UNIT-I Introduction of ASP.NET

*Introduction-

ASP.NET is the new offering for web developers from Microsoft. It is not simply the next-generation of ASP; in fact, it is a completely re-engineered and enhanced technology that offers much, much more than traditional ASP and can increase productivity significantly.

*Looking Back -: ASP-(Active Server Pages)

Microsoft's*Active Server Pages (ASP)* is ***a server-side scripting technology to create and run dynamic web applications.***

ASP is a technology that Microsoft created to ease the development of interactive web applications.

With ASP you can use client-side scripts as well as server-side scripts.

Problems with Traditional ASP

There are many problems with ASP if you think of needs for Today's powerful Web applications.

1. Interpreted and Loosely-Typed Code

2. Mixed layout (HTML Source Code) and coding logic (scripting code)

3. Limited Development and Debugging Tools

4. Limited support for state management

5. Supports Partial OOP features

6. Poor Error Handling

7. Does not having in built support for XML

*ASP.Net -(Active Server Pages.NET)

ASP.NET was developed in *direct response to the problems that developers had with **classic ASP***.

ASP.Net *is a server side web technology to create and run dynamic, interactive & high performance web server applications*.

According to **Microsoft**, "***ASP.NET is a technology for building powerful, dynamic Web applications and is part of the .NET Framework***". In fact, ASP.NET is a programming framework used to develop web applications and web services. Basically, it is the next version of ASP.

Advantages of ASP.NET

1. **Separation of Code from HTML:**
   To make a clean sweep, with ASP.NET you have the ability to completely separate design layout and business logic. This makes it much easier for teams of programmers and designers to collaborate efficiently.

2. **Support for compiled languages:**
   Developer can use C#, VB, J#, F#, Delphi etc. And access features such as strong typing and object-oriented programming.

   Using compiled languages also means that ASP.NET pages do not suffer the performance penalties associated with interpreted code.

   ASP.NET pages are precompiled to byte-code and **Just In Time (JIT)** compiled when first requested. Subsequent requests are directed to the fully compiled code, which is cached until the source changes.

3. **Use services provided by the .NET Framework:**
   The .NET Framework provides 2000+ classes in **FCL** that can be used by your application. Some of the key classes help you with input/output, access to operating system services, data access, or even debugging. We will go into more detail on some of them in this module.

4. **Graphical Development Environment:**
   Visual Studio .NET provides a very rich development environment for web developers. You can drag and drop controls and set properties the way you do in C#. And you have full IntelliSense support, not only for your code, but also for HTML and XML.

5. **State Management:**
   To refer to the problems mentioned before, ASP.NET provides solutions for session and application state management. ***State Mgt.is storing user specific information in memory or stored in a database***. It can be shared across web forms, and state information can be recovered, even if the server fails or the connection breaks down.

   Provides way to maintain state at both client and server side.

6. **XML-Based Configuration Files:**
   Configuration settings in ASP.NET are stored in XML files that you can easily read and edit. You can also easily copy these to another server, along with the other files that comprise your application.
7. **Supports All OOP's Features.**
8. **Provides full proof Error handling.**
9. **Provides High Security by supporting in built login controls to perform authentication and authorization.**

*ASP.Net Architecture and its Components –

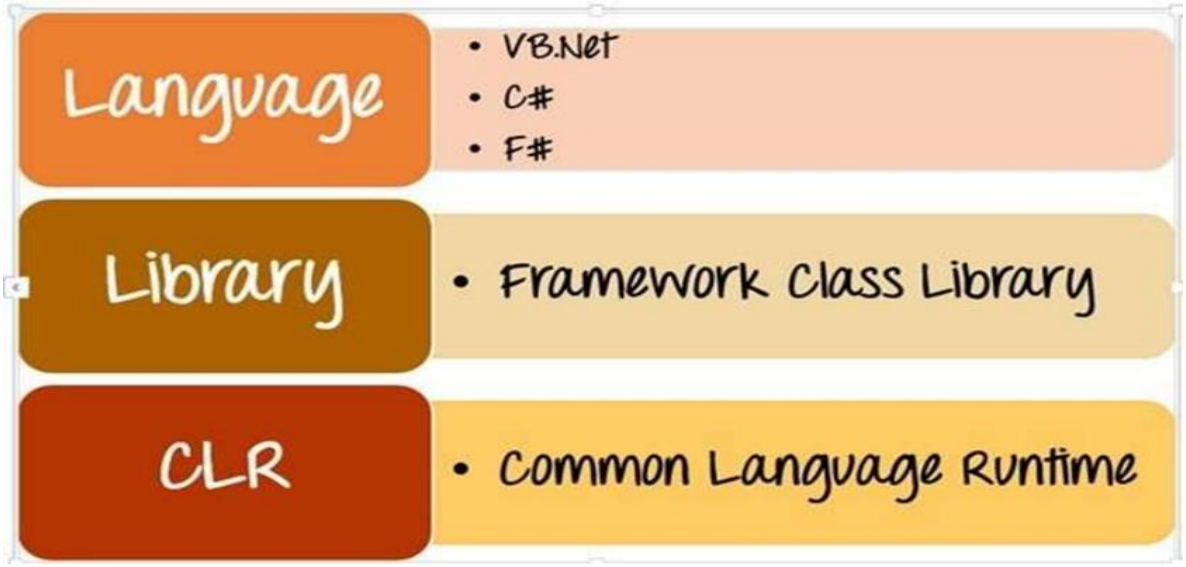


Fig: ASP.Net Architecture

The architecture of the.Net framework is based on the following key components

1. **Language**–Avarietyoflanguagesexistsfor.netframework.Theyare VB.net and C#. These can be used to develop webapplications.

2. **Library** - The .NET Framework includes a set of standard classlibraries.Themostcommonlibraryusedforwebapplicationsin .net is the Web library. The web library has all the necessarycomponentsusedtodevelop.Netweb-basedapplications.

3. **Common Language Runtime** - The Common Language Infrastructure orCLI is a platform. .Net programs are executed on this platform. The CLRis used for performing key activities. Activities include Exceptionhandling and Garbage collection.

*Compilation Technique of ASP.Net-

Modern software programming languages (like C# and VB.NET) utilize a human-friendly syntax that is not directly understandable by computers.

Software commands in this human-friendly syntax are referred to as Source Code. Before a computer can execute the source code, special programs called compilers must rewrite it into machine instructions, also known as object code.

This process (commonly referred to simply as "compilation") can be done explicitly or implicitly.

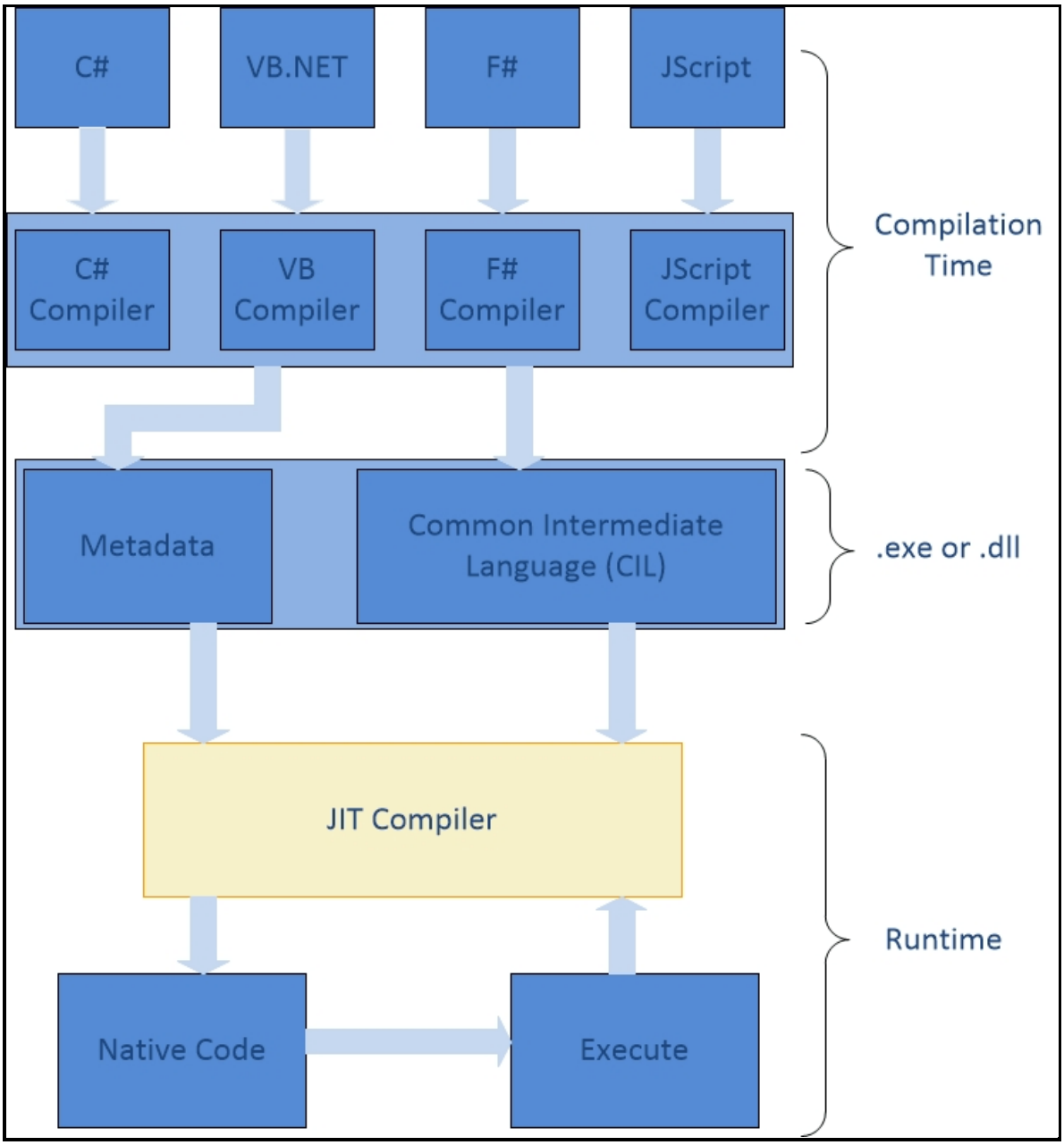
ASP.Net Page compilation is a two-step process.

The first step is converting the source code to intermediate language (IL) by a language-specific compiler.

The second step is converting the IL to machine instructions.

The main difference with the explicit compilers is that only executed fragments of IL code are compiled into machine instructions, at runtime. The .NET framework calls this compiler the JIT (Just-In-Time) compiler.

The JIT compiler is part of the Common Language Runtime (CLR). The CLR manages the execution of all .NET applications. In addition to JIT compilation at runtime, the CLR is also responsible for garbage collection, type safety and for exception handling.



ASP.NET Compilation Process

Different machine configurations use different machine level instructions.
As above Figure shows, the source code is compiled to *exe* or *dll* by the .NET compiler.

Common Intermediate Language (CIL) consists of instructions that any environment supporting .NET can execute and includes metadata describing structures of both data and code.

The JIT Compiler processes the CIL instructions into machine code specific for an environment.

Program portability is ensured by utilizing CIL instructions in the source code.

The JIT compiler compiles only those methods called at runtime.

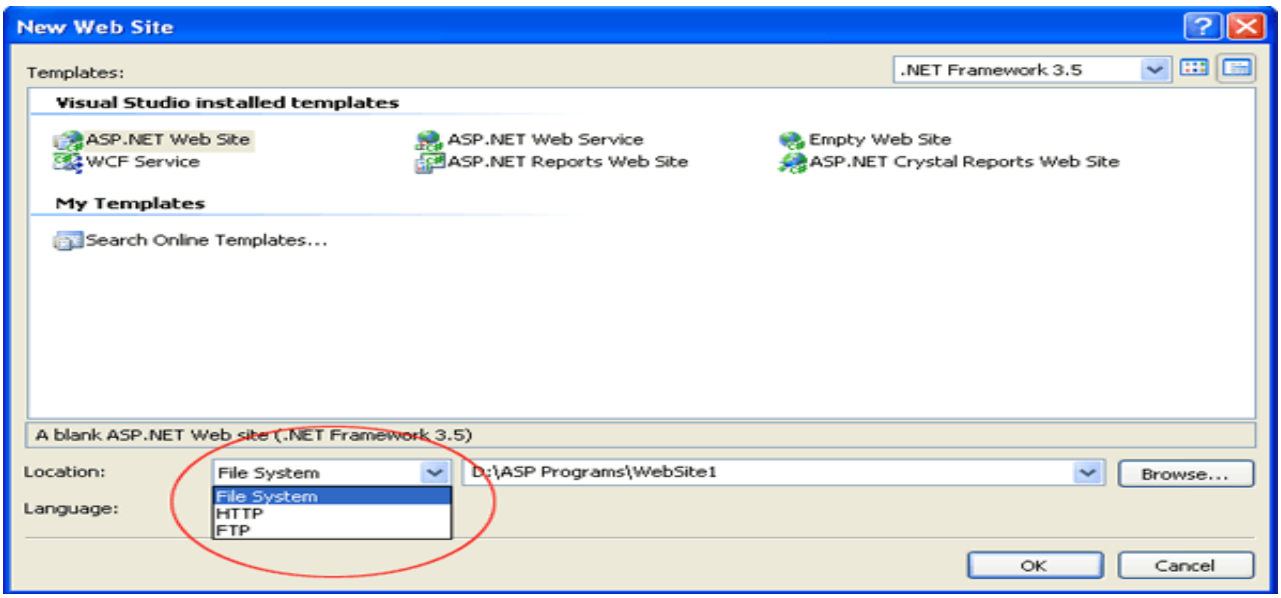
It also keeps track of any variable or parameter passed through methods and enforces type-safety in the runtime environment of the .NET Framework.

*Application Location Options-

ASP.Net Web Application can be stored either one of following Location Option.

Following are the location of your Web project:

1. File system
2. HTTP
3. FTP



1.File System

The File System option creates the new website in a location on your hard drive or on your shared network drive.

Using such a file system Web site means that you do not need to create your site as an Internet Information Services (IIS) application to develop or test it.

File systems Web sites are particularly useful in the following situations:

- When you do not want to (or cannot) install IIS on your development computer.
- In classroom settings, where students can store files on student-specific folders on a central server.
- In a team setting, where team members can access a common Web site on a central server.

File System is used till the process of development, after development it is used in FTP/HTTP server.

2.HTTP

An HTTP based Web site is used when you are working with a site deployed inside of IIS (either locally or on a remote server).

This Web site may be configured at the root of the IIS Web server or in a virtual directory that is configured as an application.

Note: A remote server running IIS will have to provide access to your Web files using Front page Server extension.

In this application will be created under IIS (Internet Information Services) server.

In HTTP your web application is stored under IIS home directory. You will not get any instance of ASP.NET Development server. No port is specified. The request will be handled at IIS port only.

3.FTP

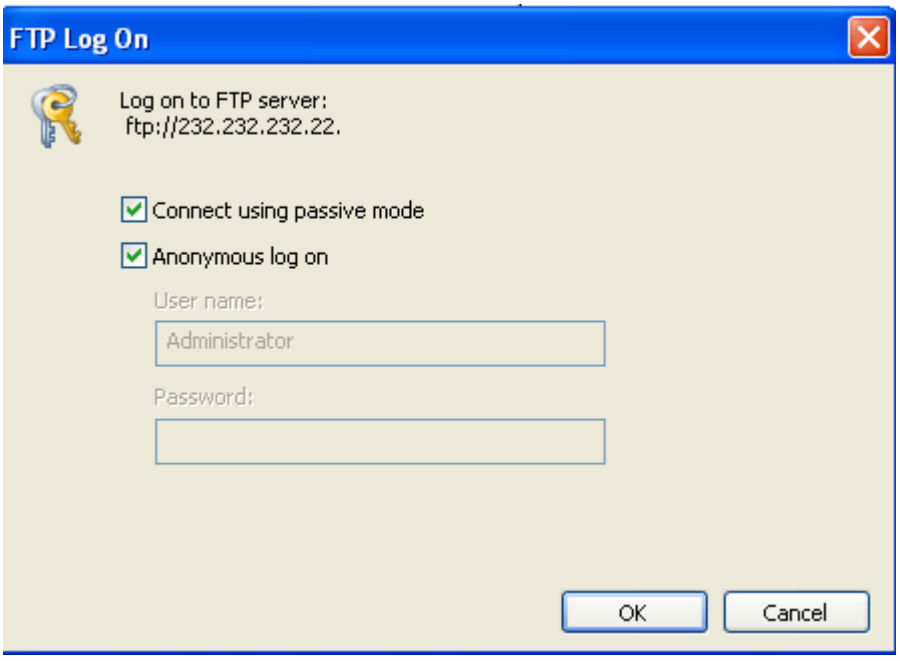
FTP location provides direct facility of creating application at FTP.

The FTP based Web site is useful when you want to connect to your web site via FTP to manage your files on a remote server.

This option is typically used when your website is hosted on a remote server computer and your access to the files and folders on that server is through FTP.

Selecting FTP is essentially the same but creates the site on a server that is accessed using the FTP (File Transfer Protocol).

Choosing HTTP or FTP generally requires that you have a username and password for a location on a server provided by a web host.



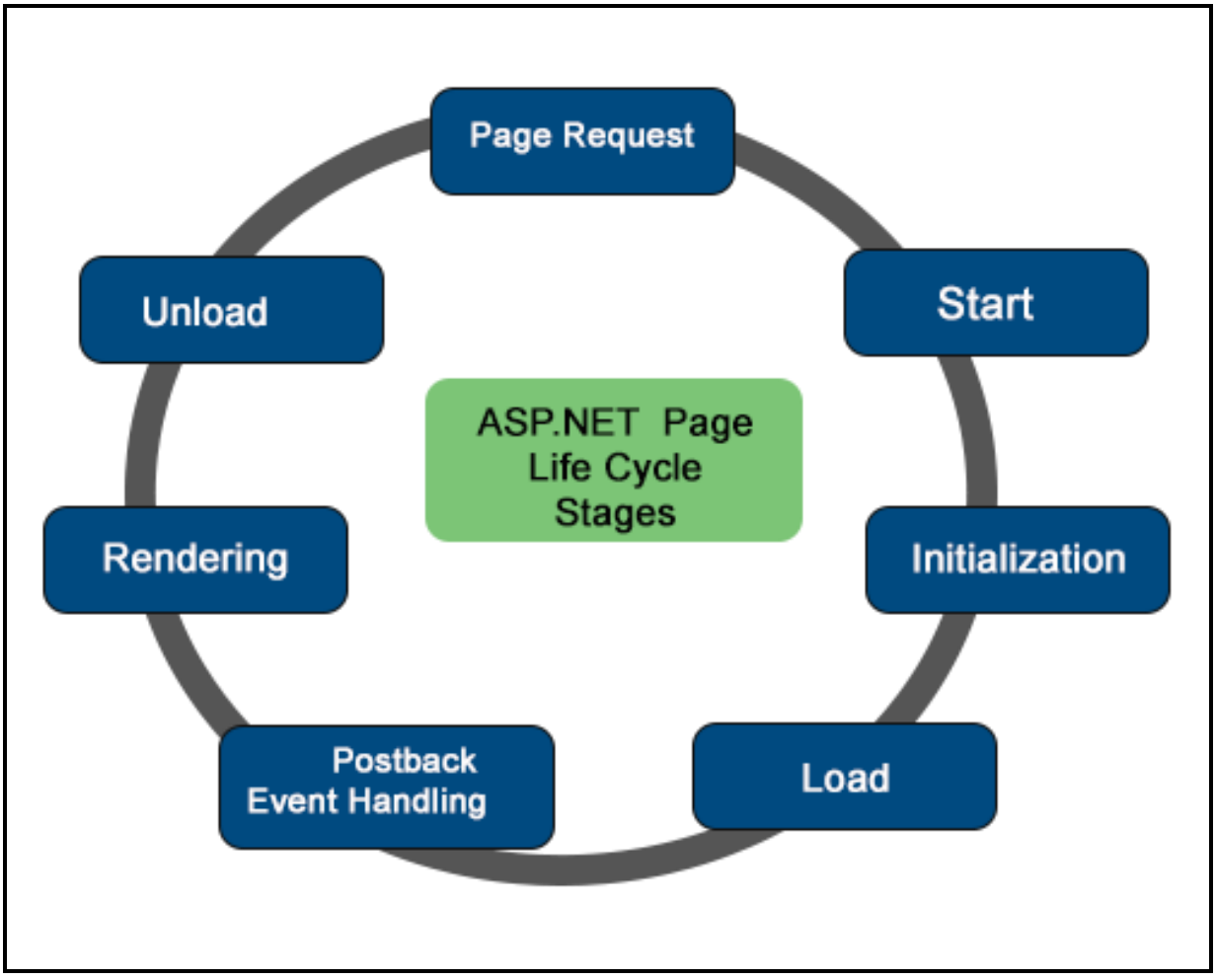
FTP by default is not secure. When a user login to a server his/her information is sent as plain text.

*Web Page and Web Site life cycle-

When an ASP.NET page runs, the page goes through a life cycle in which it performs a series of processing steps.

These include initialization, instantiating controls, restoring and maintaining state, running event handler code, and rendering. It is important for you to understand the page life cycle so that you can write code at the appropriate life-cycle stage for the effect you intend.

Some parts of the life cycle occur only when a page is processed as a postback. For postbacks, the page life cycle is the same during a partial-page postback (as when you use an UpdatePanel control) as it is during a full-page postback.



Stage	Description
Page request	The page request occurs before the page life cycle begins. When the page is requested by a user, ASP.NET determines whether the page needs to be parsed and compiled (therefore beginning the life of a page), or whether a cached version of the page can be sent in response without running the page.
Start	In the start stage, page properties such as Request and Response are set. At this stage, the page also determines whether the request is a postback or a new request and sets the IsPostBack property. The page also sets the UICulture property.
Initialization	During page initialization, controls on the page are available and each control's UniqueID property is set. A master page and themes are also applied to the page if applicable. If the current request is a postback, the postback data has not yet been loaded and control property values have not been restored to the values from view state.
Load	During load, if the current request is a postback, control properties are loaded with information recovered from view state and control state.
Postback event handling	If the request is a postback, control event handlers are called. After that, the Validate method of all validator controls is called, which sets the IsValid property of individual validator controls and of the page
Rendering	Before rendering, view state is saved for the page and all controls. During the rendering stage, the page calls the Render method for each control, providing a text writer that writes its output to the OutputStream object of the page's Response property.
Unload	The Unload event is raised after the page has been fully rendered, sent to the client, and is ready to be discarded. At this point, page properties such as Response and Request are unloaded and cleanup is performed.

Life-Cycle Events

Within each stage of the life cycle of a page, the page raises events that you can handle to run your own code. For control events, you bind the event handler to the event, either declaratively using attributes such as onclick, or in code.

Pages also support automatic event wire-up, meaning that ASP.NET looks for methods with particular names and automatically runs those methods when certain events are raised. If the **AutoEventWireup** attribute of the @ Page directive is set to true, page events are automatically bound to methods that use the naming convention of **Page_event, such as Page_Load and Page_Init**..

The following table lists the page life-cycle events that you will use most frequently.

Page Event	Typical Use
PreInit	Raised after the start stage is complete and before the initialization stage begins. Use this event for the following: • Check the IsPostBack property to determine whether this is the first time the page is being processed. The IsCallback and IsCrossPagePostBack properties have also been set at this time. • Create or re-create dynamic controls. • Set a master page dynamically. • Set the Theme property dynamically. • Read or set profile property values. **Note** If the request is a postback, the values of the controls have not yet been restored from view state. If you set a control property at this stage, its value might be overwritten in the next event.
Init	Raised after all controls have been initialized and any skin settings have been applied. The Init event of individual

Page Event	Typical Use
	controls occurs before the Init event of the page. Use this event to read or initialize control properties.
InitComplete	Raised at the end of the page's initialization stage. Only one operation takes place between the Init and InitComplete events: tracking of view state changes is turned on. View state tracking enables controls to persist any values that are programmatically added to the ViewState collection. Until view state tracking is turned on, any values added to view state are lost across postbacks. Controls typically turn on view state tracking immediately after they raise their Init event. Use this event to make changes to view state that you want to make sure are persisted after the next postback.
PreLoad	Raised after the page loads view state for itself and all controls, and after it processes postback data that is included with the Request instance.
Load	The Page object calls the OnLoad method on the Page object, and then recursively does the same for each child control until the page and all controls are loaded. The Load event of individual controls occurs after the Load event of the page. Use the OnLoad event method to set properties in controls and to establish database connections.
Control events	Use these events to handle specific control events, such as a Button control's Click event or a TextBox control's TextChanged event. **Note** In a postback request, if the page contains validator controls, check the **IsValid** property of the **Page** and of individual validation controls before performing any processing.

Page Event	Typical Use
LoadComplete	Raised at the end of the event-handling stage. Use this event for tasks that require that all other controls on the page be loaded.
PreRender	Raised after the Page object has created all controls that are required in order to render the page, including child controls of composite controls. (To do this, the Page object calls EnsureChildControls for each control and for the page.) The Page object raises the PreRender event on the Page object, and then recursively does the same for each child control. The PreRender event of individual controls occurs after the PreRender event of the page. Use the event to make final changes to the contents of the page or its controls before the rendering stage begins.
PreRenderComplete	Raised after each data bound control whose DataSourceID property is set calls its DataBind method. For more information, see Data Binding Events for Data-Bound Controls later in this topic.
SaveStateComplete	Raised after view state and control state have been saved for the page and for all controls. Any changes to the page or controls at this point affect rendering, but the changes will not be retrieved on the next postback.
Render	This is not an event; instead, at this stage of processing, the Page object calls this method on each control. All ASP.NET Web server controls have a Render method that writes out the control's markup to send to the browser. If you create a custom control, you typically override this method to output the control's markup. However, if your custom control incorporates only standard ASP.NET Web server controls and no custom markup, you do not need to override the Render method. For more information,

Page Event	Typical Use
	see Developing Custom ASP.NET Server Controls. A user control (an .ascx file) automatically incorporates rendering, so you do not need to explicitly render the control in code.
Unload	Raised for each control and then for the page. In controls, use this event to do final cleanup for specific controls, such as closing control-specific database connections. For the page itself, use this event to do final cleanup work, such as closing open files and database connections, or finishing up logging or other request-specific tasks. **Note** During the unload stage, the page and its controls have been rendered, so you cannot make further changes to the response stream. If you attempt to call a method such as the Response.Write method, the page will throw an exception.

*ASP.Net Page Structure

The main components of an ASP.NET page are:

1. Directives
2. Code Declaration Blocks
3. ASP.NET Controls
4. Code Render Blocks
5. Server-Side Comments
6. Server-Side Include Directives
7. Literal Text and HTML Tags.

1. Directives

A directive controls how the page is compiled. It is marked by the tags, <%@ and %>.

It can appear anywhere in a page. But, normally it is placed at the top of a page. The main types of directives are:

- Page
- Import

A Page directive is used to specify the default programming language for a page.

<%@ Page Language="C#" %>

OR

<%@ Language="C#" %>

Some namespaces are imported into an ASP.NET page by default. If you wish to use a class that is not contained in the default namespaces, you must import its namespace.
<%@ Import Namespace="System.Data.SqlClient" %>

2. Code Declaration Blocks

A code declaration block contains all the application logic for a page. It also includes declarations of global variables, and functions.

It must be written within the script runat= "server" tag.
The <script> tag has two optional parameters:
• Language: You can specify the programming language to be used within the <script> tag. Otherwise, the language specified in the Page directive is used.
• SRC: You can specify an external file that contains the code block.
There is no difference in output between writing the code on the same page and including an external script file.

3. ASP.NET Controls

ASP.NET controls can be mixed with text and static HTML in a page.

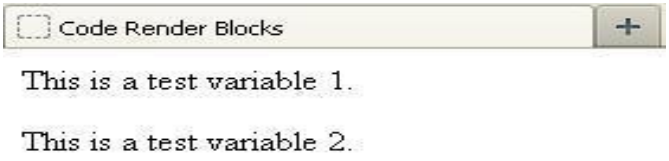
All controls must appear within a <form runat= "server"> tag. Some controls such as and the Label control can appear outside this tag. You can have only one form per page in ASP.NET.

4.Code Render Blocks

If you wish to execute code within HTML, you can include the code within code render blocks.

There are two types of code render blocks:
• Inline Code: It executes a statement or series of statements. It is marked by the characters <% and %>.
• Inline Expressions: They display the value of a variable or method. They can be considered as shorthand notation for the Response.Write method. They are marked by the characters <%= and %>.

Output:



5. Server-Side Comments

You can add comments in server-side code using the characters <%-- and --%>. The main use of these comment blocks is to add documentation to a page.

6.Server-Side Include Directives

You can include a file in an ASP.NET page by using a server-side include directive. It is executed before any of the code in the page. If the file is in the same directory or in a sub-directory of the page including the file, this directive is written as:
<!-- #INCLUDE file="includedfile.aspx" -->

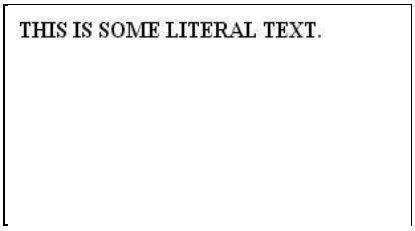
You can also specify the virtual path of the file. To include a file located in the directory MyAspx under the wwwroot directory, you will write the directive as:
<!-- #INCLUDE virtual="/MyAspx/includedfile.aspx" -->

Note: It is recommended that you avoid using server-side include directives. It is better to use user controls.

7.HTML Tags and Literal Text

You can build the static part of an ASP.NET page using HTML tags and literal text. The HTML content of your page is also compiled along with the rest of the contents.
The literal text has been made bold and converted to uppercase before being rendered in the browser.

Output



*Page Directives

Basically, Page Directives are commands. These commands are used by the compiler when the page is compiled.

How to use the directives in an ASP.NET page

It is not difficult to add a directive to an ASP.NET page. It is simple to add directives to an ASP.NET page. You can write directives in the following format:

<% @[Directive] [Attributes] %>

See the directive format, it starts with "<%@" and ends with "%>". The best way is to put the directive at the top of your page. But you can put a directive anywhere in a page. One more thing, you can put more than one attribute in a single directive.

Here is the full list of directives:

1. @Page
2. @Master
3. @Control
4. @Import

5. @Implements
6. @Register
7. @Assembly
8. @MasterType
9. @Output Cache
10. @PreviousPageType
11. @Reference

Let's discuss something about each directive.

1. @Page Directive:

It defines page specific attributes using by .net page parser and Compiler.

Syntax: - <%@ Page attribute=”value” [attribute=”value”……] %>

Example

```
<%@PageLanguage="C#"AutoEventWireup="true"CodeFile="EventTracker.aspx.cs"Inherits="EventTracker"%>
```

Language – Specifies Programming Language used for Code behind Page.

AutoEventWireUp- if value is true Page_Load () event will occur implicitly and if false Page_Load () event should occur explicitly.

CodeFile-Specifies the name for Code behind page file.

Inherits- Specifies name of partial class used on Code behind page file.

2. @Control Directive:

It is Defines control specific attributes by .net page parser and Compiler. It is used only in ascx files.

Syntax: - <%@ Control attribute=”value” [attribute=”value”……] %>

3. @Import Directive:

It is defines that explicitly imports a namespace into .net application. Web page, master page user control page and global.aspx file.

Syntax: - <%@ Import namespace="value" %>

4. @Implements Directive:

It indicates that web page, master page and User control page must be implement .net interface.

Syntax: - <%@ Implements interface="validInterfaceName" %>

5. @Register Directive:

It is creates an association of two tag prefix and custom control in .net web pages.

Syntax: <%@ Register tagprefix ="tagprefix" namespace="namespace" assembly="assembly" %>

6. @Assembly Directive:

Assembly is providing links to .net applications files such as master page, user control and Global.aspx and Web Page. It is also help in creating all interfaces and classes.

Syntax: -

<%@ Assembly Name="assemblyname" %>

<@%Assembly Src="pathname" %>

7. @Master Directive:

It is defines attribute of a master page (.master file).

Syntax: - <%@Master attribute="value" [attribute="value"……] %>

8. @PreviousPageType Directive:

Its provides strong typing against previous page.

Syntax: - <%@ PreviousPageType attribute="value" [attribute="value"……] %>

9. @MasterType Directive:

It provides reference to asp.net master page and accessed master page property.

Syntax: - <%@ MasterType attribute="value" [attribute="value"……] %>

10. @OutputCache Directive:

It is a controls output of caching policies of web page..

Syntax: -

<%@ OutputCache Duration=”#ofseconds” Location=”Any|client|Downstream|server|None|ServerAndClient” Shared=True | False” %>

11. @Reference Directive:

It provides user control and web page file at virtual path.

Syntax: -

<%@ Reference Page =”path to .aspx page” Control=”path to .ascx file” VirtualPath=”path to file” %>

*Self-page and Cross page posting

Self-page Posting

Whenever Page Sends data to itself during postback is known as Self page posting.

For example, you have a data entry page and when you fill this page and click on submit button, then after saving the data, this page posts back to itself.

You will have to know the difference between postback for the first time when the new page is loaded and postback for the second time. So postback is just posting back to the same page. Remember one thing, the postback contains all the information collected on the Initial page for processing.

IsPostBack Property-

The ASP.NET Page class has a property IsPostBack.

We can check that whether the page is postback for the first time or not using IsPostBack Property.

If(Page.IsPostBack==true)

{

//Do whatever you want

}

Cross page posting

Cross page posting means you are posting form data to another page.

This is useful when you want to post data to another page and do not want incur the overhead of reloading the current page.

We can use the property **PostBackUrl** of **Button** Control to redirect to another page. If you want to use the data of one page to another page without using session, object, or anything else, you can just use cross-page in your project.

This is usually required when you are creating a multi-page form to collect information from the user on each page.

When moving from the source to the target page, the values of controls in the source page can be accessed in the target page.

Example-

Here I will explain this concept with simple example for that first create one web application and add two new pages **Default.aspx** and **Default2.aspx**.

Now open **Default.aspx** page and write the following code

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Cross Page Postback Example in asp.net</title>
</head>
<body>
<form id="form1" runat="server">
<div>
```

```
<table>
<tr>
<td><b>Enter UserName:</b></td>
<td><asp:TextBox ID="txtUserName" runat="server"/></td>
</tr>
<tr>
<td><b>Enter Location:</b></td>
<td><asp:TextBox ID="txtLocation" runat="server"/></td>
</tr>
<tr>
<td></td>
<td><asp:Button ID="btnPostback" Text="Postback" runat="server"
PostBackUrl="~/Default2.aspx" /> </td>
</tr>
</table>
</div>
</form>
</body>
</html>
```

If you observe button control code I added property **PostBackUrl="~/Default2.aspx"** by using this property we will submit **Default.aspx** page control values to **Default2.aspx** page.

Now open **Default2.aspx** page and write the following code

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

```
<head runat="server">
<title>Cross Page Postback Example in asp.net</title>
</head>
<body>
<form id="form1" runat="server">
<div>
<b><u>Default2.aspx Page</u></b><br /><br />
<label id="lblName" runat="server" /><br /><br />
<label id="lblLocation" runat="server" />
</div>
</form>
</body>
</html>
```

Now open **Default2.aspx** code behind file and add following namespaces

C# Code

```
using System;
using System.Web.UI.WebControls;
```

Once namespaces added write the following code

```
protected void Page_Load(object sender, EventArgs e)
{
if (PreviousPage != null && PreviousPage.IsCrossPagePostBack)
{
TextBox txtName = (TextBox)PreviousPage.FindControl("txtUserName");
```

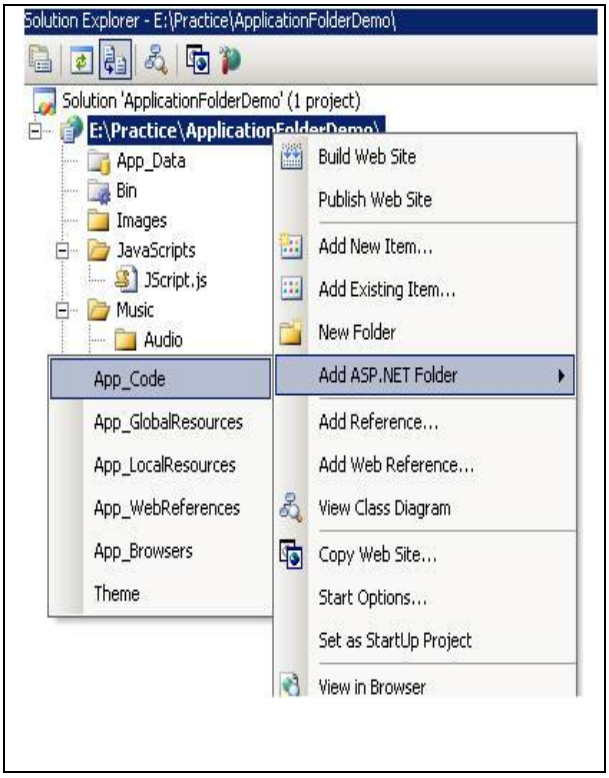
```
TextBox txtLocation = (TextBox)PreviousPage.FindControl("txtLocation");
lblName.InnerText = "Welcome to Default2.aspx page " + txtName.Text;
lblLocation.InnerText = "Your Location: " + txtLocation.Text;
}
else
{
Response.Redirect("Default.aspx");
}
}
```

*Application Folders

The asp.net application folder contains list of specified folder that you can use to store specific type of files or content in an each folder.

The root folder structure is as following

1. **Bin Directory**
2. **App_Code**
3. **App_GlobalResources**
4. **App_LocalResources**
5. **App_WebReferences**
6. **App_Data**
7. **App_Browsers**
8. **App_Themes**



1. **Bin Directory**

It is contains all the precompiled .Net assemblies like DLLs that the purpose of application uses.

2. **App_Code Directory**

It is contains source code files like .cs or .vb that are dynamically compiled for use in your application. These source code files are usually separate components or a data access library

3. **App_GlobalResources Directory**

It is contains to stores global resources that are accessible to every page.

4. **App_LocalResources Directory**

It is serves the same purpose as app_globalresources, exept these resources are accessible for their dedicated page only

5. **App_WebReferences Directory**

It is stores reference to web services that the web application uses.

6. **App_Data Directory**

It is reserved for data storage and also mdf files, xml file and so on.

7. **App_Browsers Directory**

It is contains browser definitions stored in xml files. These xml files define the capabilities of client side browsers for different rendering actions.

8. **App_Themes Directory**

It is contains collection of files like .skin and .css files that used to application look and feel appearance.

Advantages of using Application Folders-

1. We can maintain resources (classes, images, code, databases, themes) in an organized manner, which allows us to develop and maintain sites easily
2. All files and folders are accessible through the application
3. We can add as many files as required
4. Files are compiled dynamically when required