TOPIC 4 Introduction to HTML Server

LEARNING OUTCOME

By the end of this topic, students will be able to:

- Describe HTML Server
- Explain the Advantages of using HTML Server Controls
- Write Client Side Programs
- Use ASP.NET Basic Controls
- Discuss View, Control, Session and Application States Management
- Describe Validators
- Write RequiredFieldValidator, RangeValidator, CompareValidator
- Use RegularExpressionValidator, CustomValidator, ValidationSummary
- Use BaseValidator Class
- Apply Validation Groups

Introduction

The HTML server controls are basically the standard HTML controls enhanced to enable server side processing. The HTML controls such as the header tags, anchor tags, and input elements are not processed by the server but are sent to the browser for display.
They are specifically converted to a server control by adding the attribute `runat="server"` and adding an id attribute to make them available for server-side processing.

For example, consider the HTML input control:

```
<input type="text" size="40">
```

It could be converted to a server control, by adding the runat and id attribute:

```
<input type="text" id="testtext" size="40" runat="server">
```

Advantages of using HTML Server Controls

Although ASP.NET server controls can perform every job accomplished by the HTML server controls, the later controls are useful in the following cases:

- Using static tables for layout purposes.
- Converting a HTML page to run under ASP.NET

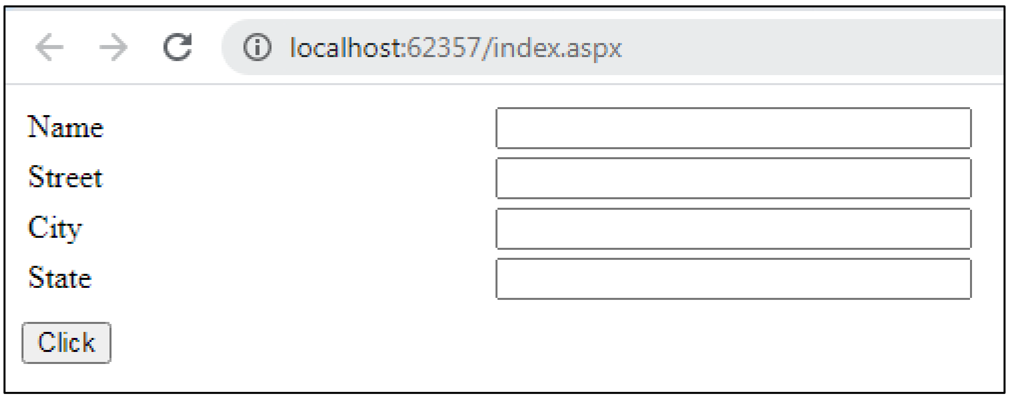
The following table describes the HTML server controls:

Control Name	HTML tag
HtmlHead	`<head>element`
HtmlInputButton	`<input type=button|submit|reset>`
HtmlInputCheckbox	`<input type=checkbox>`
HtmlInputFile	`<input type = file>`
HtmlInputHidden	`<input type = hidden>`
HtmlInputImage	`<input type = image>`
HtmlInputPassword	`<input type = password>`
HtmlInputRadioButton	`<input type = radio>`
HtmlInputReset	`<input type = reset>`
HtmlText	`<input type = text|password>`
HtmlImage	`<img> element`
HtmlLink	`<link> element`
HtmlAnchor	`<a> element`
HtmlButton	`<button> element`
HtmlButton	`<button> element`
HtmlForm	`<form> element`
HtmlTable	`<table> element`
HtmlTableCell	`<td> and <th>`
HtmlTableRow	`<tr> element`
HtmlTitle	`<title> element`
HtmlSelect	`<select&t; element`
HtmlGenericControl	All HTML controls not listed

Example 1

The following example uses a basic HTML table for layout. It uses some boxes for getting input from the users such as name, address, city, and state. It also has a button control, which is clicked to get the user data displayed in the last row of the table.

The page should look like this in the design view:



The code for the content page shows the use of the HTML table element for layout.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="index.aspx.cs"
Inherits="LUC_Web_Sample2.index" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
     <style type="text/css">
         .style1
         {
            width: 156px;
         }
         .style2
         {
            width: 332px;
         }
      </style>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <table style="width: 54%;">
               <tr>
                  <td class="style1">Name</td>
                  <td class="style2">
                     <asp:TextBox ID="txtname" runat="server"
style="width:230px">
                     </asp:TextBox>
                  </td>
               </tr>

               <tr>
                  <td class="style1">Street</td>
                  <td class="style2">
                     <asp:TextBox ID="txtstreet" runat="server"
style="width:230px">
                     </asp:TextBox>
                  </td>
               </tr>

               <tr>
                  <td class="style1">City</td>
                  <td class="style2">
                     <asp:TextBox ID="txtcity" runat="server"
style="width:230px">
                     </asp:TextBox>
                  </td>
               </tr>

               <tr>
                  <td class="style1">State</td>
                  <td class="style2">
                     <asp:TextBox ID="txtstate" runat="server"
style="width:230px">
                     </asp:TextBox>
                  </td>
               </tr>

               <tr>
                  <td class="style1"> </td>
                  <td class="style2"></td>
               </tr>

               <tr>
                  <td class="style1"></td>
                  <td ID="displayrow" runat ="server" class="style2">
                  </td>
               </tr>
            </table>
    </div>
         <asp:Button ID="Button1" runat="server" onclick="Button1_Click"
Text="Click" />
    </form>
</body>
```

```
</html>
```

The code behind the button control:

```
using System;


namespace LUC_Web_Sample2
{
    public partial class index : System.Web.UI.Page
    {

        protected void Button1_Click(object sender, EventArgs e)
        {
            string str = "";
            str += txtname.Text + "<br />";
            str += txtstreet.Text + "<br />";
            str += txtcity.Text + "<br />";
            str += txtstate.Text + "<br />";
            displayrow.InnerHtml = str;
        }
    }
  }
```

Observe the following:

- The standard HTML tags have been used for the page layout.
- The last row of the HTML table is used for data display. It needed server side processing, so an ID attribute and the runat attribute has been added to it.

Client Side Programming

ASP.NET client side coding has two aspects:

- **Client side scripts:** It runs on the browser and in turn speeds up the execution of page. For example, client side data validation which can catch invalid data and warn the user accordingly without making a round trip to the server.
- **Client side source code:** ASP.NET pages generate this. For example, the HTML source code of an ASP.NET page contains a number of hidden fields and automatically injected blocks of JavaScript code, which keeps information like view state or does other jobs to make the page work.

Client Side Scripts

All ASP.NET server controls allow calling client side code written using JavaScript or VBScript. Some ASP.NET server controls use client side scripting to provide response to the users without posting back to the server. For example, the validation controls.
Apart from these scripts, the Button control has a property `OnClientClick`, which allows executing client-side script, when the button is clicked.

The traditional and server HTML controls have the following events that can execute a script when they are raised:

Event	Description
onblur	When the control loses focus
onfocus	When the control receives focus
onclick	When the control is clicked
onchange	When the value of the control changes

onkeydown	When the user presses a key
onkeypress	When the user presses an alphanumeric key
onkeyup	When the user releases a key
onmouseover	When the user moves the mouse pointer over the control
onserverclick	It raises the ServerClick event of the control, when the control is clicked

Client Side Source Code

As mentioned in our last class, ASP.NET pages are generally written in two files:

- The content file or the markup file (.aspx)
- The code-behind file

The content file contains the HTML or ASP.NET control tags and literals to form the structure of the page. The code behind file contains the class definition. At run-time, the content file is parsed and transformed into a page class.

This class, along with the class definition in the code file, and system generated code, together make the executable code (assembly) that processes all posted data, generates response, and sends it back to the client.

Consider the simple page:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
   Inherits="clientside._Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

   <head runat="server">
      <title>
         Untitled Page
      </title>
   </head>

   <body>
      <form id="form1" runat="server">

         <div>
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
            <asp:Button  ID="Button1"  runat="server"  OnClick="Button1_Click"
Text="Click" />
         </div>

         <hr />

         <h3> <asp:Label ID="Msg" runat="server" Text=""> </asp:Label> </h3>
      </form>
   </body>

</html>
```

When this page is run on the browser, the View Source option shows the HTML page sent to the browser by the ASP.Net runtime:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >

   <head>
      <title>
         Untitled Page
      </title>
   </head>

   <body>
      <form name="form1" method="post" action="Default.aspx" id="form1">

         <div>
            <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
               value="/wEPDwUKMTU5MTA2ODYwOWRk31NudGDgvhhA7joJum9Qn5RxU2M=" />
         </div>

         <div>
            <input          type="hidden"           name="__EVENTVALIDATION"
id="__EVENTVALIDATION"

value="/wEWAwKpjZj0DALs0bLrBgKM54rGBhHsyM61rraxE+KnBTCS8cd1QDJ/"/>
         </div>

         <div>
            <input name="TextBox1" type="text" id="TextBox1" />
            <input type="submit" name="Button1" value="Click" id="Button1" />
         </div>

         <hr />
         <h3><span id="Msg"></span></h3>

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

If you go through the code properly, you can see that first two <div> tags contain the hidden fields which store the view state and validation information.

ASP.NET Basic Controls

Button Controls

ASP.NET provides three types of button control:

- **Button**: It displays text within a rectangular area.
- **Link Button**: It displays text that looks like a hyperlink.
- **Image Button**: It displays an image.

When a user clicks a button, two events are raised: Click and Command.

Basic syntax of button control:

```
<asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="Click" />
```

Common properties of the button control:

Property	Description
Text	The text displayed on the button. This is for button and link button controls only.
ImageUrl	For image button control only. The image to be displayed for the button.

AlternateText	For image button control only. The text to be displayed if the browser cannot display the image.
CausesValidation	Determines whether page validation occurs when a user clicks the button. The default is true.
CommandName	A string value that is passed to the command event when a user clicks the button.
CommandArgument	A string value that is passed to the command event when a user clicks the button.
PostBackUrl	The URL of the page that is requested when the user clicks the button.

Text Boxes and Labels

Text box controls are typically used to accept input from the user. A text box control can accept one or more lines of text depending upon the settings of the TextMode attribute.

Label controls provide an easy way to display text which can be changed from one execution of a page to the next. If you want to display text that does not change, you use the literal text.

Basic syntax of text control:

```
<asp:TextBox ID="txtstate" runat="server" ></asp:TextBox>
```

Common Properties of the Text Box and Labels:

Property	Description
TextMode	Specifies the type of text box. SingleLine creates a standard text box, MultiLIne creates a text box that accepts more than one line of text and the Password causes the characters that are entered to be masked. The default is SingleLine.
Text	The text content of the text box.
MaxLength	The maximum number of characters that can be entered into the text box.
Wrap	It determines whether or not text wraps automatically for multi-line text box; default is true.
ReadOnly	Determines whether the user can change the text in the box; default is false, i.e., the user cannot change the text.
Columns	The width of the text box in characters. The actual width is determined based on the font that is used for the text entry.
Rows	The height of a multi-line text box in lines. The default value is 0, means a single line text box.

The mostly used attribute for a label control is `'Text'`, which implies the text displayed on the label.

Check Boxes and Radio Buttons

A check box displays a single option that the user can either check or uncheck and radio buttons present a group of options from which the user can select just one option.

To create a group of radio buttons, you specify the same name for the GroupName attribute of each radio button in the group. If more than one group is required in a single form, then specify a different group name for each group.

If you want check box or radio button to be selected when the form is initially displayed, set its Checked attribute to true. If the Checked attribute is set to true for multiple radio buttons in a group, then only the last one is considered as true.

Basic syntax of check box:

```
<asp:CheckBox ID= "chkoption" runat= "Server">
</asp:CheckBox>
```

Basic syntax of radio button:

```
<asp:RadioButton ID= "rdboption" runat= "Server">
</asp: RadioButton>
```

Common properties of check boxes and radio buttons:

Property	Description
Text	The text displayed next to the check box or radio button.
Checked	Specifies whether it is selected or not, default is false.
GroupName	Name of the group the control belongs to.

List Controls

ASP.NET provides the following controls:

- Drop-down list,
- List box,
- Radio button list,
- Check box list,
- Bulleted list.

These control let a user choose from one or more items from the list. List boxes and drop-down lists contain one or more list items. These lists can be loaded either by code or by the ListItemCollection editor.

Basic syntax of list box control:

```
<asp:ListBox      ID="ListBox1"      runat="server"      AutoPostBack="True"
OnSelectedIndexChanged="ListBox1_SelectedIndexChanged">
</asp:ListBox>
```

Basic syntax of drop-down list control:

```
<asp:DropDownList   ID="DropDownList1"   runat="server"   AutoPostBack="True"
OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
</asp:DropDownList>
```

Common properties of list box and drop-down Lists:

Property	Description
Items	The collection of ListItem objects that represents the items in the control. This property returns an object of type ListItemCollection.
Rows	Specifies the number of items displayed in the box. If actual list contains more rows than displayed then a scroll bar is added.
SelectedIndex	The index of the currently selected item. If more than one item is selected, then the index of the first selected item. If no item is selected, the value of this property is -1.

SelectedValue	The value of the currently selected item. If more than one item is selected, then the value of the first selected item. If no item is selected, the value of this property is an empty string ("").
SelectionMode	Indicates whether a list box allows single selections or multiple selections.

Common properties of each list item objects:

Property	**Description**
Text	The text displayed for the item.
Selected	Indicates whether the item is selected.
Value	A string value associated with the item.

It is important to notes that:

- To work with the items in a drop-down list or list box, you use the Items property of the control. This property returns a ListItemCollection object which contains all the items of the list.
- The SelectedIndexChanged event is raised when the user selects a different item from a drop-down list or list box.

The ListItemCollection

The ListItemCollection object is a collection of ListItem objects. Each ListItem object represents one item in the list. Items in a ListItemCollection are numbered from 0.

When the items into a list box are loaded using strings like: lstcolor.Items.Add("Blue"), then both the Text and Value properties of the list item are set to the string value you specify. To set it differently you must create a list item object and then add that item to the collection.

The ListItemCollection Editor is used to add item to a drop-down list or list box. This is used to create a static list of items. To display the collection editor, select edit item from the smart tag menu, or select the control and then click the ellipsis button from the Item property in the properties window.

Common properties of ListItemCollection:

Property	**Description**
Item(integer)	A ListItem object that represents the item at the specified index.
Count	The number of items in the collection.

Common methods of ListItemCollection:

Methods	**Description**
Add(string)	Adds a new item at the end of the collection and assigns the string parameter to the Text property of the item.
Add(ListItem)	Adds a new item at the end of the collection.
Insert(integer, string)	Inserts an item at the specified index location in the collection, and assigns string parameter to the text property of the item.
Insert(integer, ListItem)	Inserts the item at the specified index location in the collection.
Remove(string)	Removes the item with the text value same as the string.
Remove(ListItem)	Removes the specified item.
RemoveAt(integer)	Removes the item at the specified index as the integer.
Clear	Removes all the items of the collection.
FindByValue(string)	Returns the item whose value is same as the string.
FindByValue(Text)	Returns the item whose text is same as the string.

Radio Button list and Check Box list

A radio button list presents a list of mutually exclusive options. A check box list presents a list of independent options. These controls contain a collection of ListItem objects that could be referred to through the Items property of the control.

Basic syntax of radio button list:

```
<asp:RadioButtonList ID="RadioButtonList1" runat="server" AutoPostBack="True"
   OnSelectedIndexChanged="RadioButtonList1_SelectedIndexChanged">
</asp:RadioButtonList>
```

Basic syntax of check box list:

```
<asp:CheckBoxList ID="CheckBoxList1" runat="server" AutoPostBack="True"
   OnSelectedIndexChanged="CheckBoxList1_SelectedIndexChanged">
</asp:CheckBoxList>
```

Common properties of check box and radio button lists:

Property	**Description**
RepeatLayout	This attribute specifies whether the table tags or the normal html flow to use while formatting the list when it is rendered. The default is Table.
RepeatDirection	It specifies the direction in which the controls to be repeated. The values available are Horizontal and Vertical. Default is Vertical.
RepeatColumns	It specifies the number of columns to use when repeating the controls; default is 0.

Bulleted lists and Numbered lists

The bulleted list control creates bulleted lists or numbered lists. These controls contain a collection of ListItem objects that could be referred to through the Items property of the control.

Basic syntax of a bulleted list:

```
<asp:BulletedList ID="BulletedList1" runat="server">
</asp:BulletedList>
```

Common properties of the bulleted list:

Property	**Description**
BulletStyle	This property specifies the style and looks of the bullets, or numbers.
RepeatDirection	It specifies the direction in which the controls to be repeated. The values available are Horizontal and Vertical. Default is Vertical.
RepeatColumns	It specifies the number of columns to use when repeating the controls; default is 0.

HyperLink Control

The HyperLink control is like the HTML `<a>` element.

Basic syntax for a hyperlink control:

```
<asp:HyperLink ID="HyperLink1" runat="server">
```

```
   HyperLink
</asp:HyperLink>
```

It has the following important properties:

Property	Description
ImageUrl	Path of the image to be displayed by the control.
NavigateUrl	Target link URL.
Text	The text to be displayed as the link.
Target	The window or frame which loads the linked page.

Image Control

The image control is used for displaying images on the web page, or some alternative text, if the image is not available.

Basic syntax for an image control:

```
<asp:Image ID="Image1" runat="server">
```

It has the following important properties:

Property	Description
AlternateText	Alternate text to be displayed in absence of the image.
ImageAlign	Alignment options for the control.
ImageUrl	Path of the image to be displayed by the control.

State Management

Hyper Text Transfer Protocol (HTTP) is a stateless protocol. When the client disconnects from the server, the ASP.NET engine discards the page objects. This way, each web application can scale up to serve numerous requests simultaneously without running out of server memory.

However, there needs to be some technique to store the information between requests and to retrieve it when required. This information i.e., the current value of all the controls and variables for the current user in the current session is called the State.

ASP.NET manages four types of states:

- View State
- Control State
- Session State
- Application State

View State
The view state is the state of the page and all its controls. It is automatically maintained across posts by the ASP.NET framework.

When a page is sent back to the client, the changes in the properties of the page and its controls are determined, and stored in the value of a hidden input field named _VIEWSTATE. When the page is again posted back, the _VIEWSTATE field is sent to the server with the HTTP request.

The view state could be enabled or disabled for:

- The entire application by setting the EnableViewState property in the <pages> section of web.config file.

- A page by setting the EnableViewState attribute of the Page directive, as <%@ Page Language="C#" EnableViewState="false" %>
- A control by setting the Control.EnableViewState property.

It is implemented using a view state object defined by the StateBag class which defines a collection of view state items. The state bag is a data structure containing attribute value pairs, stored as strings associated with objects.

The StateBag class has the following properties:

Properties	Description
Item(name)	The value of the view state item with the specified name. This is the default property of the StateBag class.
Count	The number of items in the view state collection.
Keys	Collection of keys for all the items in the collection.
Values	Collection of values for all the items in the collection.

The StateBag class has the following methods:

Methods	Description
Add(name, value)	Adds an item to the view state collection and existing item is updated.
Clear	Removes all the items from the collection.
Equals(Object)	Determines whether the specified object is equal to the current object.
Finalize	Allows it to free resources and perform other cleanup operations.
GetEnumerator	Returns an enumerator that iterates over all the key/value pairs of the StateItem objects stored in the StateBag object.
GetType	Gets the type of the current instance.
IsItemDirty	Checks a StateItem object stored in the StateBag object to evaluate whether it has been modified.
Remove(name)	Removes the specified item.
SetDirty	Sets the state of the StateBag object as well as the Dirty property of each of the StateItem objects contained by it.
SetItemDirty	Sets the Dirty property for the specified StateItem object in the StateBag object.
ToString	Returns a string representing the state bag object.

Example 2

The following example demonstrates the concept of storing view state. Let us keep a counter, which is incremented each time the page is posted back by clicking a button on the page. A label control shows the value in the counter.

The markup file code is as follows:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="viewStateSample.aspx.cs"
Inherits="LUC_Web_Sample2.viewStateSample" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
```

```
            <h3>View State Example</h3>
          Page Counter
        <asp:Label ID="lblCounter" runat="server" />
          <asp:Button ID="btnIncrement" runat="server" Text="Add Count" />
    </div>
    </form>
</body>
</html>
```

The code behind file for the example is shown here:

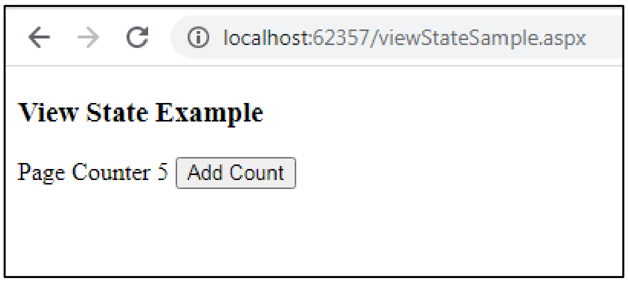
```
using System;

namespace LUC_Web_Sample2
{
    public partial class viewStateSample : System.Web.UI.Page
    {
        public int counter
        {
            get
            {
                if (ViewState["pcounter"] != null)
                {
                    return ((int)ViewState["pcounter"]);
                }
                else
                {
                    return 0;
                }
            }

            set
            {
                ViewState["pcounter"] = value;
            }
        }

        protected void Page_Load(object sender, EventArgs e)
        {
            lblCounter.Text = counter.ToString();
            counter++;
        }
    }
}
```

It would produce the following result:



Control State

Control state cannot be modified, accessed directly, or disabled.

Session State

When a user connects to an ASP.NET website, a new session object is created. When session state is turned on, a new session state object is created for each new request. This session state object becomes part of the context and it is available through the page.

Session state is generally used for storing application data such as inventory, supplier list, customer record, or shopping cart. It can also keep information about the user and his preferences, and keep the track of pending operations.

Sessions are identified and tracked with a 120-bit SessionID, which is passed from client to server and back as cookie or a modified URL. The SessionID is globally unique and random.

The session state object is created from the HttpSessionState class, which defines a collection of session state items.

The HttpSessionState class has the following properties:

Properties	**Description**
SessionID	The unique session identifier.
Item(name)	The value of the session state item with the specified name. This is the default property of the HttpSessionState class.
Count	The number of items in the session state collection.
TimeOut	Gets and sets the amount of time, in minutes, allowed between requests before the session-state provider terminates the session.

The HttpSessionState class has the following methods:

Methods	**Description**
Add(name, value)	Adds an item to the session state collection.
Clear	Removes all the items from session state collection.
Remove(name)	Removes the specified item from the session state collection.
RemoveAll	Removes all keys and values from the session-state collection.
RemoveAt	Deletes an item at a specified index from the session-state collection.

The session state object is a name-value pair to store and retrieve some information from the session state object. You could use the following code for the same:

```
void StoreSessionInfo()
{
   String fromuser = TextBox1.Text;
   Session["fromuser"] = fromuser;
}

void RetrieveSessionInfo()
{
   String fromuser = Session["fromuser"];
   Label1.Text = fromuser;
}
```

The above code stores only strings in the Session dictionary object, however, it can store all the primitive data types and arrays composed of primitive data types, as well as the DataSet, DataTable, HashTable, and Image objects, as well as any user-defined class that inherits from the ISerializable object.

Example 3

The following example demonstrates the concept of storing session state. There are two buttons on the page, a text box to enter string and a label to display the text stored from last session.

The mark up file code is as follows:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="sessionSample.aspx.cs" Inherits="LUC_Web_Sample2.sessionSample" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        &nbsp; &nbsp; &nbsp;

            <table style="width: 568px; height: 103px">
               <tr>
                  <td style="width: 209px">
                     <asp:Label ID="lblstr" runat="server" Text="Enter a
String"  style="width:94px">
                     </asp:Label>
                  </td>

                  <td style="width: 317px">
                     <asp:TextBox ID="txtstr" runat="server"
style="width:227px">
                     </asp:TextBox>
                  </td>
               </tr>

               <tr>
                  <td style="width: 209px"> </td>
                  <td style="width: 317px"> </td>
               </tr>

               <tr>
                  <td style="width: 209px">
                     <asp:Button ID="btnnrm" runat="server" Text="No action
button" style="width:128px" />
                  </td>

                  <td style="width: 317px">
                     <asp:Button ID="btnstr" runat="server" Text="Submit the
String" OnClick="btnstr_Click" />
                  </td>
               </tr>

               <tr>
                  <td style="width: 209px">  </td>

                  <td style="width: 317px">  </td>
               </tr>

               <tr>
                  <td style="width: 209px">
                     <asp:Label ID="lblsession" runat="server"
style="width:231px"  >
```

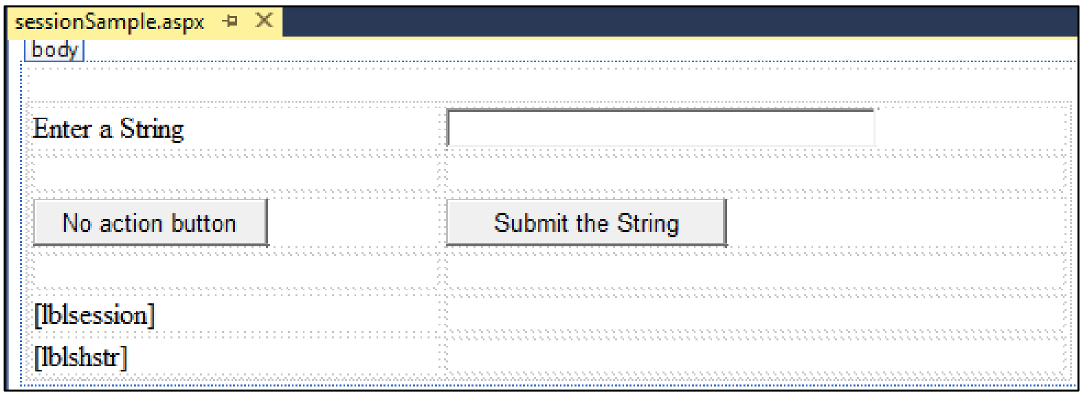
```
                        </asp:Label>
                     </td>

                     <td style="width: 317px">  </td>
                  </tr>

                  <tr>
                     <td style="width: 209px">
                        <asp:Label ID="lblshstr" runat="server">
                        </asp:Label>
                     </td>

                     <td style="width: 317px">  </td>
                  </tr>
               </table>
       </div>
       </form>
</body>
</html>
```

It should look like the following in design view:



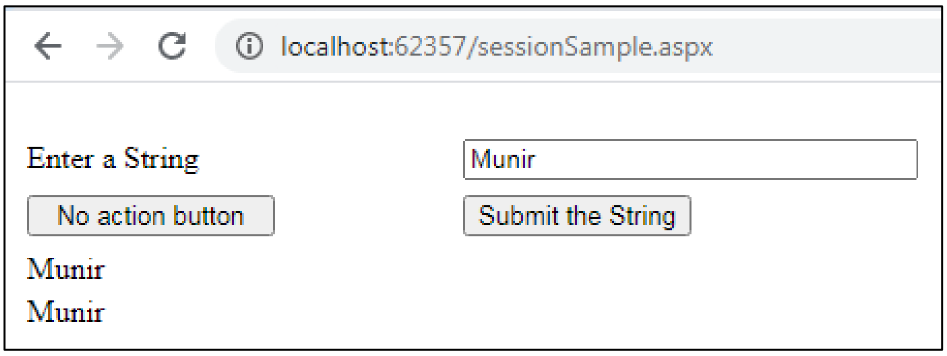
The code behind file is given here:

```
public partial class _Default : System.Web.UI.Page
{
   String mystr;

   protected void Page_Load(object sender, EventArgs e)
   {
      this.lblshstr.Text = this.mystr;
      this.lblsession.Text = (String)this.Session["str"];
   }

   protected void btnstr_Click(object sender, EventArgs e)
   {
      this.mystr = this.txtstr.Text;
      this.Session["str"] = this.txtstr.Text;
      this.lblshstr.Text = this.mystr;
      this.lblsession.Text = (String)this.Session["str"];
   }
}
```

Execute the file and observe how it works:



Munir
Munir

Application State

The ASP.NET application is the collection of all web pages, code and other files within a single virtual directory on a web server. When information is stored in application state, it is available to all the users.

To provide for the use of application state, ASP.NET creates an application state object for each application from the HTTPApplicationState class and stores this object in server memory. This object is represented by class file global.asax.

Application State is mostly used to store hit counters and other statistical data, global application data like tax rate, discount rate etc. and to keep the track of users visiting the site.

The HttpApplicationState class has the following properties:

Properties	Description
Item(name)	The value of the application state item with the specified name. This is the default property of the HttpApplicationState class.
Count	The number of items in the application state collection.

The HttpApplicationState class has the following methods:

Methods	Description
Add(name, value)	Adds an item to the application state collection.
Clear	Removes all the items from the application state collection.
Remove(name)	Removes the specified item from the application state collection.
RemoveAll	Removes all objects from an HttpApplicationState collection.
RemoveAt	Removes an HttpApplicationState object from a collection by index.
Lock()	Locks the application state collection so only the current user can access it.
Unlock()	Unlocks the application state collection so all the users can access it.

Application state data is generally maintained by writing handlers for the events:

- Application_Start
- Application_End
- Application_Error
- Session_Start
- Session_End

The following code snippet shows the basic syntax for storing application state information:

```
void Application_Start(object sender, EventArgs e)
{
   Application["startMessage"] = "The application has started.";
}

void Application_End(object sender, EventArgs e)
{
   Application["endtMessage"] = "The application has ended.";
}
```

Validators

ASP.NET validation controls validate the user input data to ensure that harmful, unauthenticated, or contradictory data don't get stored.

ASP.NET provides the following validation controls:

- RequiredFieldValidator
- RangeValidator
- CompareValidator
- RegularExpressionValidator
- CustomValidator
- ValidationSummary
- BaseValidator Class

The validation control classes are inherited from the BaseValidator class hence they inherit its properties and methods. Therefore, it would help to take a look at the properties and the methods of this base class, which are common for all the validation controls:

Members	Description
ControlToValidate	Indicates the input control to validate.
Display	Indicates how the error message is shown.
EnableClientScript	Indicates whether client side validation will take.
Enabled	Enables or disables the validator.
ErrorMessage	Indicates error string.
Text	Error text to be shown if validation fails.
IsValid	Indicates whether the value of the control is valid.
SetFocusOnError	It indicates whether in case of an invalid control, the focus should switch to the related input control.
ValidationGroup	The logical group of multiple validators, where this control belongs.
Validate()	This method revalidates the control and updates the IsValid property.

RequiredFieldValidator Control

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

The syntax of the control is as given:

```
<asp:RequiredFieldValidator ID="rfvcandidate"
   runat="server" ControlToValidate ="ddlcandidate"
   ErrorMessage="Please choose a candidate"
   InitialValue="Please choose a candidate">

</asp:RequiredFieldValidator>
```

RangeValidator Control

The RangeValidator control verifies that the input value falls within a predetermined range.

It has three specific properties:

Properties	Description
Type	It defines the type of the data. The available values are: Currency, Date, Double, Integer, and String.
MinimumValue	It specifies the minimum value of the range.
MaximumValue	It specifies the maximum value of the range.

The syntax of the control is as given:

```
<asp:RangeValidator ID="rvclass" runat="server" ControlToValidate="txtclass"
   ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"
   MinimumValue="6" Type="Integer">

</asp:RangeValidator>
```

CompareValidator Control

The CompareValidator control compares a value in one control with a fixed value or a value in another control.

It has the following specific properties:

Properties	Description
Type	It specifies the data type.
ControlToCompare	It specifies the value of the input control to compare with.
ValueToCompare	It specifies the constant value to compare with.
Operator	It specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and DataTypeCheck.

The basic syntax of the control is as follows:

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
   ErrorMessage="CompareValidator">

</asp:CompareValidator>
```

RegularExpressionValidator

The RegularExpressionValidator allows validating the input text by matching against a pattern of a regular expression. The regular expression is set in the ValidationExpression property.

The following table summarizes the commonly used syntax constructs for regular expressions:

Character Escapes	Description
`\b`	Matches a backspace.
`\t`	Matches a tab.
`\r`	Matches a carriage return.
`\v`	Matches a vertical tab.
`\f`	Matches a form feed.

`\n`	Matches a new line.
`\`	Escape character.

Apart from single character match, a class of characters could be specified that can be matched, called the metacharacters.

Metacharacters	**Description**
`.`	Matches any character except \n.
`[abcd]`	Matches any character in the set.
`[^abcd]`	Excludes any character in the set.
`[2-7a-mA-M]`	Matches any character specified in the range.
`\w`	Matches any alphanumeric character and underscore.
`\W`	Matches any non-word character.
`\s`	Matches whitespace characters like, space, tab, new line etc.
`\S`	Matches any non-whitespace character.
`\d`	Matches any decimal character.
`\D`	Matches any non-decimal character.

Quantifiers could be added to specify number of times a character could appear.

Quantifier	**Description**
`*`	Zero or more matches.
`+`	One or more matches.
`?`	Zero or one matches.
`{N}`	N matches.
`{N,}`	N or more matches.
`{N,M}`	Between N and M matches.

The syntax of the control is as given:

```
<asp:RegularExpressionValidator           ID="string"           runat="server"
ErrorMessage="string"
   ValidationExpression="string" ValidationGroup="string">

</asp:RegularExpressionValidator>
```

CustomValidator

The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation.

The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control's ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

The basic syntax for the control is as given:

```
<asp:CustomValidator ID="CustomValidator1" runat="server"
   ClientValidationFunction=.cvf_func. ErrorMessage="CustomValidator">

</asp:CustomValidator>
```

ValidationSummary

The ValidationSummary control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

The following two mutually inclusive properties list out the error message:

- ShowSummary: shows the error messages in specified format.
- ShowMessageBox: shows the error messages in a separate window.

The syntax for the control is as given:

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
   DisplayMode = "BulletList" ShowSummary = "true" HeaderText="Errors:" />
```

Validation Groups

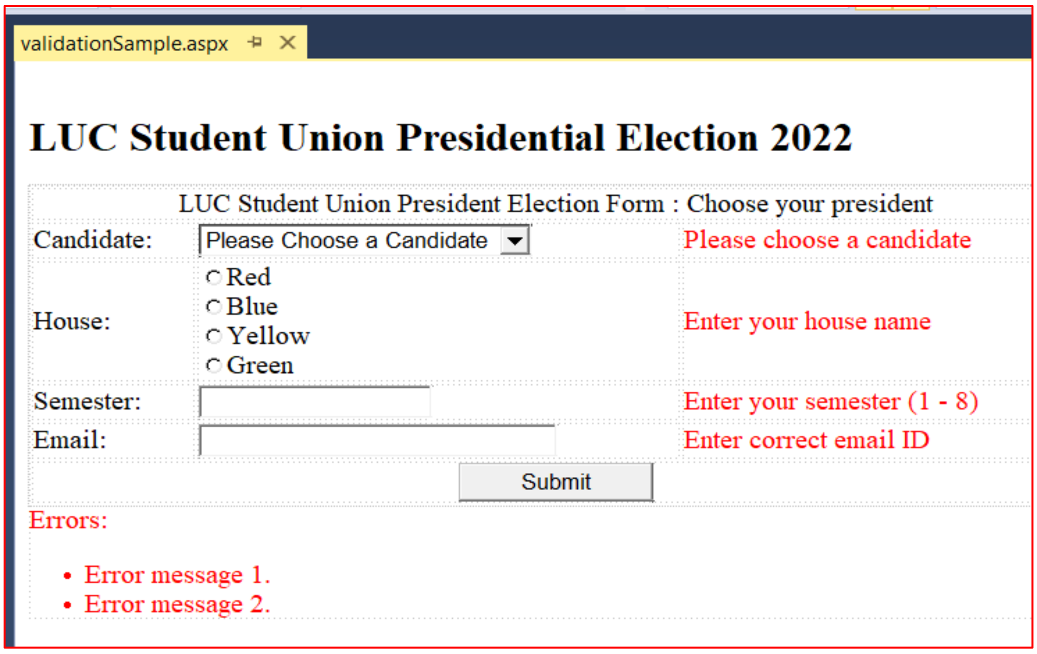
Complex pages have different groups of information provided in different panels. In such situation, a need might arise for performing validation separately for separate group. This kind of situation is handled using validation groups.

To create a validation group, you should put the input controls and the validation controls into the same logical group by setting their ValidationGroup property.

Example 4

The following example describes a form to be filled up by LUC students, divided into four houses, for electing the student union president. Here, we use the validation controls to validate the user input.

This is the form in design view:



LUC Student Union Presidential Election 2022

LUC Student Union President Election Form : Choose your president		
Candidate:	Please Choose a Candidate	Please choose a candidate
House:	○ Red ○ Blue ○ Yellow ○ Green	Enter your house name
Semester:		Enter your semester (1 - 8)
Email:		Enter correct email ID
	Submit	

Errors:

- Error message 1.
- Error message 2.

The content file code is as given:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="validationSample.aspx.cs"
Inherits="LUC_Web_Sample2.validationSample" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <h2>LUC Student Union Presidential Election 2022</h2>
    <form id="form1" runat="server">
       <table style="width: 66%;">
      <tr>
         <td class="style1" colspan="3" align="center">
         <asp:Label ID="lblmsg" Text="LUC Student Union President Election
Form : Choose your president"
            runat="server" />
         </td>
      </tr>

      <tr>
         <td class="style3">
            Candidate:
         </td>

         <td class="style2">
            <asp:DropDownList ID="ddlcandidate" runat="server"
style="width:239px">
               <asp:ListItem>Please Choose a Candidate</asp:ListItem>
               <asp:ListItem>Syikin xxi</asp:ListItem>
               <asp:ListItem>Andrew Sim</asp:ListItem>
               <asp:ListItem>Chan Weng Jun</asp:ListItem>
                <asp:ListItem>Shaufer</asp:ListItem>
               <asp:ListItem>Subavarshini Sivakumar</asp:ListItem>
            </asp:DropDownList>
         </td>

         <td>
            <asp:RequiredFieldValidator ID="rfvcandidate"
               runat="server" ControlToValidate ="ddlcandidate"
               ErrorMessage="Please choose a candidate"
               InitialValue="Please choose a candidate">
            </asp:RequiredFieldValidator>
         </td>
      </tr>

      <tr>
         <td class="style3">
            House:
         </td>

         <td class="style2">
            <asp:RadioButtonList ID="rblhouse" runat="server"
RepeatLayout="Flow">
               <asp:ListItem>Red</asp:ListItem>
               <asp:ListItem>Blue</asp:ListItem>
               <asp:ListItem>Yellow</asp:ListItem>
               <asp:ListItem>Green</asp:ListItem>
            </asp:RadioButtonList>
         </td>

         <td>
```

```
            <asp:RequiredFieldValidator ID="rfvhouse" runat="server"
               ControlToValidate="rblhouse" ErrorMessage="Enter your house
name" ForeColor="Red" ></asp:RequiredFieldValidator>
            <br />
         </td>
      </tr>

      <tr>
         <td class="style3">
            Semester:
         </td>

         <td class="style2">
            <asp:TextBox ID="txtclass" runat="server"></asp:TextBox>
         </td>

         <td>
            <asp:RangeValidator ID="rvclass"
               runat="server" ControlToValidate="txtclass"
               ErrorMessage="Enter your semester (1 - 8)" MaximumValue="8"
               MinimumValue="1" Type="Integer">
            </asp:RangeValidator>
         </td>
      </tr>

      <tr>
         <td class="style3">
            Email:
         </td>

         <td class="style2">
            <asp:TextBox ID="txtemail" runat="server" style="width:250px">
            </asp:TextBox>
         </td>

         <td>
            <asp:RegularExpressionValidator ID="remail" runat="server"
               ControlToValidate="txtemail" ErrorMessage="Enter correct email
ID"
               ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-
.]\w+)*" ForeColor="Red"></asp:RegularExpressionValidator>
         </td>
      </tr>

      <tr>
         <td class="style3" align="center" colspan="3">
            <asp:Button ID="btnsubmit" runat="server" style="text-align:
center; width:140px" Text="Submit" OnClick="btnsubmit_Click"  />
         </td>
      </tr>
   </table>
   <asp:ValidationSummary ID="ValidationSummary1" runat="server"  DisplayMode
="BulletList" ShowSummary ="true" HeaderText="Errors:" />

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

The code behind the submit button:

```
protected void btnsubmit_Click(object sender, EventArgs e)
{
   if (Page.IsValid)
```

```
      {
         lblmsg.Text = "Thank You";
      }
      else
      {
         lblmsg.Text = "Fill up all the fields";
      }
}
```

ACTIVITY

1. List 5 HTML server controls and there corresponding HTML tag
2. Describe client side scripts and client side source code
3. What is the basic syntax of textbox control in ASP.NET?
4. Highlight the common properties of check boxes and radio buttons in ASP.NET
5. What is the basic syntax of drop-down list control in ASP.NET?
6. Describe session state
7. What is the basic function of validators?
8. Highlight with example any 5 validation controls in ASP.NET

KEYWORDS

- HTML Tags
- State
- Validators
- Asp.Net Basic Controls
- Session

SUMMARY

This chapter presents the following topics:

- Introduction to HTML Server
- Advantages of using HTML Server Controls
- Client Side Programming
- Client Side Scripts
- Client Side Source Code
- ASP.NET Basic Controls
- State Management
- Validators
- ValidationSummary
- BaseValidator Class
- Validation Groups