LEARN ASP.NET

web application framework

tutorialspoint

SIMPLYEASYLEARNING

About the Tutorial

ASP.NET is a web application framework developed and marketed by Microsoft to allow programmers to build dynamic web sites. It allows you to use a full-featured programming language such as C# or VB.NET to build web applications easily.

This tutorial covers all the basic elements of ASP.NET that a beginner would require to get started.

Audience

This tutorial is prepared for the beginners to help them understand basic ASP.NET programming. After completing this tutorial, you will find yourself at a moderate level of expertise in ASP.NET programming from where you can take yourself to next levels.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of .NET programming language. As we are going to develop web-based applications using ASP.NET web application framework, it will be good if you have an understanding of other web technologies such as HTML, CSS, AJAX, etc.

Disclaimer & Copyright

Contents

1. INTRODUCTION

What is ASP.NET?

ASP.NET is a web development platform, which provides a programming model, a comprehensive software infrastructure and various services required to build up robust web applications for PC as well as mobile devices.

ASP.NET works on top of the HTTP protocol, and uses the HTTP commands and policies to set a browser-to-server bilateral communication and cooperation.

ASP.NET is a part of Microsoft .Net platform. ASP.NET applications are compiled codes, written using the extensible and reusable components or objects present in .Net framework. These codes can use the entire hierarchy of classes in .Net framework.

ASP.NET application codes can be written in any of the following languages:

- C#
- Visual Basic.Net
- Jscript
- J#

ASP.NET is used to produce interactive, data-driven web applications over the internet. It consists of a large number of controls such as text boxes, buttons, and labels for assembling, configuring, and manipulating code to create HTML pages.

ASP.NET Web Forms Model

ASP.NET web forms extend the event-driven model of interaction to the web applications. The browser submits a web form to the web server and the server returns a full markup page or HTML page in response.

All client side user activities are forwarded to the server for stateful processing. The server processes the output of the client actions and triggers the reactions.

Now, HTTP is a stateless protocol. ASP.NET framework helps in storing the information regarding the state of the application, which consists of:

- Page state
- Session state

The page state is the state of the client, i.e., the content of various input fields in the web form. The session state is the collective information obtained from various pages the user visited and worked with, i.e., the overall session state. To clear the concept, let us take an example of a shopping cart:

User adds items to a shopping cart. Items are selected from a page, say the items page, and the total collected items and price are shown on a different page, say the cart page. Only HTTP cannot keep track of all the information coming from various pages. ASP.NET session state and server side infrastructure keeps track of the information collected globally over a session.

ASP.NET runtime carries the page state to and from the server across page requests while generating ASP.NET runtime codes, and incorporates the state of the server side components in hidden fields.

This way, the server becomes aware of the overall application state and operates in a two-tiered connected way.

The ASP.NET Component Model

The ASP.NET component model provides various building blocks of ASP.NET pages. Basically it is an object model, which describes:

- Server side counterparts of almost all HTML elements or tags such as <form> and <input>.
- Server controls, which help in developing complex user-interface. For example, the Calendar control or the Gridview control.

ASP.NET is a technology, which works on the .Net framework that contains all web-related functionalities. The .Net framework is made of an object-oriented hierarchy. An ASP.NET web application is made of pages. When a user requests an ASP.NET page, the IIS delegates the processing of the page to the ASP.NET runtime system.

The ASP.NET runtime transforms the .aspx page into an instance of a class, which inherits from the base class page of the .Net framework. Therefore, each ASP.NET page is an object and all its components i.e., the server-side controls are also objects.

Components of .Net Framework 3.5

Before going to the next session on Visual Studio.Net, let us go through the various components of the .Net framework 3.5. The following table describes the components of the .Net framework 3.5 and the job they perform:

Components and their Description
(1) Common Language Runtime or CLR It performs memory management, exception handling, debugging, security checking, thread execution, code execution, code safety, verification, and compilation. The code that is directly managed by the CLR is called the managed code. When the managed code is compiled, the compiler converts the source code into a CPU independent intermediate language (IL) code. A Just-In-Time (JIT) compiler compiles the IL code into native code, which is CPU specific.
(2) .Net Framework Class Library It contains a huge library of reusable types, classes, interfaces, structures, and enumerated values, which are collectively called types.
(3) Common Language Specification It contains the specifications for the .Net supported languages and implementation of language integration.
(4) Common Type System It provides guidelines for declaring, using, and managing types at runtime, and cross-language communication.
(5) Metadata and Assemblies Metadata is the binary information describing the program, which is either stored in a portable executable file (PE) or in the memory. Assembly is a logical unit consisting of the assembly manifest, type metadata, IL code, and a set of resources like image files.
(6) Windows Forms Windows forms contain the graphical representation of any window displayed in the application.
(7) ASP.NET and ASP.NET AJAX ASP.NET is the web development model and AJAX is an extension of ASP.NET for developing and implementing AJAX functionality. ASP.NET AJAX contains the

components that allow the developer to update data on a website without a complete reload of the page.

(8) ADO.NET

It is the technology used for working with data and databases. It provides access to data sources like SQL server, OLE DB, XML etc. The ADO.NET allows connection to data sources for retrieving, manipulating, and updating data.

(9) Windows Workflow Foundation (WF)

It helps in building workflow-based applications in Windows. It contains activities, workflow runtime, workflow designer, and a rules engine.

(10)Windows Presentation Foundation

It provides a separation between the user interface and the business logic. It helps in developing visually stunning interfaces using documents, media, two and three dimensional graphics, animations, and more.

(11) Windows Communication Foundation (WCF)

It is the technology used for building and executing connected systems.

(12) Windows CardSpace

It provides safety for accessing resources and sharing personal information on the internet.

(13) LINQ

It imparts data querying capabilities to .Net languages using a syntax which is similar to the tradition query language SQL.

2. ENVIRONMENT SETUP

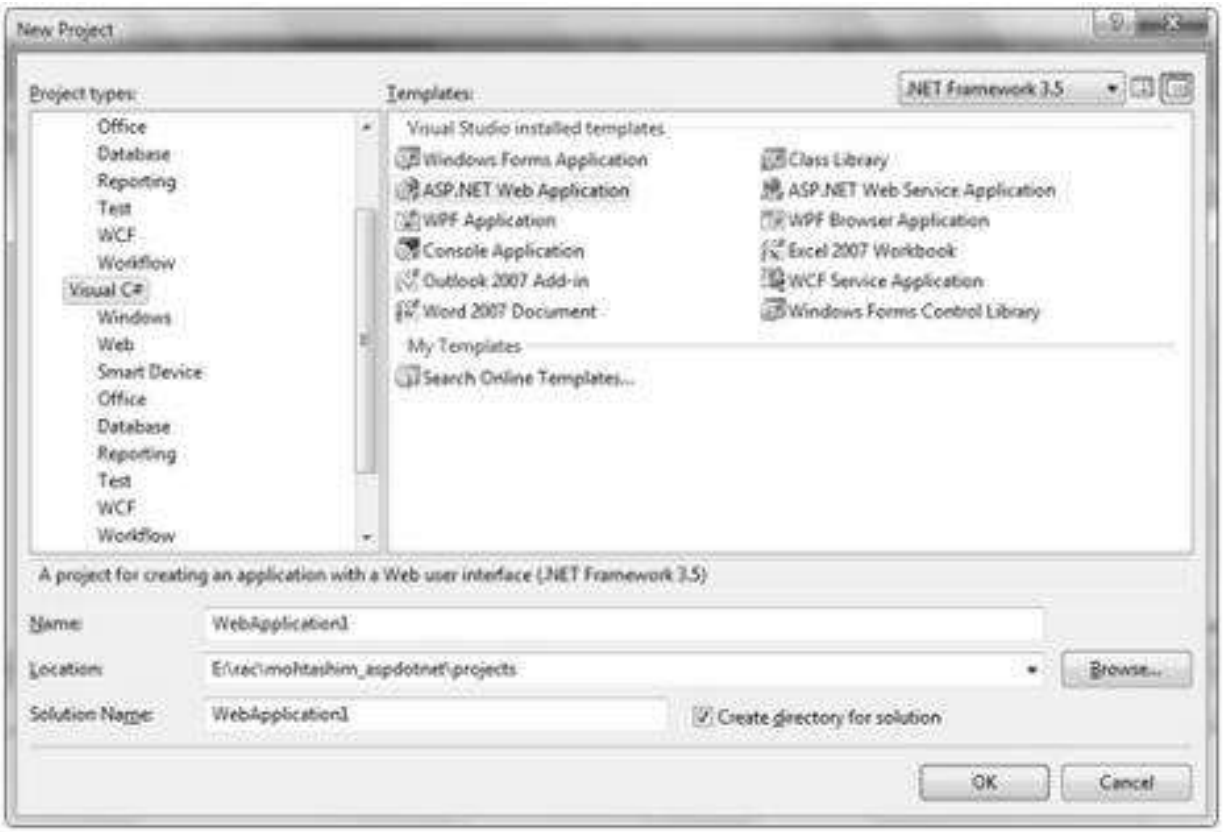
ASP.NET provides an abstraction layer on top of HTTP on which the web applications are built. It provides high-level entities such as classes and components within an object-oriented paradigm.

The key development tool for building ASP.NET applications and front ends is Visual Studio. In this tutorial, we work with Visual Studio 2008.

Visual Studio is an integrated development environment for writing, compiling, and debugging the code. It provides a complete set of development tools for building ASP.NET web applications, web services, desktop applications, and mobile applications.

The Visual Studio IDE

The new project window allows choosing an application template from the available templates.



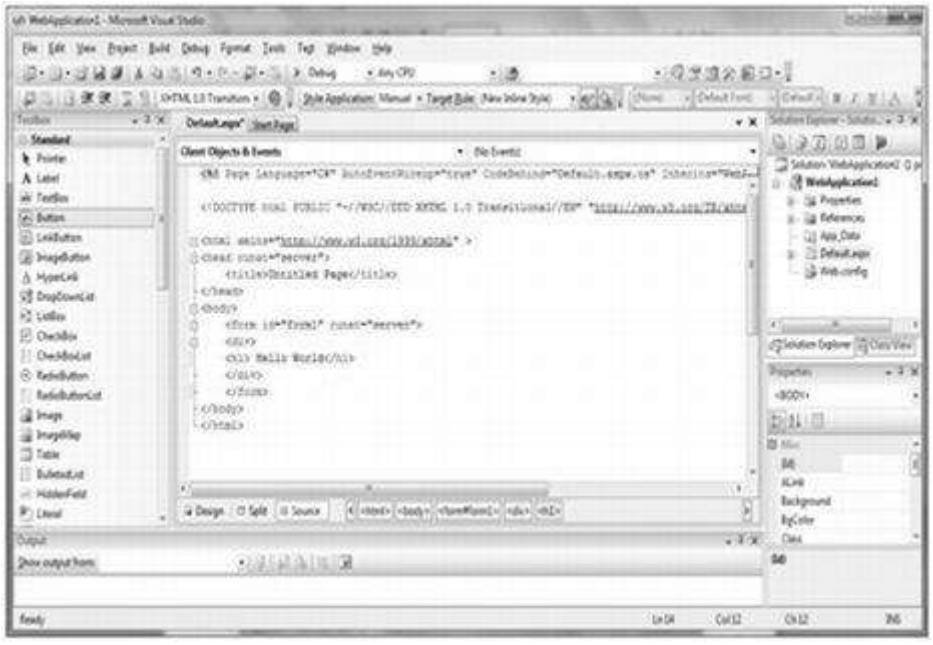
When you start a new web site, ASP.NET provides the starting folders and files for the site, including two files for the first web form of the site.

The file named Default.aspx contains the HTML and asp code that defines the form, and the file named Default.aspx.cs (for C# coding) or the file named Default.aspx.vb (for VB coding) contains the code in the language you have chosen and this code is responsible for the actions performed on a form.

The primary window in the Visual Studio IDE is the Web Forms Designer window. Other supporting windows are the Toolbox, the Solution Explorer, and the Properties window. You use the designer to design a web form, to add code to the control on the form so that the form works according to your need, you use the code editor.

Working with Views and Windows

- You can work with windows in the following ways:
- To change the Web Forms Designer from one view to another, click on the Design or source button.
- To close a window, click on the close button on the upper right corner and to redisplay, select it from the View menu.
- To hide a window, click on its Auto Hide button. The window then changes into a tab. To display again, click the Auto Hide button again.
- To change the size of a window, just drag it.



Adding Folders and Files to your wWebsite

When a new web form is created, Visual Studio automatically generates the starting HTML for the form and displays it in Source view of the web forms designer. The Solution Explorer is used to add any other files, folders or any existing item on the web site.

- To add a standard folder, right-click on the project or folder under which you are going to add the folder in the Solution Explorer and choose New Folder.
-
- To add an ASP.NET folder, right-click on the project in the Solution Explorer and select the folder from the list.
-
- To add an existing item to the site, right-click on the project or folder under which you are going to add the item in the Solution Explorer and select from the dialog box.

Projects and Solutions

A typical ASP.NET application consists of many items: the web content files (.aspx), source files (.cs files), assemblies (.dll and .exe files), data source files (.mdb files), references, icons, user controls and miscellaneous other files and folders. All these files that make up the website are contained in a Solution.

When a new website is created, VB2008 automatically creates the solution and displays it in the solution explorer.

Solutions may contain one or more projects. A project contains content files, source files, and other files like data sources and image files. Generally, the contents of a project are compiled into an assembly as an executable file (.exe) or a dynamic link library (.dll) file.

Typically a project contains the following content files:

- Page file (.aspx)
- User control (.ascx)
- Web service (.asmx)
- Master page (.master)
- Site map (.sitemap)
- Website configuration file (.config)

Building and Running a Project

You can execute an application by:

- Selecting Start
- Selecting Start Without Debugging from the Debug menu,
- pressing F5
- Ctrl-F5

The program is built meaning, the .exe or the .dll files are generated by selecting a command from the Build menu.

3. LIFE CYCLE

ASP.NET life cycle specifies how:

- ASP.NET processes pages to produce dynamic output
- The application and its pages are instantiated and processed
- ASP.NET compiles the pages dynamically

ASP.NET life cycle could be divided into two groups:

- Application Life Cycle
- Page Life Cycle

ASP.NET Application Life Cycle

The application life cycle has the following stages:

1. User makes a request for accessing application resource, a page. Browser sends this request to the web server.
2. A unified pipeline receives the first request and the following events take place:
    i. An object of the class ApplicationManager is created.
    ii. An object of the class HostingEnvironment is created to provide information regarding the resources.
    iii. Top level items in the application are compiled.
3. Response objects are created. The application objects such as HttpContext, HttpRequest and HttpResponse are created and initialized.
4. An instance of the HttpApplication object is created and assigned to the request.
5. The request is processed by the HttpApplication class. Different events are raised by this class for processing the request.

ASP.NET Page Life Cycle

When a page is requested, it is loaded into the server memory, processed, and sent to the browser. Then it is unloaded from the memory. At each of these steps, methods and events are available, which could be overridden according to the need of the application. In other words, you can write your own code to override the default code.

The Page class creates a hierarchical tree of all the controls on the page. All the components on the page, except the directives, are part of this control tree. You can

see the control tree by adding trace= "true" to the page directive. We will cover page directives and tracing under 'directives' and 'event handling'.

The page life cycle phases are:

- Initialization
- Instantiation of the controls on the page
- Restoration and maintenance of the state
- Execution of the event handler codes
- Page rendering

Understanding the page cycle helps in writing codes for making some specific thing happen at any stage of the page life cycle. It also helps in writing custom controls and initializing them at right time, populate their properties with view-state data and run control behavior code.

Following are the different stages of an ASP.NET page:

Page request

When ASP.NET gets a page request, it decides whether to parse and compile the page, or there would be a cached version of the page; accordingly the response is sent.

Starting of page life cycle

At this stage, the Request and Response objects are set. If the request is an old request or post back, the IsPostBack property of the page is set to true. The UICulture property of the page is also set.

Page initialization

At this stage, the controls on the page are assigned unique ID by setting the UniqueID property and the themes are applied. For a new request, postback data is loaded and the control properties are restored to the view-state values.

Page load

At this stage, control properties are set using the view state and control state values.

Validation

Validate method of the validation control is called and on its successful execution, the IsValid property of the page is set to true.

Postback event handling.

If the request is a postback (old request), the related event handler is invoked.

Page rendering

At this stage, view state for the page and all controls are saved. The page calls the Render method for each control and the output of rendering is written to the OutputStream class of the Response property of Page.

Unload

The rendered page is sent to the client and page properties, such as Response and Request, are unloaded and all cleanup done.

ASP.NET Page Life Cycle Events

At each stage of the page life cycle, the page raises some events, which could be coded. An event handler is basically a function or subroutine, bound to the event, using declarative attributes such as Onclick or handle.

Following are the page life cycle events:

PreInit

PreInit is the first event in page life cycle. It checks the IsPostBack property and determines whether the page is a postback. It sets the themes and master pages, creates dynamic controls, and gets and sets profile property values. This event can be handled by overloading the OnPreInit method or creating a Page_PreInit handler.

Init

Init event initializes the control property and the control tree is built. This event can be handled by overloading the OnInit method or creating a Page_Init handler.

InitComplete

InitComplete event allows tracking of view state. All the controls turn on view-state tracking.

LoadViewState

LoadViewState event allows loading view state information into the controls.

LoadPostData

During this phase, the contents of all the input fields are defined with the <form> tag are processed.

PreLoad

PreLoad occurs before the post back data is loaded in the controls. This event can be handled by overloading the OnPreLoad method or creating a Page_PreLoad handler.

Load

The Load event is raised for the page first and then recursively for all child controls. The controls in the control tree are created. This event can be handled by overloading the OnLoad method or creating a Page_Load handler.

LoadComplete

The loading process is completed, control event handlers are run, and page validation takes place. This event can be handled by overloading the OnLoadComplete method or creating a Page_LoadComplete handler.

PreRender

The PreRender event occurs just before the output is rendered. By handling this event, pages and controls can perform any updates before the output is rendered.

PreRenderComplete

As the PreRender event is recursively fired for all child controls, this event ensures the completion of the pre-rendering phase.

SaveStateComplete

State of control on the page is saved. Personalization, control state and view state information is saved. The HTML markup is generated. This stage can be handled by overriding the Render method or creating a Page_Render handler.

UnLoad

The UnLoad phase is the last phase of the page life cycle. It raises the UnLoad event for all controls recursively and lastly for the page itself. Final cleanup is done and all resources and references, such as database connections, are freed. This event can be handled by modifying the OnUnLoad method or creating a Page_UnLoad handler.

4. FIRST EXAMPLE

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

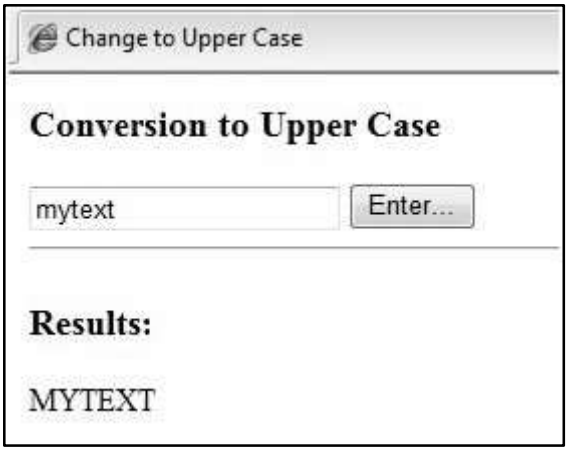
The page layout provides the interface of the page. It contains the server controls, text, inline JavaScript, and HTML tags.

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 />
```

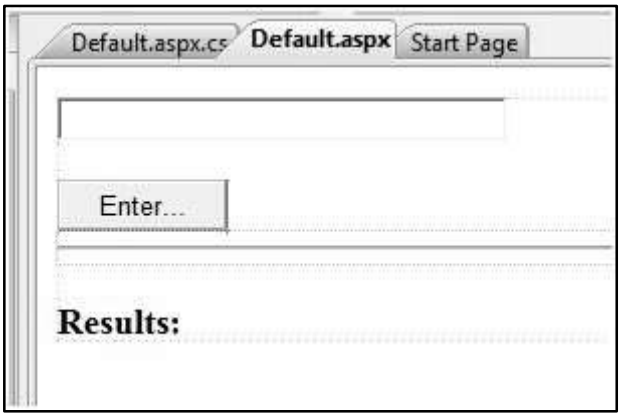
```
<h3> Results: </h3>

<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 the 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:

5. 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 are 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="Default.aspx.cs"

                          Inherits="eventdemo._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:Label ID="lblmessage" runat="server" >

     </asp:Label>

        <br />

        <br />

        <br />
```

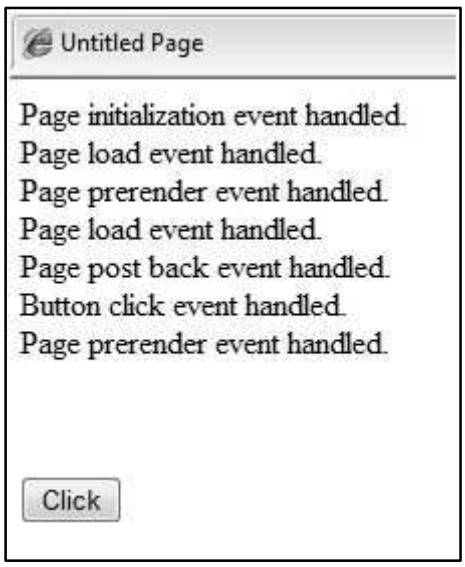
```
     <asp:Button ID="btnclick" runat="server" Text="Click"

                 onclick="btnclick_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;

using System.Configuration;

using System.Data;

using System.Linq;

using System.Web;

using System.Web.Security;

using System.Web.UI;

using System.Web.UI.HtmlControls;

using System.Web.UI.WebControls;

using System.Web.UI.WebControls.WebParts;

using System.Xml.Linq;


namespace eventdemo

{

public partial class _Default : 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:


Untitled Page
Page initialization event handled.
Page load event handled.
Page prerender event handled.
Page load event handled.
Page post back event handled.
Button click event handled.
Page prerender event handled.
Click

6. SERVER SIDE

We have studied the page life cycle and how a page contains various controls. The page itself 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 tutorial, 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.