TOPIC 3 → Event Handling and Web Server Side Programming

LEARNING OUTCOME

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

- ➢ Explain Event Handling
- ➢ Apply Application and Session Events
- ➢ Describe Page and Control Events
- ➢ Write Event Handling Using Controls
- ➢ Use Common Control Events
- ➢ Identify Default Events
- ➢ Write Server Side Programming
- ➢ Describe Server Object, Request Object and Response Object
- ➢ Use Server Controls

Introduction

An ASP.NET page is made up of a number of server controls along with HTML controls, text, and images. Sensitive data from the page and the states of different controls on the page are stored in hidden fields that form the context of that page request.
ASP.NET runtime controls the association between a page instance and its state. An ASP.NET page is an object of the Page or inherited from it.

All the controls on the pages are also objects of the related control class inherited from a parent Control class. When a page is run, an instance of the object page is created along with all its content controls.
An ASP.NET page is also a server side file saved with the .aspx extension. It is modular in nature and can be divided into the following core sections:

- Page Directives
- Code Section
- Page Layout
- Page Directives

The page directives set up the environment for the page to run. The @Page directive defines page-specific attributes used by ASP.NET page parser and compiler. Page directives specify how the page should be processed, and which assumptions need to be taken about the page.
It allows importing namespaces, loading assemblies, and registering new controls with custom tag names and namespace prefixes.

Code Section
The code section provides the handlers for the page and control events along with other functions

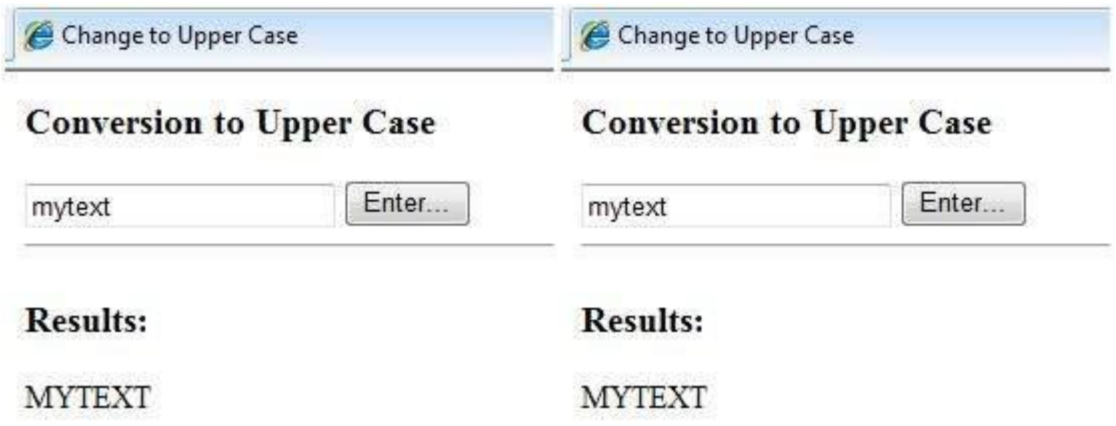
required. We mentioned that, ASP.NET follows an object model. Now, these objects raise events when some events take place on the user interface, like a user clicks a button or moves the cursor. The kind of response these events need to reciprocate is coded in the event handler functions. The event handlers are nothing but functions bound to the controls.
The code section or the code behind file provides all these event handler routines, and other functions used by the developer. The page code could be precompiled and deployed in the form of a binary assembly.

Page Layout
The page layout provides the interface of the page. It contains the server controls, text, inline JavaScript, and HTML tags.

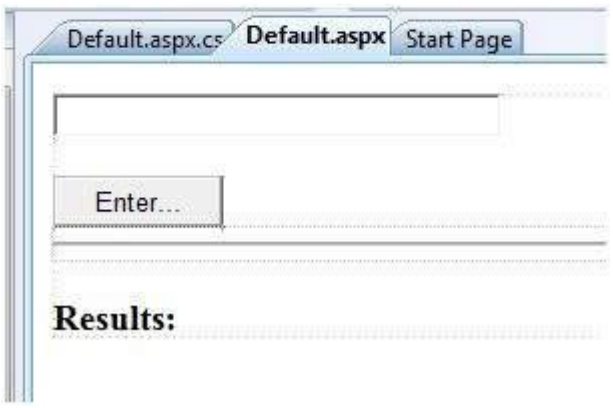
The following code snippet provides a sample ASP.NET page explaining Page directives, code section and page layout written in C#:

```
<!-- directives -->
<% @Page Language="C#" %>

<!-- code section -->
<script runat="server">

   private void convertoupper(object sender, EventArgs e)
   {
      string str = mytext.Value;
      changed_text.InnerHtml = str.ToUpper();
   }
</script>

<!-- Layout -->
<html>
   <head>
      <title> Change to Upper Case </title>
   </head>

   <body>
      <h3> Conversion to Upper Case </h3>

      <form runat="server">
         <input runat="server" id="mytext" type="text" />
         <input  runat="server"  id="button1"  type="submit"  value="Enter..." 
OnServerClick="convertoupper"/>

         <hr />
         <h3> Results: </h3>
         <span runat="server" id="changed_text" />
      </form>

   </body>

</html>
```

Copy this file to the web server root directory. Generally it is c:\iNETput\wwwroot. Open the file from the browser to execute it and it generates following result:



Using Visual Studio IDE
Let us develop the same example using Visual Studio IDE. Instead of typing the code, you can just drag the controls into the design view:



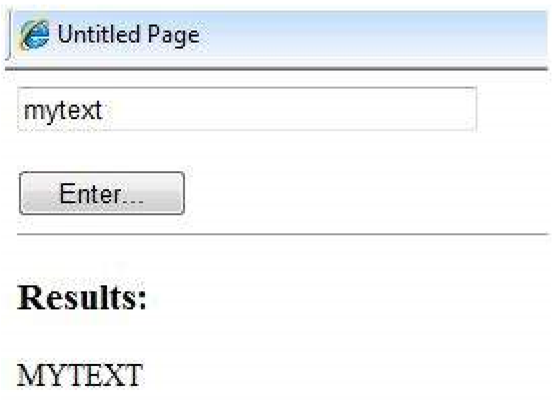
The content file is automatically developed. All you need to add is the Button1_Click routine, which is as follows:

```
protected void Button1_Click(object sender, EventArgs e)
{
   string buf = TextBox1.Text;
   changed_text.InnerHtml = buf.ToUpper();
}
```

The content file code is as given:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="firstexample._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" style="width:224px">
            </asp:TextBox>

            <br />
            <br />

            <asp:Button    ID="Button1"    runat="server"    Text="Enter..."
style="width:85px" onclick="Button1_Click" />
            <hr />

            <h3> Results: </h3>
            <span runat="server" id="changed_text" />
         </div>
      </form>
   </body>
</html>
```

Execute the example by right clicking on the design view and choosing 'View in Browser' from the popup menu. This generates the following result:



Results:

MYTEXT

Event Handling
An event is an action or occurrence such as a mouse click, a key press, mouse movements, or any system-generated notification. A process communicates through events. For example, interrupts are system-generated events. When events occur, the application should be able to respond to it and manage it.

Events in ASP.NET raised at the client machine, and handled at the server machine. For example, a user clicks a button displayed in the browser. A Click event is raised. The browser handles this client-side event by posting it to the server.

The server has a subroutine describing what to do when the event is raised; it is called the event-handler. Therefore, when the event message is transmitted to the server, it checks whether the Click event has an associated event handler. If it has, the event handler is executed.

Event Arguments
ASP.NET event handlers generally take two parameters and return void. The first parameter represents the object raising the event and the second parameter is event argument.
The general syntax of an event is:

```
private void EventName (object sender, EventArgs e);
```

Application and Session Events
The most important application events are:

- Application_Start - It is raised when the application/website is started.
- Application_End - It is raised when the application/website is stopped.

Similarly, the most used Session events are:

- Session_Start - It is raised when a user first requests a page from the application.
- Session_End - It is raised when the session ends.

Page and Control Events
Common page and control events are:

- DataBinding - It is raised when a control binds to a data source.
- Disposed - It is raised when the page or the control is released.
- Error - It is a page event, occurs when an unhandled exception is thrown.
- Init - It is raised when the page or the control is initialized.
- Load - It is raised when the page or a control is loaded.
- PreRender - It is raised when the page or the control is to be rendered.
- Unload - It is raised when the page or control is unloaded from memory.

Event Handling Using Controls
All ASP.NET controls are implemented as classes, and they have events which are fired when a user performs a certain action on them. For example, when a user clicks a button the 'Click' event is generated. For handling events, there are in-built attributes and event handlers. Event handler is coded to respond to an event, and take appropriate action on it.

By default, Visual Studio creates an event handler by including a Handles clause on the Sub procedure. This clause names the control and event that the procedure handles.
The ASP tag for a button control:

```
<asp:Button ID="btnCancel" runat="server" Text="Cancel" />
```

The event handler for the Click event:

```
Protected Sub btnCancel_Click(ByVal sender As Object, ByVal e As
System.EventArgs)
   Handles btnCancel.Click
End Sub
```

An event can also be coded without Handles clause. Then, the handler must be named according to the appropriate event attribute of the control.

The ASP tag for a button control:

```
<asp:Button ID="btnCancel" runat="server" Text="Cancel"
Onclick="btnCancel_Click" />
```

The event handler for the Click event:

```
Protected Sub btnCancel_Click(ByVal sender As Object, ByVal e As
System.EventArgs)
End Sub
```

The common control events are:

Event	Attribute	Controls
Click	OnClick	Button, image button, link button, image map
Command	OnCommand	Button, image button, link button
TextChanged	OnTextChanged	Text box
SelectedIndexChanged	OnSelectedIndexChanged	Drop-down list, list box, radio button list, check box list.
CheckedChanged	OnCheckedChanged	Check box, radio button

Some events cause the form to be posted back to the server immediately, these are called the postback events. For example, the click event such as, Button.Click.
Some events are not posted back to the server immediately, these are called non-postback events.
For example, the change events or selection events such as TextBox.TextChanged or CheckBox.CheckedChanged. The nonpostback events could be made to post back immediately by setting their AutoPostBack property to true.

Default Events
The default event for the Page object is Load event. Similarly, every control has a default event. For example, default event for the button control is the Click event.

The default event handler could be created in Visual Studio, just by double clicking the control in design view. The following table shows some of the default events for common controls:

Control	Default Event
AdRotator	AdCreated
BulletedList	Click
Button	Click
Calender	SelectionChanged
CheckBox	CheckedChanged
CheckBoxList	SelectedIndexChanged
DataGrid	SelectedIndexChanged
DataList	SelectedIndexChanged
DropDownList	SelectedIndexChanged
HyperLink	Click
ImageButton	Click
ImageMap	Click
LinkButton	Click
ListBox	SelectedIndexChanged
Menu	MenuItemClick
RadioButton	CheckedChanged
RadioButtonList	SelectedIndexChanged

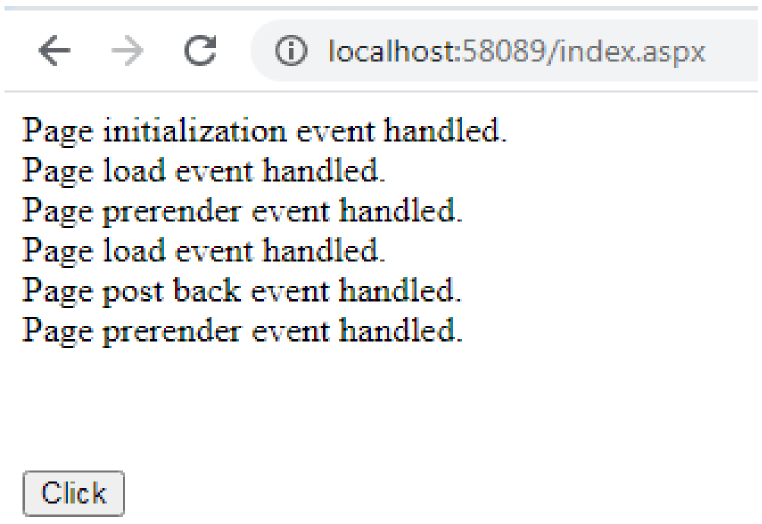
Example
This example includes a simple page with a label control and a button control on it. As the page events such as Page_Load, Page_Init, Page_PreRender etc. take place, it sends a message, which is displayed by the label control. When the button is clicked, the Button_Click event is raised and that also sends a message to be displayed on the label.
Create a new website and drag a label control and a button control on it from the control tool box. Using the properties window, set the IDs of the controls as .lblmessage. and .btnclick. respectively. Set the Text property of the Button control as 'Click'.
The markup file (.aspx):

```
<%@  Page  Language="C#"  AutoEventWireup="true"  CodeBehind="index.aspx.cs"
Inherits="WebExample1.index" %>

<!DOCTYPE html>

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

        <asp:Label ID="lblmessage" runat="server" Text="Label"></asp:Label>
        <br />
        <br />
        <br />
        <asp:Button ID="btnclick" runat="server" Text="Click" />

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

Double click on the design view to move to the code behind file. The Page_Load event is automatically created without any code in it. Write down the following self-explanatory code lines:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebExample1
{
    public partial class index : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            lblmessage.Text += "Page load event handled. <br />";

            if (Page.IsPostBack)
            {
                lblmessage.Text += "Page post back event handled.<br/>";
            }
        }

        protected void Page_Init(object sender, EventArgs e)
        {
            lblmessage.Text += "Page initialization event handled.<br/>";
        }
        protected void Page_PreRender(object sender, EventArgs e)
        {
            lblmessage.Text += "Page prerender event handled. <br/>";
        }
        protected void btnclick_Click(object sender, EventArgs e)
        {
            lblmessage.Text += "Button click event handled. <br/>";
        }

    }
}
```

Execute the page. The label shows page load, page initialization and, the page pre-render events. Click the button to see effect:



```
Page initialization event handled.
Page load event handled.
Page prerender event handled.
Page load event handled.
Page post back event handled.
Page prerender event handled.
```

Server Side Programming
Webpage is instantiated as a control object. All web forms are basically instances of the ASP.NET Page class. The page class has the following extremely useful properties that correspond to intrinsic objects:

- Session
- Application
- Cache
- Request
- Response
- Server
- User
- Trace

We will discuss each of these objects in due time. In this lesson, we will explore the Server object, the Request object, and the Response object.

Server Object
The Server object in Asp.NET is an instance of the `System.Web.HttpServerUtility` class. The `HttpServerUtility` class provides numerous properties and methods to perform various jobs.

Properties and Methods of the Server object
The methods and properties of the `HttpServerUtility` class are exposed through the intrinsic Server object provided by ASP.NET.

The following table provides a list of the properties:

Property	**Description**
MachineName	Name of server computer
ScriptTimeOut	Gets and sets the request time-out value in seconds.

The following table provides a list of some important methods:

Method	Description
CreateObject(String)	Creates an instance of the COM object identified by its ProgID (Programmatic ID).
CreateObject(Type)	Creates an instance of the COM object identified by its Type.
Equals(Object)	Determines whether the specified Object is equal to the current Object.
Execute(String)	Executes the handler for the specified virtual path in the context of the current request.
Execute(String, Boolean)	Executes the handler for the specified virtual path in the context of the current request and specifies whether to clear the QueryString and Form collections.
GetLastError	Returns the previous exception.
GetType	Gets the Type of the current instance.
HtmlEncode	Changes an ordinary string into a string with legal HTML characters.
HtmlDecode	Converts an Html string into an ordinary string.
ToString	Returns a String that represents the current Object.
Transfer(String)	For the current request, terminates execution of the current page and starts execution of a new page by using the specified URL path of the page.
UrlDecode	Converts an URL string into an ordinary string.
UrlEncodeToken	Works same as UrlEncode, but on a byte array that contains Base64-encoded data.
UrlDecodeToken	Works same as UrlDecode, but on a byte array that contains Base64-encoded data.
MapPath	Return the physical path that corresponds to a specified virtual file path on the server.
Transfer	Transfers execution to another web page in the current application.

Request Object
The request object is an instance of the `System.Web.HttpRequest` class. It represents the values and properties of the HTTP request that makes the page loading into the browser.

The information presented by this object is wrapped by the higher level abstractions (the web control model). However, this object helps in checking some information such as the client browser and cookies.

Properties and Methods of the Request Object
The following table provides some noteworthy properties of the Request object:

Property	Description
AcceptTypes	Gets a string array of client-supported MIME accept types.
ApplicationPath	Gets the ASP.NET application's virtual application root path on the server.
Browser	Gets or sets information about the requesting client's browser capabilities.
ContentEncoding	Gets or sets the character set of the entity-body.
ContentLength	Specifies the length, in bytes, of content sent by the client.
ContentType	Gets or sets the MIME content type of the incoming request.
Cookies	Gets a collection of cookies sent by the client.
FilePath	Gets the virtual path of the current request.
Files	Gets the collection of files uploaded by the client, in multipart MIME format.
Form	Gets a collection of form variables.
Headers	Gets a collection of HTTP headers.

HttpMethod	Gets the HTTP data transfer method (such as GET, POST, or HEAD) used by the client.
InputStream	Gets the contents of the incoming HTTP entity body.
IsSecureConnection	Gets a value indicating whether the HTTP connection uses secure sockets (that is, HTTPS).
QueryString	Gets the collection of HTTP query string variables.
RawUrl	Gets the raw URL of the current request.
RequestType	Gets or sets the HTTP data transfer method (GET or POST) used by the client.
ServerVariables	Gets a collection of Web server variables.
TotalBytes	Gets the number of bytes in the current input stream.
Url	Gets information about the URL of the current request.
UrlReferrer	Gets information about the URL of the client's previous request that is linked to the current URL.
UserAgent	Gets the raw user agent string of the client browser.
UserHostAddress	Gets the IP host address of the remote client.
UserHostName	Gets the DNS name of the remote client.
UserLanguages	Gets a sorted string array of client language preferences.

The following table provides a list of some important methods:

Method	**Description**
BinaryRead	Performs a binary read of a specified number of bytes from the current input stream.
Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from object.)
GetType	Gets the Type of the current instance.
MapImageCoordinates	Maps an incoming image-field form parameter to appropriate x-coordinate and y-coordinate values.
MapPath(String)	Maps the specified virtual path to a physical path.
SaveAs	Saves an HTTP request to disk.
ToString	Returns a String that represents the current object.
ValidateInput	Causes validation to occur for the collections accessed through the Cookies, Form, and QueryString properties.

Response Object

The Response object represents the server's response to the client request. It is an instance of the `System.Web.HttpResponse` class.

In ASP.NET, the response object does not play any vital role in sending HTML text to the client, because the server-side controls have nested, object oriented methods for rendering themselves.

However, the `HttpResponse` object still provides some important functionalities, like the cookie feature and the `Redirect()` method. The `Response.Redirect()` method allows transferring the user to another page, inside as well as outside the application. It requires a round trip.

Properties and Methods of the Response Object

The following table provides some noteworthy properties of the Response object:

Property	**Description**
Buffer	Gets or sets a value indicating whether to buffer the output and send it after the complete response is finished processing.
BufferOutput	Gets or sets a value indicating whether to buffer the output and send it after the complete page is finished processing.
Charset	Gets or sets the HTTP character set of the output stream.
ContentEncoding	Gets or sets the HTTP character set of the output stream.

ContentType	Gets or sets the HTTP MIME type of the output stream.
Cookies	Gets the response cookie collection.
Expires	Gets or sets the number of minutes before a page cached on a browser expires.
ExpiresAbsolute	Gets or sets the absolute date and time at which to remove cached information from the cache.
HeaderEncoding	Gets or sets an encoding object that represents the encoding for the current header output stream.
Headers	Gets the collection of response headers.
IsClientConnected	Gets a value indicating whether the client is still connected to the server.
Output	Enables output of text to the outgoing HTTP response stream.
OutputStream	Enables binary output to the outgoing HTTP content body.
RedirectLocation	Gets or sets the value of the Http Location header.
Status	Sets the status line that is returned to the client.
StatusCode	Gets or sets the HTTP status code of the output returned to the client.
StatusDescription	Gets or sets the HTTP status string of the output returned to the client.
SubStatusCode	Gets or sets a value qualifying the status code of the response.
SuppressContent	Gets or sets a value indicating whether to send HTTP content to the client.

The following table provides a list of some important methods:

Method	**Description**
AddHeader	Adds an HTTP header to the output stream. AddHeader is provided for compatibility with earlier versions of ASP.
AppendCookie	Infrastructure adds an HTTP cookie to the intrinsic cookie collection.
AppendHeader	Adds an HTTP header to the output stream.
AppendToLog	Adds custom log information to the InterNET Information Services (IIS) log file.
BinaryWrite	Writes a string of binary characters to the HTTP output stream.
ClearContent	Clears all content output from the buffer stream.
Close	Closes the socket connection to a client.
End	Sends all currently buffered output to the client, stops execution of the page, and raises the EndRequest event.
Equals(Object)	Determines whether the specified object is equal to the current object.
Flush	Sends all currently buffered output to the client.
GetType	Gets the Type of the current instance.
Pics	Appends a HTTP PICS-Label header to the output stream.
Redirect(String)	Redirects a request to a new URL and specifies the new URL.
Redirect(String, Boolean)	Redirects a client to a new URL. Specifies the new URL and whether execution of the current page should terminate.
SetCookie	Updates an existing cookie in the cookie collection.
ToString	Returns a String that represents the current Object.
TransmitFile(String)	Writes the specified file directly to an HTTP response output stream, without buffering it in memory.
Write(Char)	Writes a character to an HTTP response output stream.
Write(Object)	Writes an object to an HTTP response stream.
Write(String)	Writes a string to an HTTP response output stream.
WriteFile(String)	Writes the contents of the specified file directly to an HTTP response output stream as a file block.

WriteFile(String, Boolean)	Writes the contents of the specified file directly to an HTTP response output stream as a memory block.

Example
The following example has a text box control where the user can enter name, a button to send the information to the server, and a label control to display the URL of the client computer.

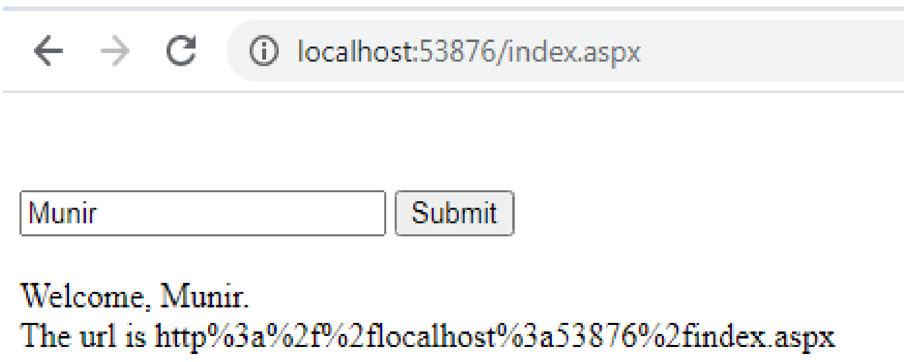
The content file:

```
<%@   Page   Language="C#"   AutoEventWireup="true"   CodeBehind="index.aspx.cs"
Inherits="WebExample2.index"  %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <br />
        <br />
        <asp:TextBox     ID="TextBox1"     placeholder="Enter     your     name"
runat="server"></asp:TextBox>
        <asp:Button ID="Button1" runat="server" Text="Submit" />
        <br />
        <br />
        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
    </div>
    </form>
</body>
</html>
```

The code behind Button1_Click:

```
protected void Button1_Click(object sender, EventArgs e) {

   if (!String.IsNullOrEmpty(TextBox1.Text)) {

      // Access the HttpServerUtility methods through
      // the intrinsic Server object.
      Label1.Text = "Welcome, " + Server.HtmlEncode(TextBox1.Text) + ". <br/>
The url is " + Server.UrlEncode(Request.Url.ToString())
   }
}
```

Run the page to see the following result:



```
Welcome, Munir.
The url is http%3a%2f%2flocalhost%3a53876%2findex.aspx
```

Server Controls
Controls are small building blocks of the graphical user interface, which include text boxes, buttons, check boxes, list boxes, labels, and numerous other tools. Using these tools, the users can enter data, make selections and indicate their preferences.

Controls are also used for structural jobs, like validation, data access, security, creating master pages, and data manipulation.

ASP.NET uses five types of web controls, which are:

- HTML controls
- HTML Server controls
- ASP.NET Server controls
- ASP.NET Ajax Server controls
- User controls and custom controls

ASP.NET server controls are the primary controls used in ASP.NET. These controls can be grouped into the following categories:

- Validation controls - These are used to validate user input and they work by running client-side script.
- Data source controls - These controls provides data binding to different data sources.
- Data view controls - These are various lists and tables, which can bind to data from data sources for displaying.
- Personalization controls - These are used for personalization of a page according to the user preferences, based on user information.
- Login and security controls - These controls provide user authentication.
- Master pages - These controls provide consistent layout and interface throughout the application.
- Navigation controls - These controls help in navigation. For example, menus, tree view etc.
- Rich controls - These controls implement special features. For example, AdRotator, FileUpload, and Calendar control.

The syntax for using server controls is:

```
<asp:controlType       ID    ="ControlID"    runat="server"    Property1=value1
[Property2=value2] />
```

In addition, visual studio has the following features, to help produce error-free coding:

- Dragging and dropping of controls in design view
- IntelliSense feature that displays and auto-completes the properties
- The properties window to set the property values directly

Properties of the Server Controls

ASP.NET server controls with a visual aspect are derived from the WebControl class and inherit all the properties, events, and methods of this class.

The WebControl class itself and some other server controls that are not visually rendered are derived from the System.Web.UI.Control class. For example, PlaceHolder control or XML control.

ASP.Net server controls inherit all properties, events, and methods of the WebControl and System.Web.UI.Control class.

ACTIVITY

1. Highlight application and session events
2. Highlight page and control events
3. Describe event handling as regards ASP.NET
4. Highlight any 5 methods of server object
5. Highlight any 5 properties of request object
6. Describe response object.
7. Highlight any 5 categories of ASP.NET server controls

KEYWORDS

- ➢ Event
- ➢ Server Objects
- ➢ Server Controls
- ➢ Response Objects
- ➢ Session

SUMMARY

This chapter presents the following topics:

- Introduction to Event Handling
- Application and Session Events
- Page and Control Events
- Event Handling Using Controls
- Common Control Events
- Default Events
- Server Side Programming
- Server Object
- Request Object
- Response Object
- Server Controls