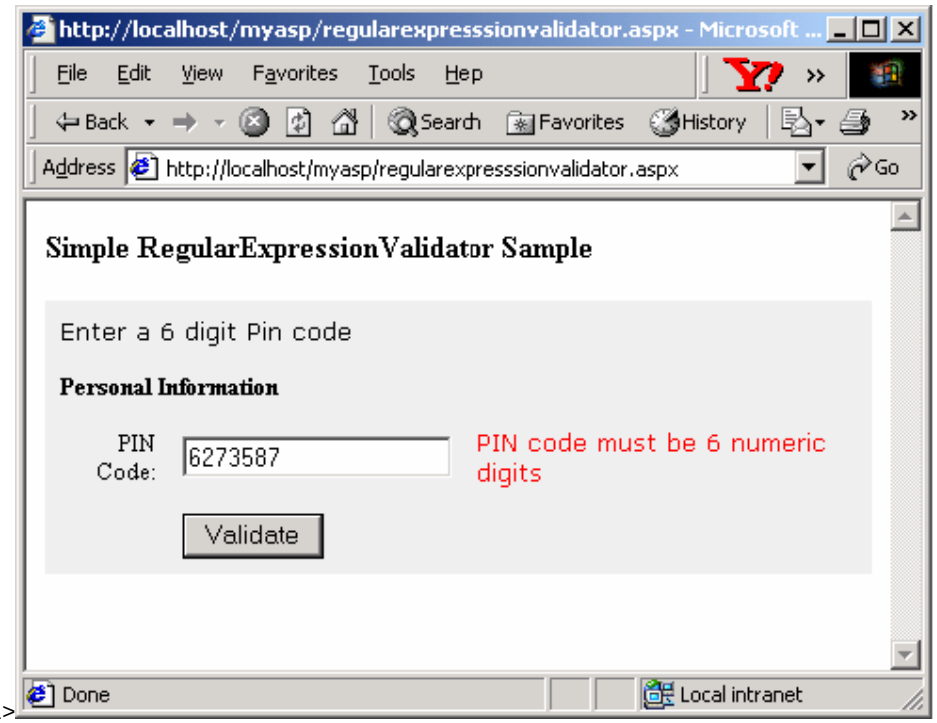
The RegularExpression Validator evaluates the data that a user enters against a regular expression. This allows for complex formatting restrictions on the field data. A popular example of this kind is the email address. A RegularExpression Validator can check if there is at least one character, followed by the '@' symbol, followed by at least one character, followed by a period, which is followed by either 'com', 'org', 'net' or any other valid email extension. This enables the developer to force the user to enter a valid email address.

### Practice 5.4

Here is an example with RegularExpression validator which validates if a specific number of digits have been entered.

```
<html>
    <head>
         <script language="C#" runat=server>
            void ValidateBtn_Click(Object Src, EventArgs E)
            {
                 if (Page.IsValid)
                 {
                    lblMessage.Text="Page is Valid!";
                 }
            }
        </script>
    </head>
<body>
<div class="header"><h3>Simple RegularExpressionValidator Sample</h3></div>
```

```
<form runat=”server”>
<center>
<table bgcolor=”#eeeeee” cellpadding=6>
<tr valign=”top”>
    <td colspan=3> <asp:label id=”lblMessage” text=”Enter a 6 digit Pin code”
font-name=”Verdana” font-size=”10pt” runat=”server”/>
    </td>
</tr>
<tr>
    <td colspan=3><b>Personal Information</b></td></tr>
<tr>
    <td align=right>
        PIN Code:</font>
    </td>
    <td>
        <asp:textbox id=”TextBox1" runat=server /></td>
   <td> <asp:regularexpressionvalidator id=”RegularExpressionValidator1"
           runat=”server”
     controltovalidate=”TextBox1"
     validationexpression=”^\d{6}$”
     display=”Static”
     font-name=”verdana”
      font-size=”10pt”>
        PIN code must be 6 numeric digits
      </asp:regularexpressionvalidator>
     </td>
   </tr>
   <tr>
     <td></td>
     <td>
      <asp:button text=”Validate” OnClick=”ValidateBtn_Click” runat=server />
     </td>
   <td></td></tr>
</table>
</center>
</form>
</body>
```




```
</html>
```

In the above example, the text box control textbox1 is used to get the PIN Code from the user. RegularExpressionValidator is used to check the validity of the PIN Code. So the attribute control to validate is set to textbox1. The color of the error message is set to red so that the user can easily notice the error message. (When the control shifts from the textbox, the RegularExpressionValidator checks for the validity of the PIN Code and if there is an error prints the error message even before the "Validate" button is clicked. It checks if all the entries are numeric and that there are six significant digits. When the "validate" button is clicked, it checks for the validation of the page and if it is valid prints "Page is Valid!".

## 5.7 CustomValidator

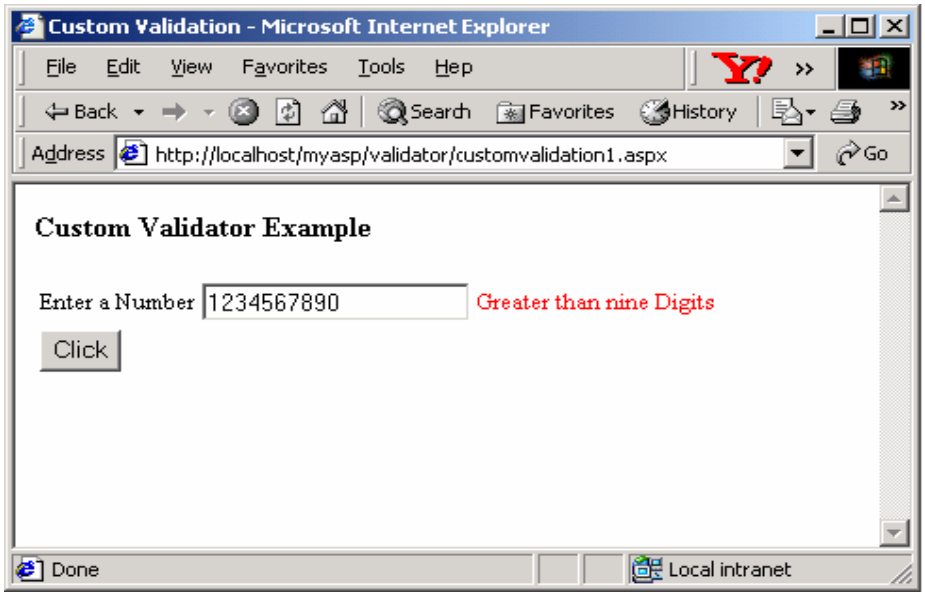
The Custom Validator allows the developer to create user-defined criteria for validation. The Validation can be defined at two funcions which they determine. The developer can place any logic he or she wants in these functions to determine if the data is acceptable. A common use of this Validator is to check for a username against the usernames in a database, to see if the person is a registered user. Credit Card number validation is also a very common use of the Custom Validator.

### Practice 5.5

Given below is a typical example of using the custom validator control. A text control is used to get an user input and if the input exceeds 9 digits the custom validator prints an error message.

```
<html>
<title>Custom Validation</title>
<script language="c#" runat="server">
     bool ServerValidate(object sender,String value)
            {
                if (value.Length <= 9)
                   return true;
              else
```

```
                    return false;
                }
</script>
<body>
<h3>Custom Validator Example</h3>
<form runat=server>
<table>
<tr>
<td> <asp:label id="label1" text="Enter a Number" runat="server"/> </td>
<td> <asp:textbox id="text1" runat="server" /> </td>

<td> <asp:CustomValidator id="CustomValidator1" runat="server"
        ControlToValidate="text1"
        OnServerValidate="ServerValidate"
        Display="static">
        Greater than nine Digits
        </asp:customvalidator>
</td>
</tr>
<tr>
<td><asp:button id="bt1" text="Click" runat="server" /> </td>
<td> <asp:label id="label2" text=" " runat="server" /> </td>
<td> <asp:label id="label3" text=" " runat="server" /> </td>
</tr>
</table>
</form>
</body>
</html>
```



In the above example, the text box control text1 is used to allow the user to enter a number. CustomerValidator checks the validity of the number entered. So the attribute control to validate is set to text1. When the control shifts from text1, the CustomValidator checks for the validity of the number and if there is an error, prints the error message even before the click button is clicked. The color of the error message is set to red so that the user can easily notice the error message. This checks if all the entries are numeric and that the number of digits are less than or equal to 9. If it is greater than 9, it prints the error message "Greater than nine Digits".

## 5.8 Validation Summary Control

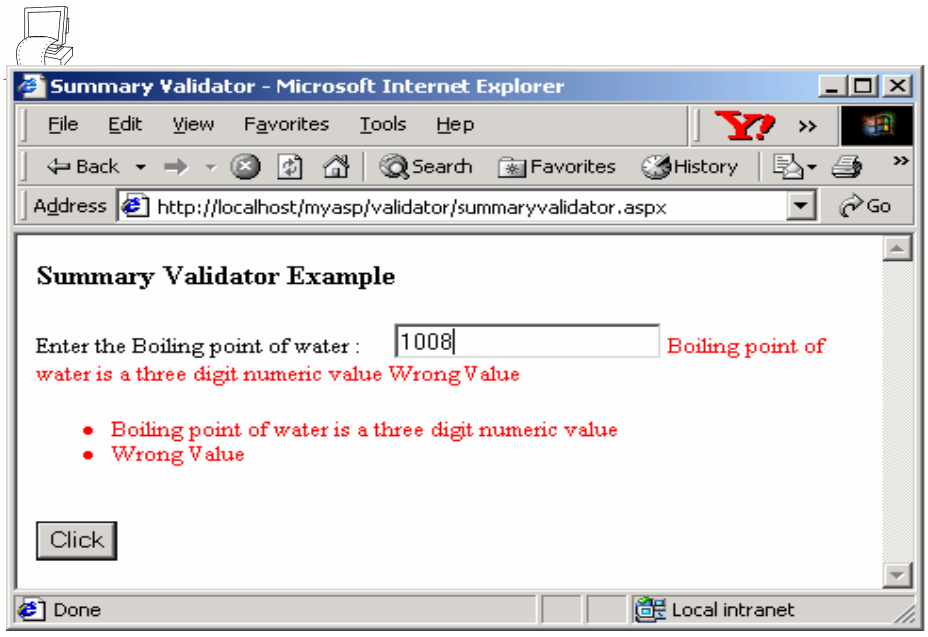
ValidationSummary control, displays a list of all the errors. This control collects all the validation errors and places them in a list within the page. The ValidationSummary control uses the ErrorMessage property of the other validation controls. The color of the error text and a header for the list can be specified. This header will be displayed only if there is an error.

### Practice 5.6

In this example four validation controls are used. All the controls are managed by the ValidationSummary control.

```
<html>
<title>Summary Validator </title>
 <script language="C#" runat=server>
  void ValidateBtn_Click(Object Src, EventArgs E)
    {
         if (Page.IsValid == true)
         {
            lblMessage.Text="Page is Valid!";
         }
    }
</script>
<body>
<h3>Summary Validator Example </h3>
<form runat="server">
    <asp:label id="label1"
      text="Enter the Boiling point of water : "
      runat="server" />
        
    <asp:textbox id="text1"
      runat="server"/>
    <asp:label id="lblMessage"  runat="server"/>
    <asp:regularexpressionvalidator
      id="RegularExpressionValidator1"
      runat="server"
      controltovalidate="Text1"
      validationexpression="^\d{3}$"
      display="Static"
  Errormessage = "Boiling point of water is a three digit numeric    value" >
```

```
    </asp:regularexpressionvalidator>
    <asp:comparevalidator id="compboilpt"
      controltovalidate="text1"
      Type="Integer"
      ValueToCompare=100
      Operator="Equal"
      display="static"
      Errormessage="Wrong Value"
      runat="server">
    </asp:comparevalidator>
  <asp:requiredfieldvalidator
       id="RequiredFieldValidator1"
       controltovalidate="Text1"
       runat="server"
       Errormessage = "Should Enter a Value">
     </asp:requiredfieldvalidator>
  <asp:ValidationSummary
       id="sumErrors"
       runat="server"
       showSummary = true
       displayMode = "BulletList">
    </asp:ValidationSummary >
<br>
<asp:button text="Click" OnClick="ValidateBtn_Click" runat=server />
</form>
</body>
</html>
```



In the above example, the text box control text1 is used to get the boiling point of water. RegularExpressionValidator, CompareValidator, RequiredFieldValidator are used to check the validity of the figure entered. So the attribute controltovalidate is set to text1 for all the validators. When the control shifts from text1, the RegularExpressionValidator checks for the validity of the number entered for the boiling point of water and if there is an error, prints the error message even before the "click" button is clicked. The RegularExpressionValidator checks for the number of digits that are entered, if it is less than or greater than 3 an error message is displayed. The ComparisonValidator checks whether the entry is exactly equal to 100. The RequiredFieldValidator checks if there is an entry in the text box. The ValidationSummary control lists all the errors as bulleted list.

## 5.9 Short Summary

- Validator controls can be used to validate input on any Web Form
- More than one control can be used on a given input field
- Client-side validation may be used in addition to server-validation to improve form usability
- The CustomValidator control lets the user define custom validation criteria

## 5.10 Brain Storm

1. What is client-side validation?
2. What are the validation controls available in ASP.NET?
3. Which control can be used to validate against a data type?
4. How is pattern-matching validation done through RegualarExpression Validator?
5. What is the advantage of ValidationSummary control?
6. What is the use of TabIndex?
7. How are user inputs validated?
8. What are the different ways in which error messages ca nbe displayed?
9. What are the different types of Validators available in ASP.NET?
10. What is the purpose of the ValidationSummary control?