OVERVIEW OF ASP.NET FRAMEWORK:

ASP.NET is a web application framework developed and marketed by Microsoft to allow programmers to build dynamic web sites, web applications and web services. It was first released in January 2002 with version 1.0 of the .NET Framework, and is the successor to Microsoft's Active Server Pages (ASP) technology. ASP.NET is built on the Common Language Runtime (CLR), allowing programmers to write ASP.NET code using any supported .NET language.

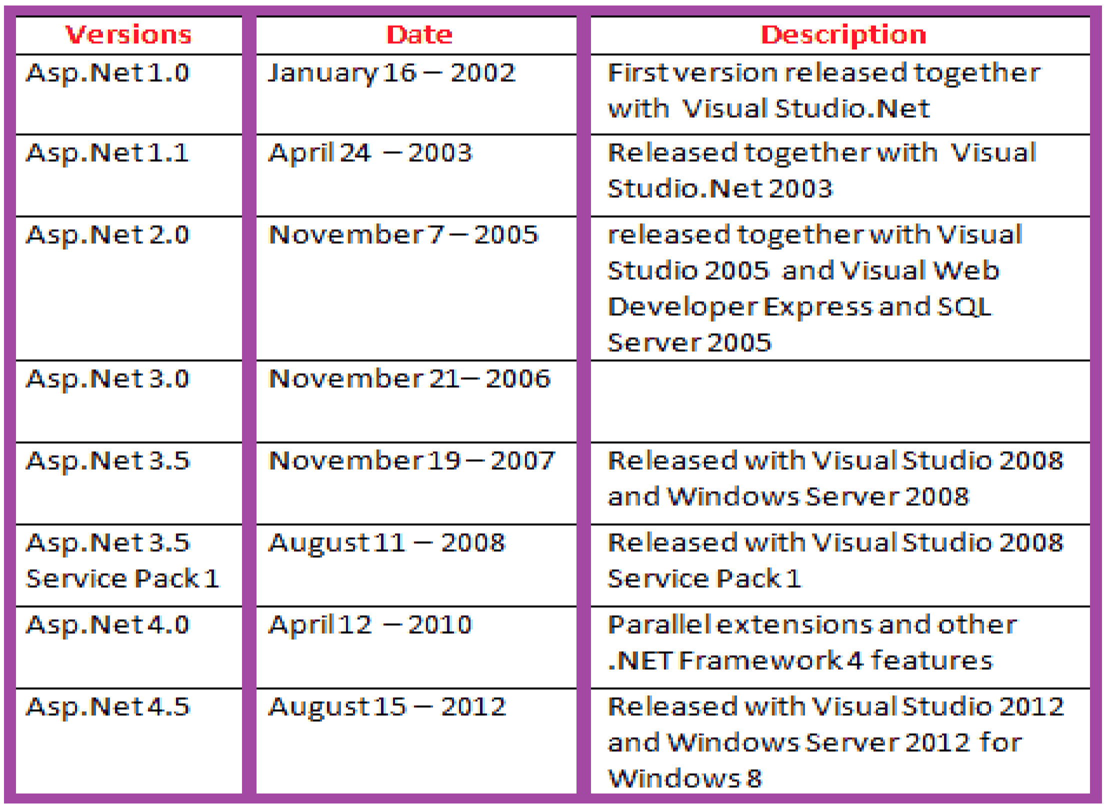
The .NET Framework contains three major parts:

- the Common Language Runtime
- the Framework Class Library
- ASP.NET.

When you develop any C# application, you will initially have to decide on which .net framework are you going to use in your code. First version of .net framework was released in the year 2002. From then on, new features and enhancements were released in the subsequent versions. So far, the following versions of .net framework have been released:

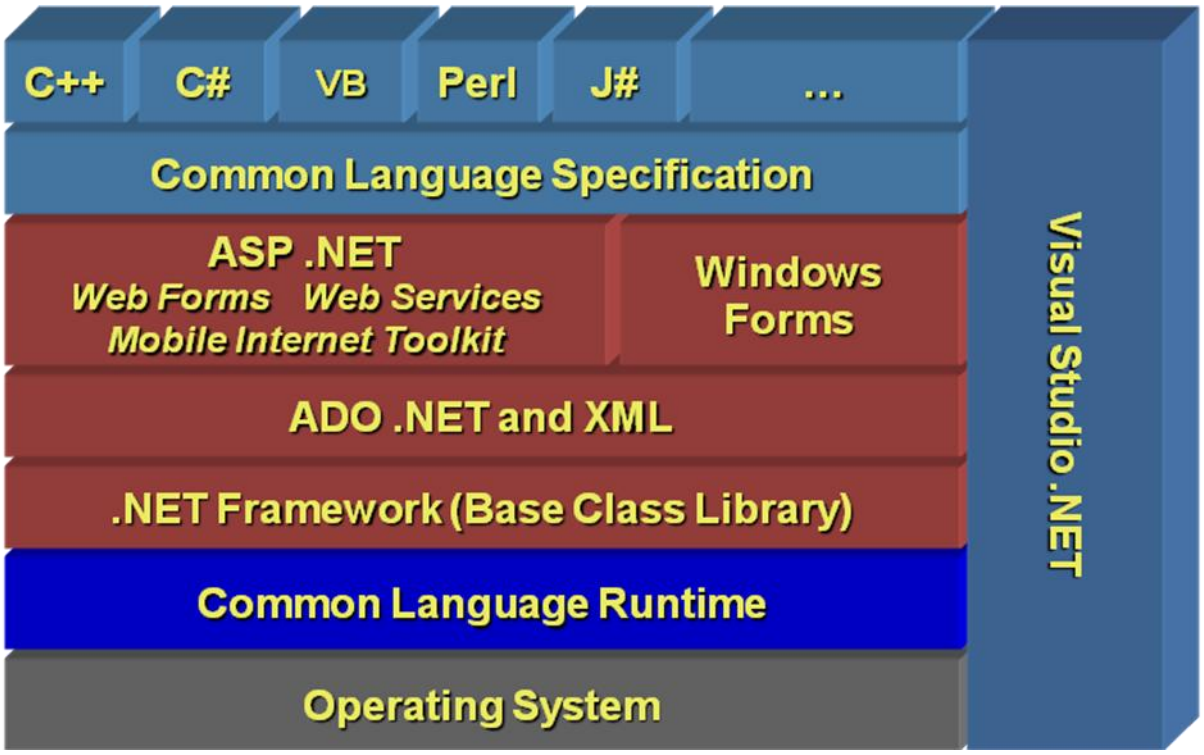
Asp.Net Versions

The following table gives you in details of Asp.Net versions with the .NET Framework releases.



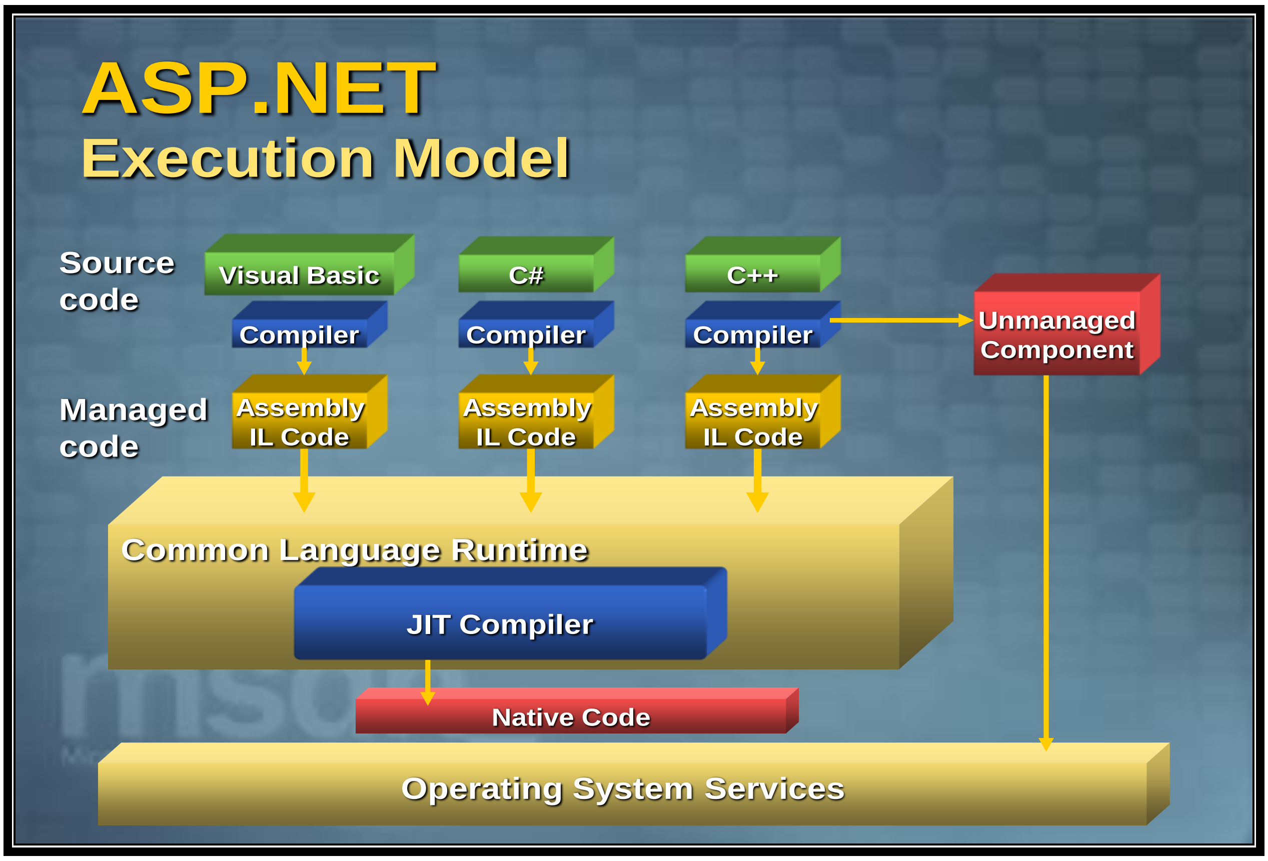
Versions	Date	Description
Asp.Net 1.0	January 16 – 2002	First version released together with Visual Studio.Net
Asp.Net 1.1	April 24 – 2003	Released together with Visual Studio.Net 2003
Asp.Net 2.0	November 7 – 2005	released together with Visual Studio 2005 and Visual Web Developer Express and SQL Server 2005
Asp.Net 3.0	November 21 – 2006	
Asp.Net 3.5	November 19 – 2007	Released with Visual Studio 2008 and Windows Server 2008
Asp.Net 3.5 Service Pack 1	August 11 – 2008	Released with Visual Studio 2008 Service Pack 1
Asp.Net 4.0	April 12 – 2010	Parallel extensions and other .NET Framework 4 features
Asp.Net 4.5	August 15 – 2012	Released with Visual Studio 2012 and Windows Server 2012 for Windows 8

The recent release is version 4.0. But this is a beta version and it is still not rapidly used in the industry. Hottest version of .NET framework acceptable and used widely is the 3.5 Version. Microsoft released ASP.NET 3.5 on November 19, 2007. Along with it, was released Visual Studio 2008.



What is CLR in .NET?

Common Language Runtime - It is the implementation of CLI. The core runtime engine in the Microsoft .NET Framework for executing applications. The common language runtime supplies managed code with services such as cross-language integration, code access security, object lifetime management, resource management, type safety, pre-emptive threading, metadata services (type reflection), and debugging and profiling support. The ASP.NET Framework and Internet Explorer are examples of hosting CLR.



The CLR is a multi-language execution environment. There are currently over 15 compilers being built by Microsoft and other companies that produce code that will execute in the CLR.

The CLR is described as the "execution engine" of .NET. It's this CLR that manages the execution of programs. It provides the environment within which the programs run. The software version of .NET is actually the CLR version.

When the .NET program is compiled, the output of the compiler is not an executable file but a file that contains a special type of code called the Microsoft Intermediate Language (MSIL, now called CIL, Common Intermediate Language). This MSIL defines a set of portable instructions that are independent of any specific CPU. It's the job of the CLR to translate this Intermediate code into a executable code when the program is executed making the program to run in any environment for which the CLR is implemented. And that's how the .NET Framework achieves Portability. This MSIL is turned into executable code using a JIT (Just In Time) complier. The process goes like this, when .NET programs are executed, the CLR activates the JIT complier. The JIT complier converts MSIL into native code on a demand basis as each part of the program is needed. Thus the program executes as a native code even though it is compiled into MSIL making the program to run as fast as it would if it is compiled to native code but achieves the portability benefits of MSIL.

What is Managed Code and Unmanaged Code? :

Microsoft ASP.NET Web applications run under the control of the common language runtime (CLR). The CLR controls how the application's assembly executes, allocates, and recovers memory; therefore, ASP.NET applications are said to use managed code. In contrast, most other Windows executables use unmanaged code because the executable itself determines how memory is used.
Examples of unmanaged code include the Microsoft Win32 API, legacy DLLs and EXEs created for Windows applications prior to the Microsoft .NET Framework and COM objects.

ADVANTAGES USING ASP.NET - FEATURES OF ASP.NET:

1. **Easy Programming Model:**
   In ASP.NET we can easily build real world web applications. Many features likes displaying data validating user inputs, and uploading files are supported in ASP.NET. ASP.NET pages work in all browsers.

2. **Flexible language options:**
   Unlike ASP, which only supports VB scripts and Jscript, ASP.NET now supports more than 25 .NET languages.

3. **Rich Class Framework:**
   External application futures that are hard to implements or which require a 3rd party components, can be easily added in .NET framework now, the .NET framework provides over 4500 classes that have rich functionality like XML, data access. File upload, regular expression, image generation and many more.

4. **Easy Deployment and Dynamic update of running Application:**
   ASP.NET has taken out the extra need for deploying of server application. With ASP.NET we can easily deploy an entire application, by just copy the entire application on to the server. Also ASP.NET lets us update compiled application or pages of application without restarting the web server again and again. We have to just replace the page in which changes are done.

5. **XML Web Services:**
   With XML web services, our web application are able to communicate and share data over internet, regardless of operating system or programming language . Any class can be easily

converted into a XML based web service with few line of code, and can be called by SOAP client.

6. **Mobile Web Device Support:**
ASP.NET mobile controls are developed which targets cell phones, PDAs and over 80 mobile web devices. We can write our application just once, and mobile controls automatically generate its respective mobile files which are required by the devices we are using.

UNDERSTANDING ASP.NET CONTROLS:

APPLICATIONS WEB SERVERS:

To test or run ASP.NET Web applications, you need a Web server. The production Web server for Microsoft operating systems is IIS, which includes a Web server, File Transfer Protocol (FTP) server, Simple Mail Transfer Protocol (SMTP) virtual e-mail server, and other facilities. There are two types of web servers which can work with asp.net:
1) ASP.NET Development server
2) IIS (Internet Information Services/Server)

The ASP.NET Development Server:
If you cannot or do not want to use IIS as your Web server, you can still test your ASP.NET pages by using the ASP.NET Development Server. The ASP.NET Development Server, which is included with Visual Web Developer, is a Web server that runs locally on Windows operating systems, including Windows XP Home Edition. It is specifically built to serve, or run, ASP.NET Web pages under the local host scenario (browsing from the same computer as the Web server). In other words, the ASP.NET Development Server will serve pages to browser requests on the local computer. It will not serve pages to another computer. Additionally, it will not serve files that are outside of the application scope. The ASP.NET Development Server provides an efficient way to test pages locally before you publish the pages to a production server running IIS. The ASP.NET Development Server only accepts authenticated requests on the local computer.
```
http://localhost:31544/ExamplePage.aspx
```
When you close the browser, the ASP.NET Development Server shuts down again.
If you want to run the ASP.NET Development Server on a specific port, you can configure the server to do so. You might do this in these scenarios:

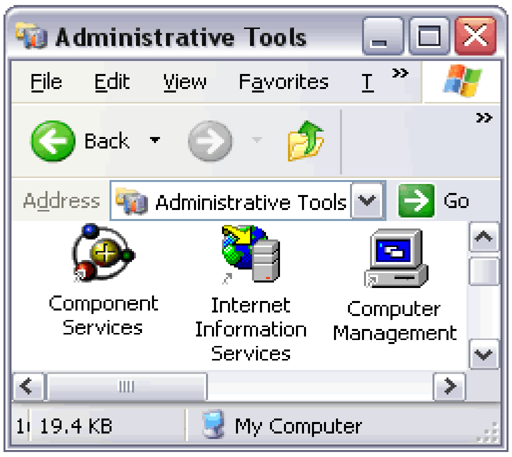
- If code in your application listens to a specific port and you want to be able to test the application using the ASP.NET Development Server.
- If your application includes a reference to a client project or Web service that is bound to a specific port.

INSTALLATION OF IIS:

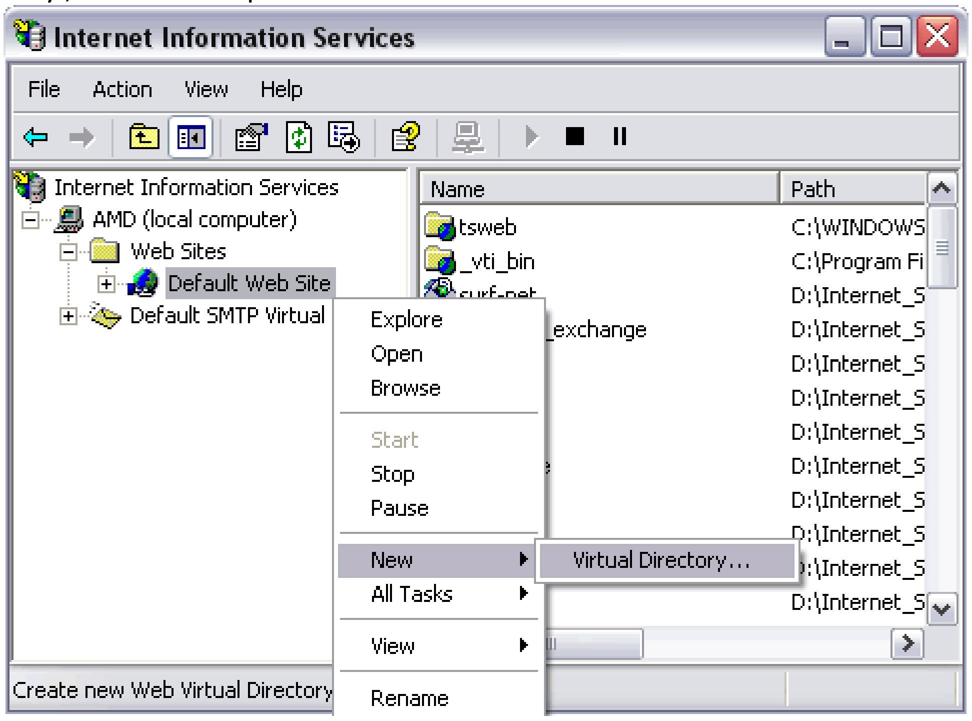
If you are running Windows XP Professional on your computer you can install Microsoft's web server, Internet Information Server 5.1 (IIS) for free from the Windows XP Pro installation CD and configure it to run on your system by following the instructions below:-
1. Place the Windows XP Professional CD-Rom into your CD-Rom Drive.
2. Open 'Add/Remove Windows Components' found in 'Add/Remove Programs' in the 'Control Panel'.
3. Place a tick in the check box for 'Internet Information Services (IIS)' leaving all the default installation settings intact.
4. Once IIS is installed on your machine you can view your home page in a web browser by typing 'http://localhost' (you can substitute 'localhost' for the name of your computer) into the address bar of your web browser. If you have not placed your web site into the default directory you should now be looking at the IIS documentation.

5. If you are not sure of the name of your computer right-click on the 'My Computer' icon on your desktop, select 'Properties' from the shortcut menu, and click on the 'Computer Name' tab.
6. Your default web directory to place your web site in is 'C:\Inetpub\wwwroot', but if you don't want to over write the IIS documentation found in this directory you can set up your own virtual directory through the 'Internet Information Services' console.
7. The 'Internet Information Services' console can be found in the 'Administration Tools' in the 'Control Panel' under 'Performance and Maintenance', if you do not have the control panel in Classic View.

8. Double-click on the 'Internet Information Services' icon.



9. Once the 'Internet Information Services' console is open you will see any IIS web services you have running on your machine including the SMTP server and FTP server, if you chose to install them with IIS.
10. To add a new virtual directory right click on 'Default Web Site' and select 'New', followed by 'Virtual Directory', from the drop down list.



11. Next you will see the 'Virtual Directory Creation Wizard' from the first screen click the 'next' button.
12. You will then be asked to type in an 'Alias' by which you will access the virtual directory from

your web browser (this is the name you will type into your web browser after 'localhost' to view any web pages you place in the directory).
10. Next you will see a 'Browse...' button, click on this to select the directory your web site pages are in on your computer, after which click on the 'next' button to continue.
14. On the final part of the wizard you will see a series of boxes, if you are not worried about security then select them all, if you are and want to run ASP scripts then check the first two, followed by the 'next' button.
15. Once the virtual directory is created you can view the web pages in the folder by typing 'http://localhost/aliasName' (where 'aliasName' is, place the alias you called the virtual directory) into the address bar of your web browser (you can substitute 'localhost' for the name of your computer if you wish).

FILES TYPES USED IN ASP.NET

ASP.NET have many types of files. They are

(1).aspx(Web Form):

The .NET pages, known also as as web forms, are the main building block for application development. The Web forms are contained in files with an .aspx extension. These files typically contain static (X)HTML markup, as well as markup defining server-side Web Controls and User Controls where the developers place all the required static and dynamic content for the web page.
The **ASPX file extension** is used for files that are of the Active Server Page Extended format. These are internet webpages which are made using ActiveX scripting, and which are based on the Microsoft ASP.NET programming framework. These webpages are generated by Internet servers. The ASP code is created by the Internet server, and then is interpreted as HTML by the web browser.
The native application that used ASPX script files is ASP.NET server (mainly Microsoft IIS - Internet Information Server). ASPX scripts files can be written in Microsoft Visual Studio and WWW pages, that are written in ASP.NET and used ASP.NET, can be viewed in Internet browser such as Internet Explorer, Firefox, Opera, and Chrome.

(2).ascx:

It is the file format for the User controls in ASP.NET. The ascx file will be representing any ASP.NET user control in the project.A user control is a kind of composite control that works much like an ASP.NET Web page—you can add existing Web server contols and markup to a user control, and define properties and methods for the control. You can then emed them in ASP.NET Web pages, where they act as a unit.

(3).asmx:

There are ASP.NET web services. Web services are components on a Web server that a client application can call by making HTTP requests across the Web. ASP.NET enables you to create custom Web services or to use built-in application services, and to call these services from any client application.

(4).ashx:

A handler file that is invoked in response to a Web request in order to generate dynamic content.
An ASP.NET HTTP handler is the process (frequently referred to as the "endpoint") that runs in response to a request made to an ASP.NET Web application. The most common handler is an ASP.NET page handler that processes .aspx files. When users request an .aspx file, the request is processed by the page through the page handler. You can create your own HTTP handlers that render custom output to the browser.
An HTTP module is an assembly that is called on every request that is made to your application. HTTP modules are called as part of the ASP.NET request pipeline and have access to life-cycle events throughout the request. HTTP modules let you examine incoming and outgoing requests and take action based on the request.

(5)Web.Config:
Web.config file, as it sounds like is a configuration file for the Asp .net web application. An Asp .net application has one web.config file which keeps the configurations required for the corresponding application. Web.config file is written in XML with specific tags having specific meanings.
(6).global.asax:
The Global.asax file, also known as the ASP.NET application file, is an optional file that contains code for responding to application-level events raised by ASP.NET or by HttpModules. The Global.asax file resides in the root directory of an ASP.NET-based application. At run time, Global.asax is parsed and compiled into a dynamically generated .NET Framework class derived from the HttpApplication base class. The Global.asax file itself is configured so that any direct URL request for it is automatically rejected; external users cannot download or view the code written within it.
The ASP.NET Global.asax file can coexist with the ASP Global.asax file. You can create a Global.asax file either in a WYSIWYG designer, in Notepad, or as a compiled class that you deploy in your application's \Bin directory as an assembly. However, in the latter case, you still need a Global.asax file that refers to the assembly.
The Global.asax file is optional. If you do not define the file, the ASP.NET page framework assumes that you have not defined any application or session event handlers.
(7).cs/ .vb/ .js(Code Behind Files): These are code –behind files that contain c# code, VB code or J# code. They allow you to separate the application from the user interface of web page.

WEB FORMS:
Web pages in ASP.Net are called ASP.NET Web Forms which have certain server controls such as text, dropdown list, checkboxes, and buttons. An ASP.NET Web Form looks similar to the web forms in HTML. The only difference is that in ASP.NET, the Web Forms runs at the server side and in HTML the web forms runs at the client side. Apart from this difference an ASP.NET Web Form has more features than an ordinary HTML web form such as:
• The code blocks are processed on the server.
• The entire page in ASP.NET is compiled when it is requested for the first time. When you make subsequent requests, the page is not compiled but shown directly in your browser.
• ASP.NET Web Forms can contain page directives. Page directives allow you to set the default language and user controls tags for the entire page. You can also turn off session state and ViewState management using page directives.
• An ASP.NET Web Form can contain both user controls and Server side Includes (SSIs).
• An ASP.NET Web Form though run on the server can contain client side script such as JavaScript or Jscript.
An ASP.NET Web Form first gets compiled into Intermediate Language (IL) and later to the machine language. Not just the server side code gets compiled but every control and element in the page gets compiled. This enables the ASP.NET Web Form to be in any language that is compatible with Common Language Runtime (CLR) engine. Another advantage with the code getting translated to IL is that the user can use the application in any browser. In addition, you can reduce the processors and server's load when you compile the ASP.NET Web Forms in ASP.NET, thereby, increasing the user's productivity time. Thus ASP.NET allows you to create Web Forms that are platform independent, language independent, and browser independent.
WEB FORM CONTROLS:
SERVER CONTROLS:
Controls are small building blocks of the graphical user interface, which includes text boxes, buttons, check boxes, list boxes, labels and numerous other tools, using which 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, 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 again could be grouped into the following categories:

- Validation controls - these are used to validate user input and 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 display
- Personalization controls - these are used for personalization of a page according to the user's preference, based on user information
- Login and security controls - these controls provide user authentication
- Master pages - these provides consistent layout and interface throughout the application
- Navigation controls - these helps in navigation, for example, the menus, tree view etc.
- Rich controls - these implements special features, for example, AdRotator control, FileUpload control, Calendar control etc.

CLIENT CONTROLS:

Client controls are control which don't execute on server side. Client controls don't expose the functionality of event handling because event handling is generally done on the control which execute on the server side. HTML controls are generally called as client control but they can be converted to server control after adding ID and runat attribute to the code. ASP.NET Framework (version 3.5) caontains over 70 controls. These controls can be divided into eight groups:

Standard Controls:

The standard controls enable you to render standard form elements such as buttons, labels and input fields. Standard controls duplicate the functionalities of the basic HTML tags but are more consistent in terms of properties and methods. All standard ASP.NET controls have two important attributes with them i.e. "ID" and "runat" attribute.

Validation Controls:

An important aspect of creating ASP.NET Web pages for user input is to be able to check that the information users enter is valid. ASP.NET provides a set of validation controls that provide an easy-to-use but powerful way to check for errors and, if necessary, display messages to the user. For example RequiedFieldValidator control to check whether a user entered a value for a required input field.

Rich Controls:

In addition to the preceding controls, the ASP.NET page framework provides a few, task-specific controls called rich controls. Rich controls are built with multiple HTML elements and contain rich functionality. Examples of rich controls are the Calendar control and the AdRotator control.

Data Controls:

The data controls enable you to work with data such as database data. For example, you can use these controls to submit new records to a database table or display a list of database records. Navigation controls enable you to display standard navigation elements such as menus and tree views.

Login Controls:

The login controls enable you to display login, change password and registration forms. Login controls support form authentication.

Web Part Controls:

The web part controls enable you to build personalize portal applications.

HTML Controls:

The HTML controls enables you to convert any HTML tag into a server –side control. If you an attribute called “runat=’server’”, HTML controls works as standard controls and are sent to server while processing page.

ASP.NET Mobile Controls:

These are controls which are somewhat similar to web controls but these are customized to support mobile clients. Mobile clients can be PDAs, Smart Phones, etc.

An Internet Port:

Definition: In computer networking, the term port can refer to either physical or virtual connection points. Physical network ports allow connecting cables to computers, routers, modems and other peripheral devices. Several different types of physical ports available on computer network hardware include:

- Ethernet ports
- USB ports
- serial ports

Virtual ports are part of TCP/IP networking. These ports allow software applications to share hardware resources without interfering with each other. Computers and routers automatically manage network traffic traveling via their virtual ports. Network firewalls additionally provide some control over the flow of traffic on each virtual port for security purposes.

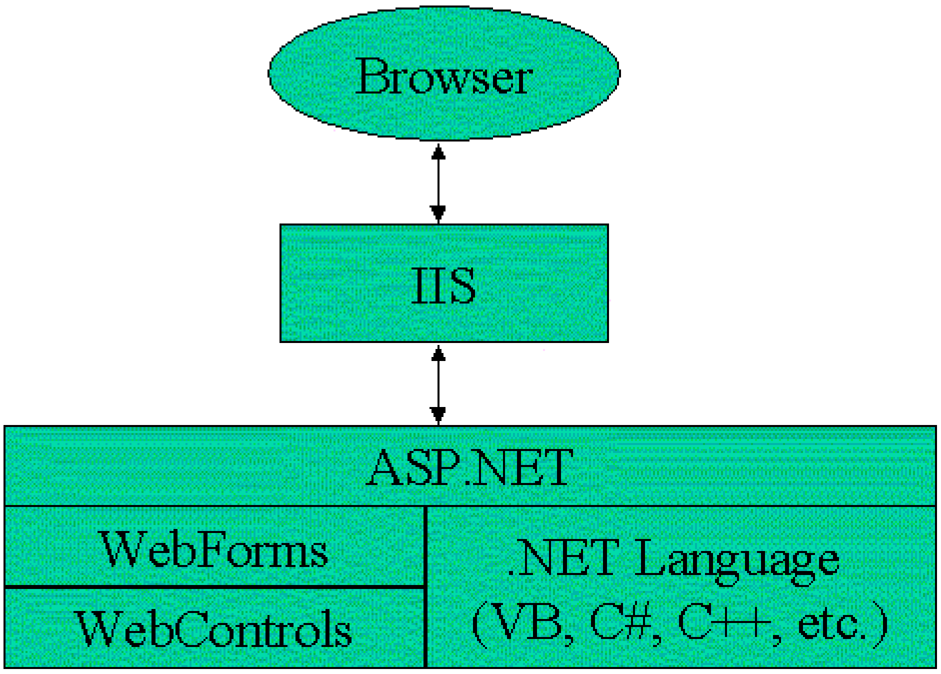
In both TCP and UDP, port numbers start at 0 and go up to 65535. Numbers in the lower ranges are dedicated to common Internet protocols (like 21 for FTP, 80 for HTTP, etc.). Sometimes, a Web site URL will require a specific TCP port number be included. For example, http://localhost:8080/ uses TCP port 8080. Again, this is more usually seen in software development environments than on the Internet.

WEB FORMS AND HTML:

Microsoft® ASP.NET is the next generation technology for Web application development. It takes the best from Active Server Pages (ASP) as well as the rich services and features provided by the Common Language Runtime (CLR) and add many new features. The result is a robust, scalable, and fast Web development experience that will give you great flexibility with little coding.

Web Forms are the heart and soul of ASP.NET. Web Forms are the User Interface (UI) elements that give your Web applications their look and feel. Web Forms are similar to Windows Forms in that they provide properties, methods, and events for the controls that are placed onto them. However, these UI elements render themselves in the appropriate markup language required by the request, e.g. HTML. If you use Microsoft Visual Studio® .NET, you will also get the familiar drag-and-drop interface used to create your UI for your Web application.

Web Forms are made up of two components: the visual portion (the ASPX file), and the code behind the form, which resides in a separate class file.



Web Forms are a part of ASP.NET

The Purpose of Web Forms:

Web Forms and ASP.NET were created to overcome some of the limitations of ASP. These new strengths include:

- Separation of HTML interface from application logic
- A rich set of server-side controls that can detect the browser and send out appropriate markup language such as HTML
- Less code to write due to the data binding capabilities of the new server-side .NET controls
- Event-based programming model that is familiar to Microsoft Visual Basic® programmers
- Compiled code and support for multiple languages, as opposed to ASP which was interpreted as Microsoft Visual Basic Scripting (VBScript) or Microsoft Jscript®
- Allows third parties to create controls that provide additional functionality

On the surface, Web Forms seem just like a workspace where you draw controls. In reality, they can do a whole lot more. But normally you will just place any of the various controls onto the Web Form to create your UI. The controls you use determine which properties, events, and methods you will get for each control. There are two types of controls that you can use to create your user interface: HTML controls and Web Form controls.

HTML Controls :

HTML controls mimic the actual HTML elements that you would use if you were using Front Page or any other HTML editor to draw your UI. You can use standard HTML elements in Web Forms, too. For example, if you wanted to create a text box, you would write:

```
<input type="text" id=txtFirstName size=25>
```

If you are using Visual Studio .NET, you choose a TextField control from the HTML Toolbox tab and draw the control where you want it on the HTML page.

Any HTML element can be marked to also run as an HTML control when the Web Form is processed on the server by adding "runat=server" to the tag:

```
<input type="text" id=txtFirstName size=25 runat=server>
```

HTML controls allow you to handle server events associated with the tag (a button click, for example), and manipulate the HTML tag programmatically in the Web Form code. When the control is rendered to the browser, the tag is rendered just as it is saved in the Web Form, minus the "runat=server". This gives you precise control over the HTML that is sent to the browser.

ADDING CONTROLS TO A WEB FORM:

All Web Form controls inherit from a common base class, namely the **System.Web.UI.WebControls** class. All web controls begin by inheriting from the web control base class. This class defines the essential functionality for tasks such as data binding and includes some basic properties that you can use with any control. In fact .NET has a class called"Control" class. "Web control" is a class which derived from "Control" class. Webcontrol is further derived into the classes like Textbox, Label, Calendar, AdRotator, etc class which are used as different types of controls.

BUTTON:

Button control is generally used to post the form or fire an event either client side or server side. When it is rendered on the page, it is generally implemented through <input type=submit> HTML tag. However, if UserSubmitBehavior property is set to false then control will render out as <input type=button>.

Its properties like BackColor, ForeColor, BorderColor, BorderStyle, BorderWidth, Height etc. are implemented through style properites of <input> tag. You can set its Text property either by setting Text property in the .aspx page or from server side page. (other properties can also be set from both pages) .

A button can be set as default button (Default button is a button that is Clicked when the user presses Enter key while filling a form.) by specifying DefaultButton property of the <form> tag or to the asp:Panel tag in which it is placed.

TEXT BOX:

TextBox control is used to enter data into the form that can be sent to the webserver by posting the form. The form can be posted by clicking a button or an Image. When it is rendered on the page, it is implemented through <input> or <textarea> HTML tag. When TextMode property is Singleline then this control is rendered as <input> HTML tag with Type property as Text, if Multiline then <textarea>, and if Password then <input> HTML tag with Type property as Password. If you will not set TextMode property, it will be set to Singleline by default. Its properties like BackColor, ForeColor, BorderColor, BorderStyle, BorderWidth, Height etc. are implemented through style property of respective tags. You can set its Text property either by setting Text properties in the .aspx page or from server side page. (other properties can also be set from both pages) When AutoPostBack property of the TextBox is set to true, then Form is posted back to the server when cursor leaves the box

LABEL:

Ideally Label control is used to place a static, non clickable (can't fire onclick event) piece of text on the page. When it is rendered on the page, it is implemented through <span></span> HTML tag. Its properties like BackColor, ForeColor, BorderColor, BorderStyle, BorderWidth, Height etc. are implemented through style properites of <span>. You can set its Text property either by setting Text properties in the .aspx page or from server side page. (other properties can also be set from both pages)

CHECKBOX:

Ideally CheckBox control is used to give option to the user. When it is rendered on the page, it is implemented through <input type=checkbox></input> HTML tag. Its properties like BackColor, ForeColor, BorderColor, BorderStyle, BorderWidth, Height etc. are implemented through style properites of <input>. You can set its Text property either by setting Text properties in the .aspx page or from server side page. (Other properties can also be set from both pages)

Following are some important properties that are very useful.

AutoPostBack	Form is automatically posted back when CheckBox is checked or Unchecked.
CausesValidation	true/false. If true, Form is validated if Validation control has been used in the

	form.
Checked	true/false. If true, Check box is checked by default.
OnCheckedChanged	Fires when CheckBox is checked or Unchecked. This works only if AutoPostBack property is set to true.
ValidationGroup	Used to put a checkbox under a particular validation group. It is used when you have many set of form controls and by clicking a paricular button you want to validate a particular set of controls only.

RADIO BUTTONS:

RadioButton control is used to give single select option to the user from multiple items. When it is rendered on the page, it is implemented through <input type=radio></input> HTML tag. Its properties like BackColor, ForeColor, BorderColor, BorderStyle, BorderWidth, Height etc. are implemented through style properites of <input>.
Following are some important properties that are very useful.

AutoPostBack	Form is automatically posted back when Radio button selection is changed.
CausesValidation	true/false. If true, Form is validated if Validation control has been used in the form.
Checked	true/false. If true, Radio button is selected by default.
OnCheckedChanged	Fires when Radio button selection changes. This works only if AutoPostBack property is set to true.
ValidationGroup	Used to put a radio button under a particular validation group. It is used when you have many set of form controls and by clicking a paricular button you want to validate a particular set of controls only.
GroupName	It is used a group a set of radion buttons so only one of them can be selected at a time.

LIST BOX:

ListBox control is used to give a single or multiple select options to the user from multiple listed items. All properties and its working resembles DropDownList box. However, ListBox has two extra properties called Rows and SelectionMode. ListBox control is used to give a single or multiple select option to the user (based on the property set) from multiple listed items. You can specify its height and width in pixel by setting its height and width but you will not be able give mutliple select option to the user. When it is rendered on the page, it is implemented through <select/> HTML tag. It is also called as Combo box. Its properties like BackColor, ForeColor etc. are implemented through style properites of <span>. It has less property to decorate in comparison with other controls. There is no property like BorderStyle, BorderWidth. in DropDownList control. You can add its option items by directly writing into .aspx page directly or dynamically add at run time or bind through database.
Following are some important properties that are very useful.

Rows	No. of rows (items) can be set to display in the List.
SelectionMode	Single or Multiple. If multiple, it allows user to select multiple items from the list by holding Ctrl or Shift key.
SelectedValue	Get the value of the Selected item from the dropdown box.
SelectedIndex	Gets or Sets the index of the selected item in the dropdown box.
SelectedItem	Gets the selected item from the list.
Items	Gets the collection of items from the dropdown box.

DataTextField	Name of the data source field to supply the text of the items. (No need to set when you are adding items directly into .aspx page.)
DataValueField	Name of the data source field to supply the value of the items. (No need to set when you are adding items directly into .aspx page.)
DataSourceID	ID of the datasource component to provide data. (Only used when you have any DataSource component on the page, like SqlDataSource, AccessDataSource etc.)
DataSource	The datasource that populates the items in the listbox box. (Generally used when you are dynamically generating the items from Database.)
AutoPostBack	true or false. If true, the form is automatically posted back to the server when user changes the dropdown list selection. It will also fire OnSelectedIndexChanged method.
AppendDataBoundItems	true or false. If true, the statically added item (added from .aspx page) is maintained when adding items dynamically (from code behind file) or items are cleared.
OnSelectedIndexChanged	Method name that fires when user changes the selection of the dropdown box. (Fires only when AutoPostBack=true.)

DropDownList Control:

It is used to represents a control that allows the user to select a single item from a drop-down list. The basic syntax can be given as:
<asp:DropDownList Id="DropDownList1" runat="server"></asp:DropDownList>
Here ID is unique identifier of control and runat specifies that control is made to run on the server.
Some of the properties of DropDownList control can be given as :

Property	Meaning
Enabled	Enables to make a button enabled or disable control.
TabIndex	Enables to get or set tab index for control.
Items	Enables to get the collection of items in the List Control.
SelectedIndex	Enables to get or set the index of the selected item in the DropDownlist control.
SelectedItem	Enables to get the selected item with the lowest index in the list.
AutoPostBack	Enabless to get post form containg the DropDownList back to the server automatically when the dropdwonlist item is selected.
SelectedValue	Enables us to get the value of selected item in the list control, or selects the item that contains specified values.

Page Class:

All the web forms in ASP.NET are actually instances of the page class, which is defined in System.Web.UI namespace. The page class inherits from the TemplateControl Claass, which in turn inherits from the Control class. As a result, the page class provides useful properties and methods that to be used our code.

Object Name	Function
Session	The session object is an instance of the System. Web.SessionState.HttpSessionState class. It acts as a global repository to store user specific data that needs to persist between web-page requests. The session object stores data in the form of name/ value pairs which is used to maintain data like user's name ID and other elopements which are discarded when a user logs out or is not accessing any page of the web site. Session state can also be used for both in process and out process.
Application	The application object is an instance of the System.HttpApplicationState

	class. Like session object, it also stores data in the form of name/value pairs, but the difference is the scope of his stored is global to the entire application.
Cache	Cache object is an instance of the System.Web.Caching.Cache class. It is also used to store global information. It is also a name/ value collection of objects, but we can define the expiration policies and dependencies for each item so that items are automatically removed when other resources, such as files or database tables, are modified.
Request	The request object is an instance of the Syste.Web>HttpRequest class. It represents the values and properties of the HTTP request that caused the page to be loaded. It contains all the URL parameters and all other information sent by the client.
Response	The Response object is an instance of the Syste.Web.HttpResponse class, and it represents the web server's response to a client's request.

Page Directive:

Page directive is used to specify the default programming language for a page. Page directives can be used for a tracing page and import directives. The basic syntax of page Directive is:

<%a Page language="C#" autoEventWireup="true" CodeFile="Default.aspx.cs"Inherits="_default"%>

These are default arguments available as we create page. The above properties indicate:

1. Language indicates default programming language of the page.
2. AutoEventWireup indicates that the events are enabled on this page.
3. CodeFile indicates the code file associated with this page.
4. Inherits indicates the class file inherited by this page.

Ii also has other properties such as Trace, Debug, and many others. Some of the them will be discussed the class file inherited by this page.

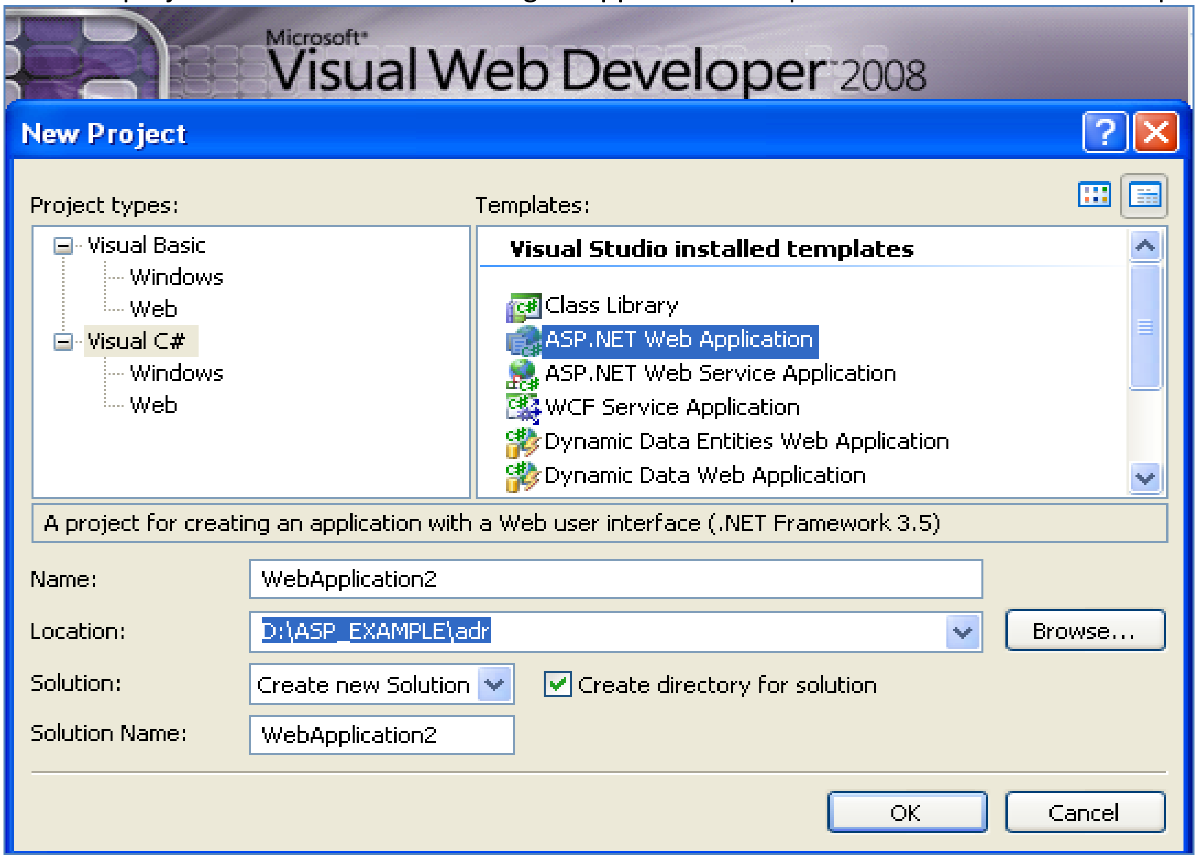
It also has other properties such as Trace, Debug and many others.

RUNNING A WEB APPLICATION:

ASP.Net provides an abstraction layer on top of HTTP on which the web applications are built. It provides high-level entities like classes and components within an object-oriented paradigm. The key development tool for building ASP.Net applications and front ends is Visual Studio. In these tutorials, we will work on 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.


Microsoft Visual Web Developer 2008
New Project
Project types:
Visual Basic
Windows
Web
Visual C#
Windows
Web
Templates:
Visual Studio installed templates
Class Library
ASP.NET Web Application
ASP.NET Web Service Application
WCF Service Application
Dynamic Data Entities Web Application
Dynamic Data Web Application
Browse...
OK
Cancel

A project for creating an application with a Web user interface (.NET Framework 3.5)

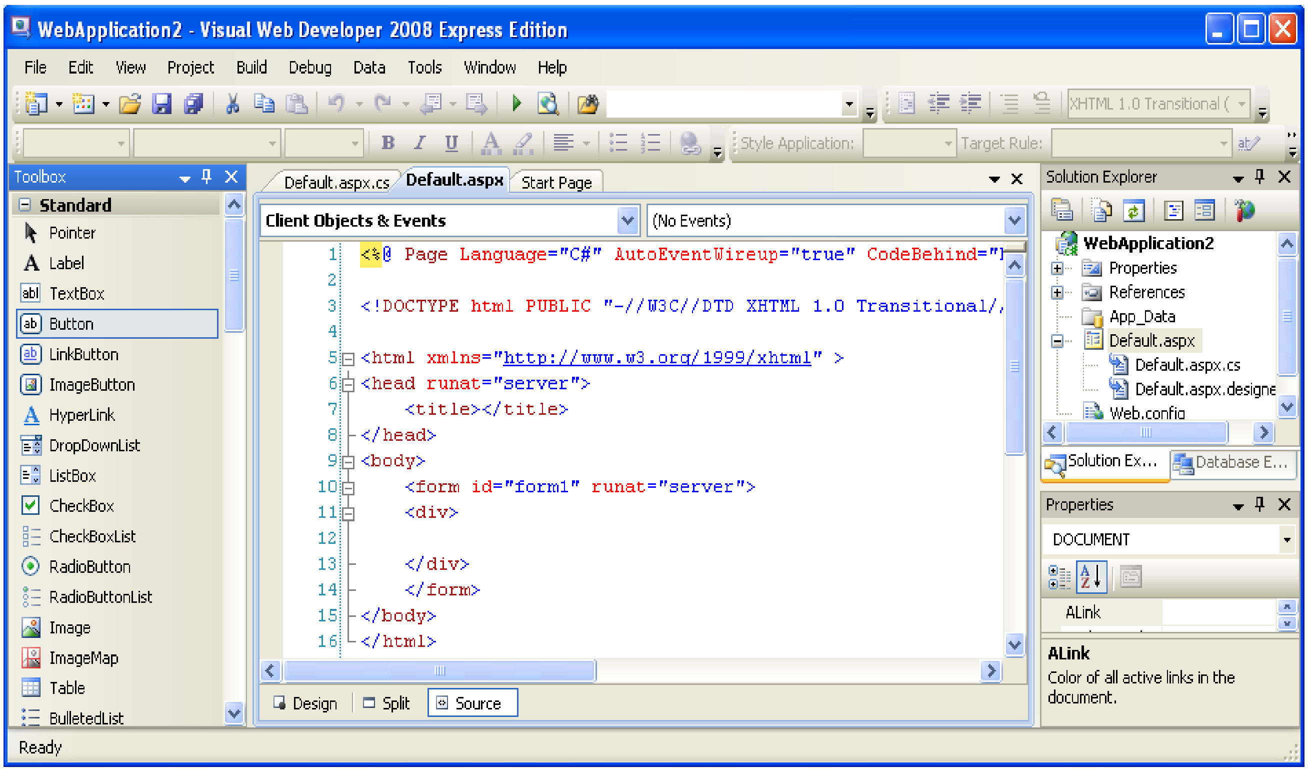
Name: WebApplication2
Location: D:\ASP_EXAMPLE\adr
Solution: Create new Solution
Create directory for solution ✓
Solution Name: WebApplication2

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 form's works. 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.

Ways to work with views and windows:

The following are the ways to work with different windows:

- 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 changes into a tab, to redisplay again click on the Auto Hide button again.
- To size a wind just drag it.



```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="]

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//

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

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

ALink
Color of all active links in the document.

Adding folders and files to your web site:

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 (e.g., the .cs files), assemblies (e.g., the .dll files and .exe files), data source files (e.g., .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:

The application is run by selecting either Start or Start without Debugging from the Debug menu, or by pressing F5 or Ctrl-F5. The program is built i.e. the .exe or the .dll files are generated by selecting a command from the Build menu.

CALENDAR CONTROL:

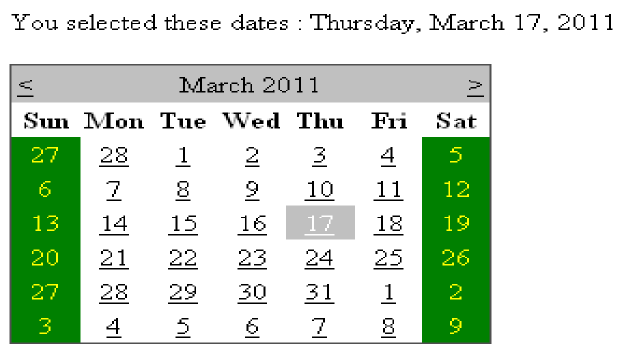
ASP.NET 3.5 Calendar control creates a functionally rich and good looking calendar box that shows one month at a time. The user can move from month to month, select a date, and even select a range of days. The calendar control has many properties like you can change foreground color, background color, font, title, format of date, currently selected date and so on. Use the Calendar control to display a single month of a calendar on a Web page. The control allows you to select dates and move to the next or previous month.
The important Calendar event is SelectionChanged which fires every time a user clicks a date. Here is a basic event handler that responds to the SelectionChanged event and displayes the selected date.
There are nine events in calendar control.

1. DayRender - Occurs when each day is created in the control hierarchy for the Calendar control.
2. SelectionChanged - Occurs when the user selects a day, a week, or an entire month by clicking the date selector controls.
3. VisibleMonthChanged - Occurs when the user clicks on the next or previous month navigation controls on the title heading.
4. DataBinding - Occurs when the server control binds to a data source.
5. Disposed - Occurs when a server control is released from memory, which is the last stage of the server control lifecycle when an ASP.NET page is requested.
6. Init - Occurs when the server control is initialized, which is the first step in its lifecycle.
7. Load - Occurs when the server control is loaded into the Page object.
8. PreRender - Occurs when the server control is about to render to its containing Page object.
9. Unload - Occurs when the server control is unloaded from memory.

```
protected void Calendar1_SelectionChanged(object sender, EventArgs e)
     {
          DatesLabel.Text = "You have selected these dates : <br />";
          foreach (DateTime dt in Calendar1.SelectedDates)
          {
               DatesLabel.Text += dt.ToLongDateString() + "<br />";
          }

    }
protected void Calendar1_DayRender(object sender, DayRenderEventArgs e)
    {
        if (e.Day.IsWeekend)
        {
            e.Cell.BackColor = System.Drawing.Color.Green;
            e.Cell.ForeColor = System.Drawing.Color.Yellow;
            e.Day.IsSelectable = false;

        }
}
```



You selected these dates : Thursday, March 17, 2011

<			March 2011			>
Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

AD ROTATOR CONTROL:

The AdRotator is one of the rich web server control of asp.net. AdRotator control is used to display a sequence of advertisement images as per given priority of image.

Adrotator control display the sequence of images, which is specified in the **external XML file**. In a xml file we indicate the images to display with some other attributes, like image impressions, NavigateUrl, ImageUrl, AlternateText.
In a Adrotator control **images will be changed** each time **while refreshing** the web page.
The basic syntax of adding an AdRotator is as follows:

```
<asp:AdRotator  runat = "server" AdvertisementFile = "adfile.xml"  Target =
"_blank" />
```

We need to add in the advertisements to our XML file. The advertisements can contain any of the following properties:

Element	Description
<ImageUrl>	The path to the image file
<NavigateUrl>	The URL to link to if the user clicks the ad
<AlternateText>	An alternate text for the image
<Keyword>	A category for the ad
<Impressions>	Describe the display rate of images or priority of image.

The XML file that is used for advertisements would have the root element as the <Advertisements> element. This may have many <Ad> child elements. The elements <ImageUrl>, <NavigateUrl>, <AlternateText>, <Impressions> and <Keyword> are found in the <Ad> element. These elements identify the properties of the image that is displayed using the adrotator control.
The <ImageUrl> element specifies the path to the image that is displayed using the AdRotator control. Once a user clicks the image that is displayed the user is taken to the URL specified in the <NavigateUrl> element. The <AlternateText> element holds the text that is displayed when the image is not loaded. The text in this element is also used as a ToolTip in some of the browsers. The element <Impressions> has a number as a value. This number indicates how often the image is displayed with respect to other images. The more the value of the number in this element, the more is the frequency of the image that is displayed. The sum of all the impressions in the advertisement file should not exceed 2,047,999,999. Otherwise the AdRotator will throw an runtime exception. The element <Keyword> is used to give the category of the advertisement.

Way to Storing Advertisements in XML file:

The general structure of advertisement file in XML can be given as follows:

```
<Advertisements>
           <Ad>
                      <ImageUrl>           </ImageUrl>
                      <Width>              </Width>
                      <Height>             </Height>
                      <AlternateText>      </AlternateText>
                      <NavigateUrl>        </NavigateUrl>
                      <Keyword>            </Keyword>
                      <Impressions>        </Impressions>
          </Ad>
</Advertisements>
```

FILE UPLOAD CONTROL:

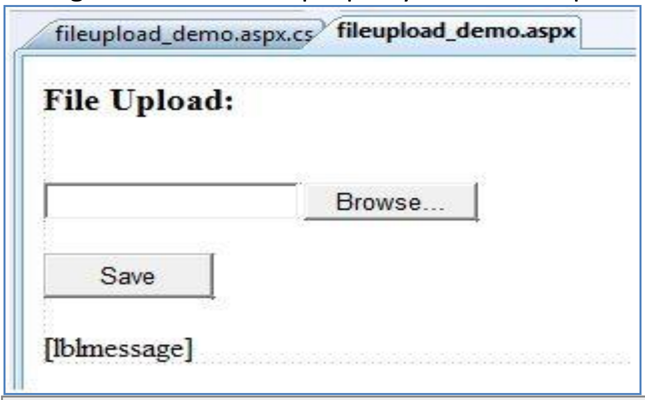
The FileUpload control enables us to upload any files to a given location in our web application. After the file is uploaded, you can store the file anywhere you want. You can either store file either on file system or in the database as per your requirement.

```
<asp:FileUpLoad id="FileUpLoad1" runat="server" />
```

The FileUpload class is derived from the WebControl class, and inherits all its members. Apart from those, the FileUpload class has the following read-only properties:

Properties	Description
FileBytes	Returns an array of the bytes in a file to be uploaded.
FileContent	Returns the stream object pointing to the file to be uploaded.
FileName	Returns the name of the file to be uploaded.
HasFile	Specifies whether the control has a file to upload.
PostedFile	Returns a reference to the uploaded file.

The posted file is encapsulated in an object of type HttpPostedFile, which could be accessed through the PostedFile property of the FileUpload class.



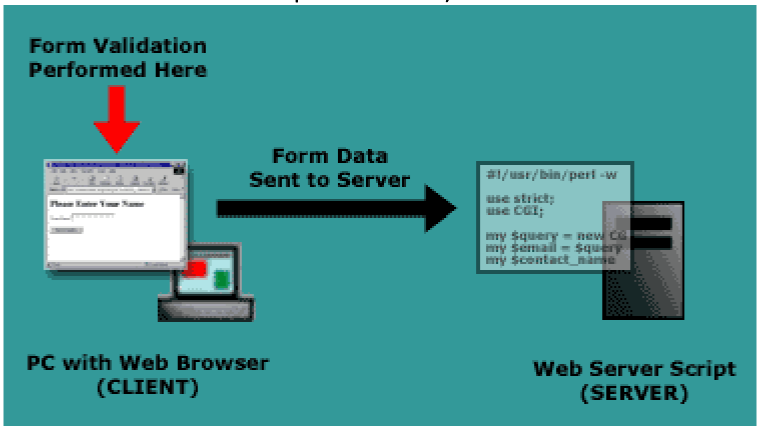
```
protected void UploadButton_Click(object sender, EventArgs e)
{
    if(FileUploadControl.HasFile)
    {
        try
        {
            string filename = Path.GetFileName(FileUploadControl.FileName);
            FileUploadControl.SaveAs(Server.MapPath("~/") + filename);
            StatusLabel.Text = "Upload status: File uploaded!";
        }
        catch(Exception ex)
        {
            StatusLabel.Text = "Upload status: The file could not be uploaded. The following
error occured: " + ex.Message;
        }
    }
}
```

The above example demonstrates the FileUpload control and its properties. The form has a FileUpload control along with a save button and a label control for displaying the file name, file type and file length.

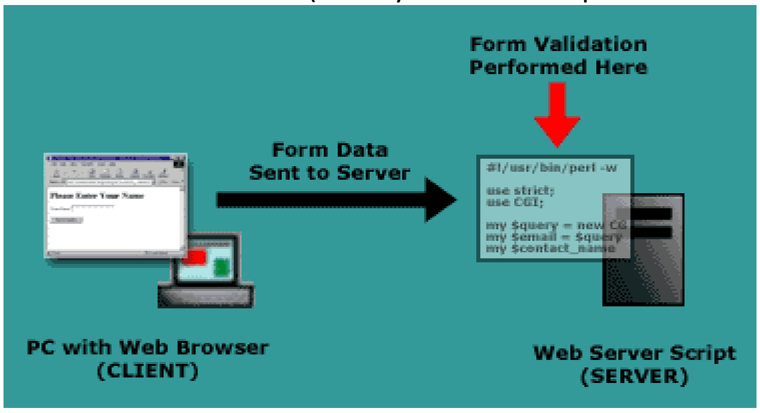
VALIDATION CONTROL

FORM VALIDATION:

Form validation is the process of checking that a form has been filled in correctly before it is processed. For example, if your form has a box for the user to type their email address, you might want your form handler to check that they've filled in their address before you deal with the rest of the form. There are two main methods for validating forms: *server-side* (using CGI scripts, ASP, etc), and *client-side* (usually done using JavaScript). Server-side validation is more secure but often more tricky to code, whereas client-side (JavaScript) validation is easier to do and quicker too (the browser doesn't have to connect to the server to validate the form, so the user finds out instantly if they've missed out that required field!).



Client-side form validation (usually with JavaScript embedded in the Web page)



Server-side form validation (usually performed by a CGI or ASP script)

CLIENT SIDE VALIDATION:

The Validation controls make use of a JavaScript script library that is automatically installed on your server when you install the .NET framework. This library is located in a file named WebUIValidation.js. By default, the validation controls perform validation on both the client (the browser) and the server. The validation controls use client-side JavaScript. This is great from a user experience perspective because you get immediate feedback whenever you enter an invalid
Value into a form field. Client-side JavaScript is supported on any up level browser. Supported browsers include Internet Explorer, Firefox, and Opera. This is a change from the previous version of ASP.NET, which supported only Internet Explorer as an up level browser. You can use the validation controls with browsers that do not support JavaScript (or do not have JavaScript enabled). If a browser does not support JavaScript, the form must be posted back to the server before a validation error message is displayed. Even when validation happens on the client, validation is still performed on the server. This is done for security reasons. If someone creates a fake form and submits the form data to your web server, the person still won't be able to submit invalid data. If you prefer, you can disable client-side validation for any of the validation controls by

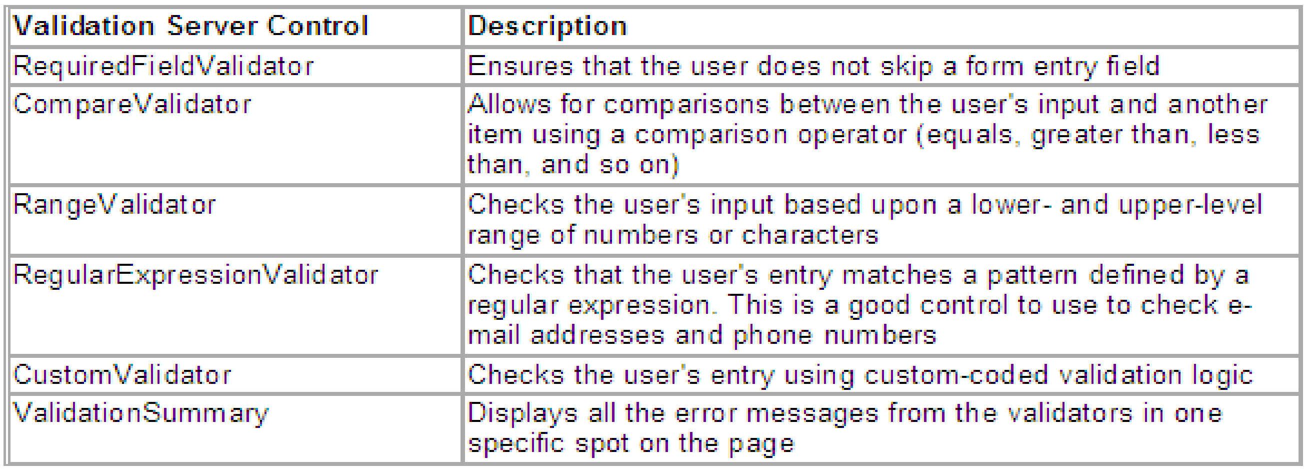
Assigning the value False to the validation control's EnableClientScript property.

SERVER SIDE VALIDATION:

Validation server controls are a series of controls that helps you validate the data that the user enters into the other controls that are provided with ASP.NET. They determine whether the form can be processed based upon the rules that you define in the validation server controls.
Presently, six different validation server controls are available for ASP.NET:
You can also customize validation for your own needs. Then, if there are any errors in the form data, these validation server controls enable you to customize the display of error information on the browser.
You place validation server controls on your page as you would any other type of controls. After the user submits the form, the user's form information is sent to the appropriate validation control, where it is evaluated. If the information doesn't validate, the control sets a page property that indicates this. After all the form information is sent to all the validation server controls, if one or more of the validation server controls cannot validate the information sent to it, the entire form input is found to be invalid, and the user is notified.

VALIDATION CONTROL:

Table : Available validation server controls



Validation Server Control	Description
RequiredFieldValidator	Ensures that the user does not skip a form entry field
CompareValidator	Allows for comparisons between the user's input and another item using a comparison operator (equals, greater than, less than, and so on)
RangeValidator	Checks the user's input based upon a lower- and upper-level range of numbers or characters
RegularExpressionValidator	Checks that the user's entry matches a pattern defined by a regular expression. This is a good control to use to check e-mail addresses and phone numbers
CustomValidator	Checks the user's entry using custom-coded validation logic
ValidationSummary	Displays all the error messages from the validators in one specific spot on the page

All the validation controls are derived from baseValidator class. Some of the properties of baseValidator class can be given as:

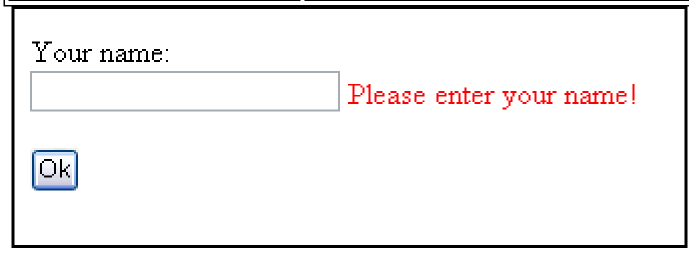
Property	**Meaning**
ControlToValidate	This property is used to set the control which we want to validate. Generally we have would provide the name of control which we want we validate like textbox, listbox, dropdownlist and many other
ErrorMessage	This property is used to set error message which will be displayed when validator control fails. This same message wil be shown for ValidationSummary
IsValid	It returns either true or false after checking the validation control. If any of the validation fails, then IsValid property returns false else vice versa.
Enabled	It is used enable or disable automatic checking of validation control
Text	Used to set error message, which would only be displayed on validator control. ErrorMessage is displayed when we are using ValidationSummary control whicle text will be shown on Validation Control

REQUIRED FIELD:

The RequiredFieldValidator control is used to make an input control a required field. With this control, the validation fails if the input value does not change from its initial value. By default, the

initial value is an empty string (""). Note: The InitialValue property does not set the default value for the input control. It indicates the value that you do not want the user to enter in the input control.
Add a RequiredFieldValidator control to the page and set the following properties:

Property	Description
ControlToValidate	The ID of the control for which the user must provide a value.
ErrorMessage, Text, Display	Properties that specify the text and location of the error or errors that will appear if the user skips the control. For details, see How to: Control Validation Error Message Display for ASP.NET Server Controls.



The following example shows the .aspx file of a TextBox server control with required field validation.

```
<asp:Textbox id="txtLastName" runat="server"></asp:Textbox>
<asp:RequiredFieldValidator id="RequiredFieldValidator1" runat="server"
  ControlToValidate="txtLastName"
  ErrorMessage="Please enter your name!"
  ForeColor="Red">
</asp:RequiredFieldValidator>
```

COMPARISON RANGE – COMPARE VALIDATOR

As the name specifies, this control is used to check value of two controls. It is used to check whether both the controls have got same values or not. If not then it gives an error message else nothing. One of the most common examples of CompareValidator control is password and confirm password. In this we compare whether both the passwords entered are same or not.

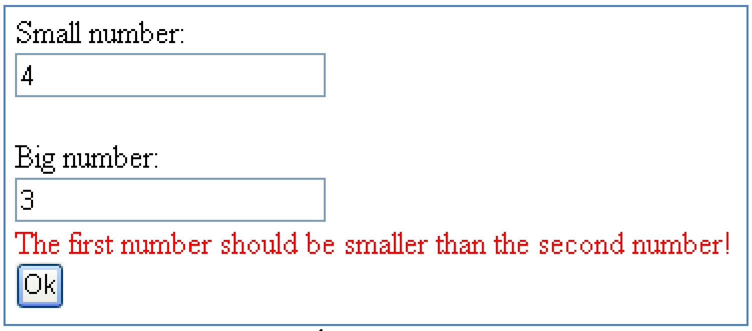
CompareValidator not compare two controls, but it also checks the data types of controls. So for example if we want the control to have only integer value or string value it is possible using this control.

Some examples of CompareValidator are age must not be greater than 20, TextBox should contain a value minimum 8 characters.

Some properties of CompareValidator can be given as below

Property	Description
ControlToCompare	This property is used to set the control which you want to compare with given control. Values of CompareToValidate and control to compare must be same.
ValueToCompare	This property is used to set specific value to be compared with this control.
Type	Specifies the data type of the values to compare. The types are: • Currency • Date • Double • Integer • String
Operator	The type of comparison to perform. The operators are: • Equal

	• GreaterThan • GreaterThanEqual • LessThan • LessThanEqual • NotEqual • DataTypeCheck
ErrorMessage	This property is used to set error message which will be displayed when validator control fails.



Small number:

4

Big number:

3

The first number should be smaller than the second number!

Ok

```
Small number:<br />
<asp:TextBox runat="server" id="txtSmallNumber" /><br /><br />
Big number:<br />
<asp:TextBox runat="server" id="txtBigNumber" /><br />
<asp:CompareValidator runat="server" id="cmpNumbers"
controltovalidate="txtSmallNumber" controltocompare="txtBigNumber"
operator="LessThan" type="Integer" errormessage="The first number should
be smaller than the second number!" /><br />
```

RANGEVALIDATOR CONTROL

The **RangeValidator** does exactly what the name implies; it makes sure that the user input is within a specified range. You can use it to validate both numbers, strings and dates, which can make it useful in a bunch of cases.

Some examples of RangeValidator control are

Input age within a specific range

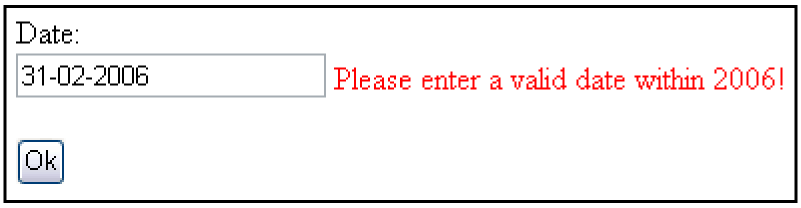
Input Birth data within a range

Some of properties of RangeValidator Control can be given as.

Property	Description
ControlToValidate	The Name of the control to be validated. Here it would be TextBox
Minimum Value and Maximum Value	It indicates the range of data to be entered. Here it would be for minimum value as 01-01-2006 and maximum value as 31-12-2006.
Type	Specifies the data type of the values to compare. The types are: • Currency • Date • Double • Integer • String
ErrorMessage	The Error Message to be displayed when the value entered is not as per the range given. Here it would be “please enter a date within 2006!

```
Date:<br />
<asp:TextBox runat="server" id="txtDate" />
<asp:RangeValidator runat="server" id="rngDate"
controltovalidate="txtDate" type="Date" minimumvalue="01-01-2006"
maximumvalue="31-12-2006" errormessage="Please enter a valid date within
2006!" />
<br /><br />
```

In this example, if you enter a date of 2006, it will not give any message. But if you enter a date which is not between 01-01-2006 and 31-12-2006 it will generate a message "please enter a date within 2006!



Date:

31-02-2006 Please enter a valid date within 2006!

REGULAREXPRESSIONVALIDATOR CONTROL:

This control is used to check the value of given control against a specified expression. It is used to check the value of control against string value, numeric value, alphanumeric values etc.

Some examples of this control are:

1. To input mobile number in specified format such as +91 – 9999999999
2. To input a pin code, code must be of 6 digits only such as 361002
3. To input an email id such as jjkcc@yahoo.com
4. To input a web URL such as http://www.jjkcc.com

A property called ValidationExpression property is used to set the expression format which the user wants.

To use RegularExpressionValidator control, you should be aware different types of RegularExpression symbols. Some of characters are specified as follows:

Character	Meaning
\w	Specifies any word character(alphabets, numbers or underscore).
\d	Specifies any digit.
\D	Specifies any character that is a digit.
\s	Specifies any white space character like tab or space.
\S	Specifies any non white space character.
|	Specifies one of the given choices.
[ ]	Provides a range of characters or set of characters.
[^]	Provides a range of characters not to be included in the input list.
*	Specifies zero or more number of characters.
+	Specifies zero or more number of characters.
{ , }	Provides a range of character or set of characters to be included in input list.

Some of examples can be given as:

Character	Meaning
\w {8,15}	Allows character length between 8 to 15.
\d {6}	Allows digits length of 6 digits only
[Y | y | N |n]	Allows one of characters from Y, y, N or n.
[B – E]	Allows one of character between B and E.
[^B – E]	Allows characters except between B and E.
[B – E]*	Allows zero or more characters between B and E.
[B – E]+	Allows one or more characters between B and E.

I have a text box for mobile number, number which should be only of 10 digits. Here is the code for validating textbox for 10 digit mobile numbers.

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br\>
    <asp:RegularExpressionValidator ID="RegularExpressionValidator1"
runat="server"
      ControlToValidate="TextBox1"
ErrorMessage="RegularExpressionValidator"
    ValidationExpression="[0-9]{10}"> </asp:RegularExpressionValidator>
```

CUSTOMVALIDATOR CONTROL:

CustomValidator Control is used to validate an input with user defined functions. It is used to define user defined function which would validate the user input. These functions are of two types.

1. **Client Side Validation Function:**

This function is executed on client side and is written in JavaScript, VBScript etc which are scripting languages. The name of Client function is specified in Client Validation Function property of CustomValidator control. The Client function has two arguments Object Source (written as ObjSourece) and Object Arguments (written as ObjArgs).

2. **Server Side Validation Function:**

This function is executed on server side and is written in code behind file in any of .NET supported programming language. The name of function is set in the OnServerValidate property of CustomValidator Control.

This function has two arguments:

1. Object source: this gives the source name with the custom validator is connected.
2. ServerValidateEventArgs: this arguments has two important properties about the control value and IsValid
   a. Value : this gives you the value of the control
   b. IsValid: This can be set as true or false. It depends on choice of condition checking.

VALIDATIONSUMMARY CONTROL:

ValidationSummary control is used to show the summary of all the validation controls which are included in the current web page. This control is useful when you have a large web form which has more controls, which may be visible in single screen. There you can use this control to get the summary of validation on the page.

Each validation control has two property names ErrorMessage and Text. ErrorMessage property is used by ValidationSummary control. It displays the message that are shown, when a ValidationSummary control is validated. While the Text property is used by validator control. It displays the message which is shown when Validator control is validated.

Some of the property of ValidationSummary control can be given as:

Property	Meaning
DisplayMode	It is used to display the summary of all the error messages in different ways such as List, BulletList or SingleParagraph. By default , the display mode is of BulletList.
HeaderText	It is used to specify the header text for validationSummary control.
ShowMessageBox	It is Boolean property which is used to see error list as MessageBox
ShowSummary	It is boolen property which is used to see error list as Summary.

INTRODUCTION:

In the web world, for transferring the information from one place to another you are using HTTP protocol. HTTP is a communication protocol which is used by World Wide Web. Client will send request to server and server responds to these requests. HTTP uses TCP protocol for communication between client and server. This communication is done via ports. Once the client receives the response, client will be disconnected from the server. For any new request the same process, is followed again. ASP.NET files are like text files that are located on server. On each request for a page, the server locates the requested file. Then the ASP.NET engine produces the response for that file by processing the server tags and generate HTTML format file for the client.

As HTTP is a stateless protocol, once the client gets the response it terminates the connection with server and does not save any information on HTTP. This means a new instance of web page is created each time the page is posted to a server. It also means that all information associated with page and the controls on the page are lost with each page request response. To overcome the limitations of web programming, stateless management is used. The current value of all the controls and variables for current user in the current session is called sate of web page. State management techniques are used to maintain state and page information of web page through the application for same or multiple requests.

There are two types of state management:

1. Client side state management
2. Server side state management

1. CLIENT SIDE STATE MANAGEMENT

In client side state managements information is stored on client's computer by embedding the information into a web page, a URL or a cookie. The techniques available to store this information on client side can be listed down as below:

(A)View State:

ASP.NET uses view state to store state of the page and all its control. View state is automatically maintained by ASP.NET framework. When a page is sent to the client as per request, the changes in properties and control of pages are determined and stored in the value of hidden input filed. This field is named as _VIEWSTATE. When the page is post back the _VIEWSATE field is sent to the server with HTTP request.

(B)Control State:

It is new mechanism implemented by ASP.NET 2. It is used to store critical, Private Infroamtion across post backs. It is another type of state container which is reserved for controls core behavioral functionality. It is addressed some of the shortcomings of view state. It shares same memory data structures with view state. It is propagated even though the view sate control is disabled.

(C)Hidden State:

They are used to store data at page level. These fields are not rendered by web browser. This data is available only when the form is processed. It like a control whose properties you can set.

(D)Cookies:

It is small piece of text stored on user's computer. Then the browser sends this information with every page request to the same server. It is used by websites to keep track of visitors.

(D)Query State:
They are used to store values in the URL that are visible to user. They are keep track of visitors.

2. Server Side State Management:

(A)Application State:
Application state allows us to save values which are an instance of HTTP application state class for each active web application. It is a global storage mechanism which is accessible from all pages of web application. Application sate is useful for storing information between server round trips and between requests for pages.

(B)Session State:
Session state is quite opposite to cookies. Cookies are used to store the data on client side while sessions are used to store the data on the server side

View State:

Http is a stateless protocol. Hence the state of controls is not saved between postbacks. Viewstate is the means of storing the state of server side controls between postbacks. Viewstate stores the state of controls in HTML hidden fields. In other words, it is a snapshot of the contents of a page.

When set to True, the 'EnableViewState' property enables storing the state of an object in a page between postbacks. Objects are saved in a Base64 encoded string. Because it is a Base64 encoded string, it is not readable by the human eye. However it is also not difficult to decode the viewstate and view the contents of the viewstate when it is passed over the wire
When you view the page source (in your browser) of a page the uses ViewState, you may see this hidden viewstate input which will look something like this:

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUKMTM1ODM3Nj......." />
```

The view state can be enabled or disabled for:
The entire application:
By setting the EnableViewState property of pages attributes in web.config file.
A page:
By setting the EnableViewState attribute for page directive, as

```
<%@ Page Language="C#" EnableViewState="false" %>
```

A Control :
By setting the EnableViewState property of individual control. View state is defined by state bag class which defines a collection of view state items. The state bag is data structure which contains attribute/value pairs. These pairs are stored as strings which are associated with the objects.

Cookies:

Web applications store small piece of information in Client's web browser using cookies. A cookie is a small file that is stored on client side. According to RCF, a cookie can have a maximum site of 4KB. Cookies can be stored either in a textfile on the client mechanism or in memory in the client browser session. The location where the cookies are stored is controlled by the browser.

The web server creates a cookie and attaches an additional HTTP header to the response, and sends it to the browser. The browser would create cookie in client's computer and include this cookie for all requests made to the same client. Servers can read value of cookie from the request and retain that state. For different web applications different cookies are maintained. Cookies are

used to keep the track of visitors on a website. All the cookies would be cleared by browser immediately when we close client's browser
Cookies can be also created by using HttpCookie class. It can be given as below:

```
HttpCookie userInfo=new HttpCookie();
userInfo["UserName"]="jjkcc";
```

Here userInfo is the name of the HttpCookie object created, username is cookie variable and jjkcc is the value of the cookie assigned to that variable.

You can write a cookie by using the page's response property. The response object supports a collection named cookies, to which you can add the cookies you want to write to the browser. It can be created in following way:

```
Response.Cookies["username"].Value="jjkcc";
```

Here username is the name of the cookie variable and jjkcc is the value of assigned to that cookie variable.

As the cookies are stored on client's machine, while adding cookie you must specify Response object to add cookie. As server will send cookie from server to client's matching using response. You can add the userInfo in response by using the following code:

```
Response.Cookies.Add(userInfo);
```

You can retrieve cookies value from response object by using the request object. It can be given as

```
string userName=Request.Cookies["username"].value
```

HIDDEN FIELDS:

Hidden field are the data which is not visible to the user at runtime. Hidden field are sent back to server when the when the user submits a form. However, the information stored in hidden field is never displayed to user unless the user views the page source of the web page.

A hidden field control stores a single variable in its value and property. Hidden field are used to stores data at page level. If you use hidden fields, you must submit your page to the server using HTTP post method rather than HTTP Get method.

QUERY STRING CONTROL:

The QueryString collection is used to retrieve the variable values in the HTTP query string.
The HTTP query string is specified by the values following the question mark (?), like this:

```
<a href= "test.asp?txt=this is a query string test">Link with a query string</a>
```

The line above generates a variable named txt with the value "this is a query string test".
Query strings are also generated by form submission, or by a user typing a query into the address bar of the browser.
Note: If you want to send large amounts of data (beyond 100 kb) the Request.QueryString cannot be used.

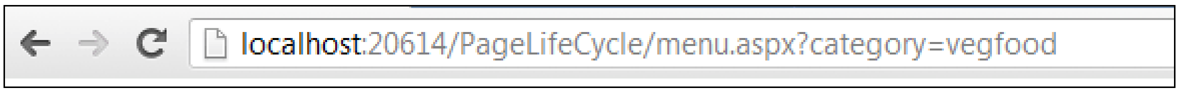
Syntax

```
Request.QueryString(variable)[(index)|.Count]
```

Parameter	Description
variable	Required. The name of the variable in the HTTP query string to retrieve
index	Optional. Specifies one of multiple values for a variable. From 1 to Request.QueryString(variable).Count

For example, when redirecting a request from one page to another page, you can pass the Query String as shown here –

```
Response.Redirect("menu.aspx?category=vegfood");
```


localhost:20614/PageLifeCycle/menu.aspx?category=vegfood

Query string is limited to simple string information, and they're easily accessible and readable.

SESSION STATE:

A session is defined as the period of time that a unique user interacts with a Web application.
ASP.NET session state enables you to store and retrieve values for a user as the user navigates ASP.NET pages in a Web application. HTTP is a stateless protocol. This means that a Web server treats each HTTP request for a page as an independent request. The server retains no knowledge of variable values that were used during previous requests. ASP.NET session state identifies requests from the same browser during a limited time window as a session, and provides a way to persist variable values for the duration of that session. By default, ASP.NET session state is enabled for all ASP.NET applications.
Alternatives to session state include the following:

- Application state, which stores variables that can be accessed by all users of an ASP.NET application.
- Profile properties, which persists user values in a data store without expiring them.
- ASP.NET caching, which stores values in memory that is available to all ASP.NET applications.
- View state, which persists values in a page.
- Cookies.
- The query string and fields on an HTML form that are available from an HTTP request.

Storing and retrieving a value in the Session is as simple as:

```
Session["Name"] = "JJKCC";
Name = (string)Session["Name"];
```

By default the Session will be created within the same process that your web site runs in (InProc). This is controlled by a setting in the web.config file:

```
<sessionState mode="InProc" />
```

Although running the Session In Process is very convenient, it does mean that all Session values will be lost whenever the application recycles (such as when deploying updates) . There are alternate modes you can use that will allow the Session state to survive even when the application recycles. The available options are:

- Off - No session state will be stored
- InProc - (The Default) Session state exists within the process the web is using
- StateServer - Session data is sent to the configured stateserver service
- SQLServer - Session data is store in the configured sql server database

Both the StateServer mode and the SQLServer mode allow Session state to survive an application recycle. But, when storing reference type objects (such as class instances), they can only be stored to StateServer or SQLServer if they have been marked with the Serializable attribute.
The Session Timeout is adjustable through a web.config setting but increasing the timeout value can put memory pressure on your server that may be undesirable.

```
<sessionState timeout="number of minutes" />
```

Other commonly used Session methods are:

- Session.Abandon() - removes the Session and all items that it contains
- Session.Clear() - removes all items from the Session
- Session.RemoveAll() - removes all items from the Session
- Session.Remove("itemName") - removes the item that was stored under the name "itemName"

APPLICATION STATE:

ASP.NET provides you a means of saving values using application state. It is a global storage mechanism that is accessible from all pages in the web application. It is stored in application key value dictionary. Once application specific information is added to application state, the server manages it, and it is never exposed to the client. Adding data to application state makes the data accessible to all pages from a single location in memory, rather than keeping separate copies of the data. Data stored in application state object is not permanent, it gets lost time the application is restarted.
ASP.NET creates an application state object for each application by using the HTTPApplicationState class and stores this object in server memory. This object is represented by file global.asax. Application state is used mostly to store hit counter and other statistical data, global application data like tax rate, discount rate etc and to keep track of users visiting the web application.

Advantages of application state:

- Application object memory released when we removed.
- Multi user can able to access application variable.
- To avoid deadlock or conflict we should use Lock and Unlock when we use write or update in the application object.
- Other application can't access this application values.

Disadvantages of application state:

- Application variable will exist until exit our application.
- If we do not have Lock () and Unlock, deadlock will occur.
- Its gloabal variable so anyone can access within this application.

INTRODUCTION:

Data Access is the main part of any application that you develop. More concern is given on how you store the data and manipulate the data. Any web based application or window based applications stores the data for accessing the data for different purpose.

ADO.NET is a group of libraries provided by .NET framework which are used to create powerful database using various sources such as MS SQL, Microsoft Access, Oracle, XML, etc. it relies on various these classes to process requests and performs transition between a database system and the user. It provides classes for connecting with a data source, submitting queries, and processing results.

Thus ADO.NET is a large set of .NET classes which enables us to retrieve and manipulate data, and update data sources in many different ways.

ADVANTAGES OF ADO.NET:

1. **Connection with any database :**

ADO.NET can be used to connect with any type of database such as SQL Server, Oracle, and Access MySql etc. it can also connect with any third party database.

2. **Connected and Disconnected Architecture :**

It supports both connected and disconnected architecture. Connected architecture is supported by DAO, ADO, RDO or ODBC, disconnected is new feature provided by ADO.NET. you will study about the disconnected architecture in the future topics.

3. **XML Support :**

ADO.NET provides the new feature of XML supported, today XML is used widely for transfer data globally. XML is standard data transfer features for any type of data transfer. Using ADO.NET you can easily convert data from DataSet to XML and perform operation in secure and fast way.

4. **Fast, Scalable and Standard:**

ADO.NET is quite fats then other applications. It is quite scalable i.e. it can be expected. It also provides XML and other features thus it is quite standard used.

5. **Cross language Support:**

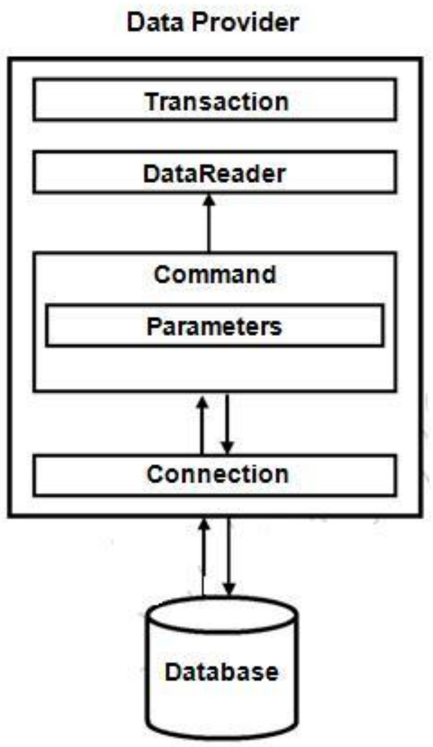
ADO.NET provides supports for many languages. You can use VB, C# or J# for ADO.NET programming. So it provides cross language support features.

ARCHITECTURE OF ADO.NET (Connected and Disconnected):

ADO.NET is a data access technology from Microsoft .Net Framework, which provides communication between relational and non-relational systems through a common set of components. ADO.NET consists of a set of Objects that expose data access services to the .NET environment. ADO.NET is designed to be easy to use, and Visual Studio provides several wizards and other features that you can use to generate ADO.NET data access code. ADO.NET provides two types of architecture for accessing data via database: Connected and Disconnected Architecture .

CONNECTED ARCHITECTURE:

The architecture of ADO.NET in which connection to the database must be opened to access the data retrieved from database is called connected architecture. It is built on the classes called connection, command, datareader and transaction.



Components of Connected Architecture:

Connection: in connected architecture also the purpose of connection is to just establish a connection to database and itself will not transfer any data.

DataReader : DataReader is used to store the data retrieved by command object and make it available for .net application. Data in DataReader is read only and within the DataReader you can navigate only in forward direction and it also only one record at a time.

Command: command object is used to execute SQL queries against database. Using command object you can execute select, insert, update and delete SQL command.

Transaction: enables you to get the list of commands in transactions at the data source.

DISCONNECTED ARCHITECTURE:

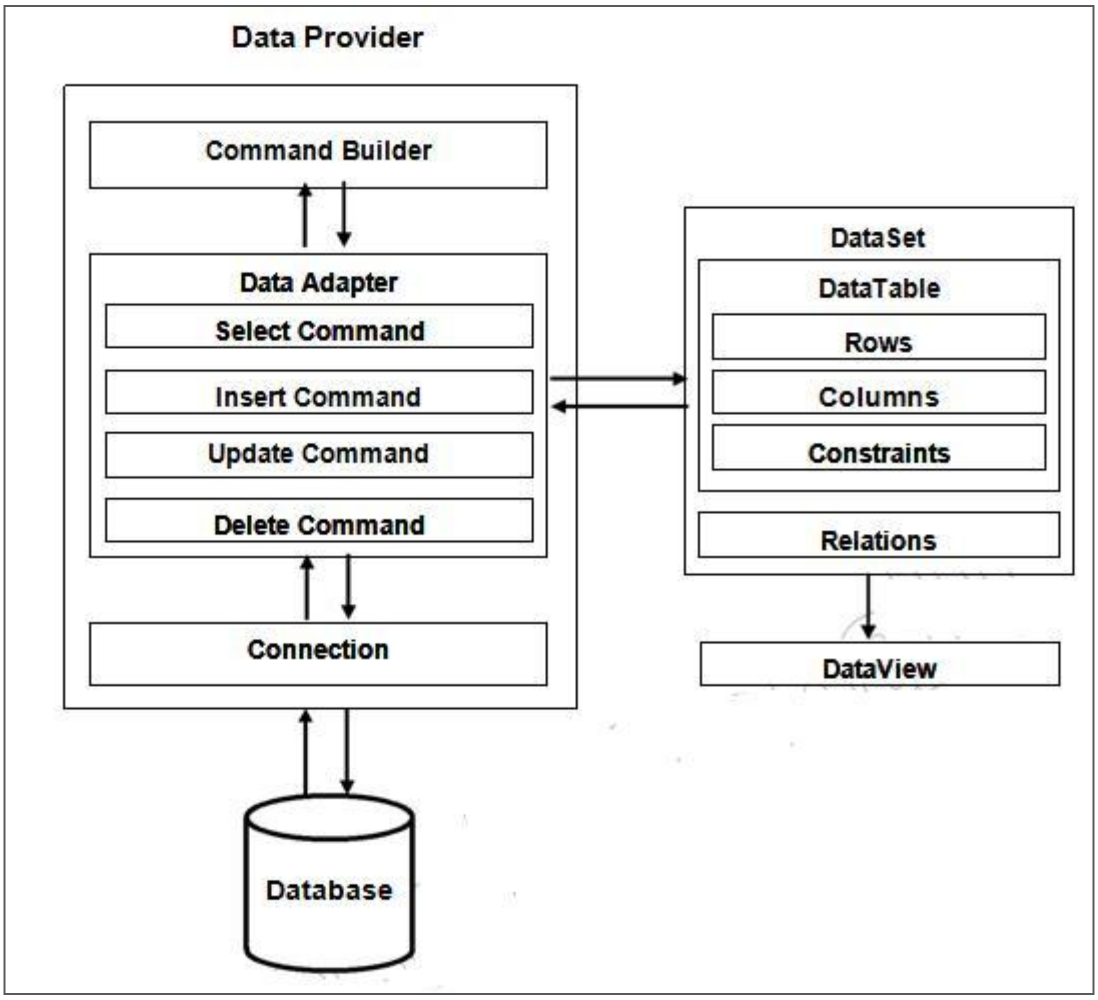
The architecture of ADO.net in which data retrieved from database can be accessed even when connection to database was closed is called as disconnected architecture. Disconnected architecture of ADO.net was built on classes connection, dataadapter, commandbuilder and dataset and dataview.

In disconnected architecture, the copy of the data is loaded from database and kept in local memory of client. Database is only connected during the copying of the database. After copying the databse the databse the connection to the databse is lost. You can perform insert, update, or select operation on the local of the database. The updated database is in our local memory only. Now when you connect to database again memory only. Now when you connect to database again then the local copy of the database is copied into the final database. In this way data consistency is maintained.

Components of DisConnected Architecture:

Connection : Connection object is used to establish a connection to database and connection it self will not transfer any data.

DataAdapter : DataAdapter is used to transfer the data between database and dataset. It has commands like select, insert, update and delete. Select command is used to retrieve data from database and insert, update and delete commands are used to send changes to the data in dataset to database. It needs a connection to transfer the data.



CommandBuilder : by default dataadapter contains only the select command and it doesn't contain insert, update and delete commands. To create insert, update and delete commands for the dataadapter, commandbuilder is used. It is used only to create these commands for the dataadapter and has no other purpose.
DataSet : Dataset is used to store the data retrieved from database by dataadapter and make it available for .net application.
To fill data in to dataset **fill()** method of dataadapter is used and has the following syntax.

```
Da.Fill(Ds);
```

When fill method was called, dataadapter will open a connection to database, executes select command, stores the data retrieved by select command in to dataset and immediately closes the connection.

DIFFERENCE BETWEEN CONNECTED AND DISCONNECTED ARCHITECTURE:

NO	CONNECTED ARCHITECTURE:	DISCONNECTED ARCHITECTURE:
1.	It is connection oriented architecture.	It is connection less architecture.
2.	It uses DataReader for communication between Database and application.	It uses datasets for communication between database and application.
3.	It gives faster performances the application is always connected with the database.	It gives low speed and performance as the database is not connected with the database at all time.
4.	Connected architecture can work with a single table at a time.	Disconnected architecture can work with multiple tables at a time.
5.	DataReader cannot persist the data.	DataSet can persist the data.

6.	It is only used Read only mode, you cannot update the data.	In this we can update the data, as the data is stored locally.
7.	More traffic would be creating if the number of users increases on the server side.	In this there is no concern on the traffic as the data is copied locally.
8.	Much memory of the server is used while processing connected architecture	Much memory of server is used once only when you copy the data into local memory, after that local memory of each user would be used.

DATA PROVIDERS IN ADO.NET

A set of libraries that is used to communicate with data source. Eg: SQL data provider for SQL, Oracle data provider for Oracle, OLE DB data provider for access, excel or mysql. Data providers are the one which does the work of communication between application and database. They are responsible for maintaining the connection to the database. Different classes are provided these data providers which help to perform different database operations either in connected or disconnected modes.

A data provider is a set of classes provided by ADO.NET that allows you to access a specific database, execute SQL commends, and retrieve data providers which allows you to do operations on any of databases. For different types of database, different types of data providers are provided by ADO.NET.

There are 4 types of data providers provided by Microsoft:

SQL Server Provider:

This provider is designed especially to work SQL server database.

Oracle Provider:

This provider deals with oracle database.

OLEDB provider:

This provider works with any database that has OLE DB driver. It provides a means to access global data. It can access any data from any type of database by using OLEDB driver.

ODBC provider:

This provider deals with any database which has ODBC driver. Almost all databases today come with ODBC driver inbuilt. Thus you can connect with any database even through drivers are not provided.

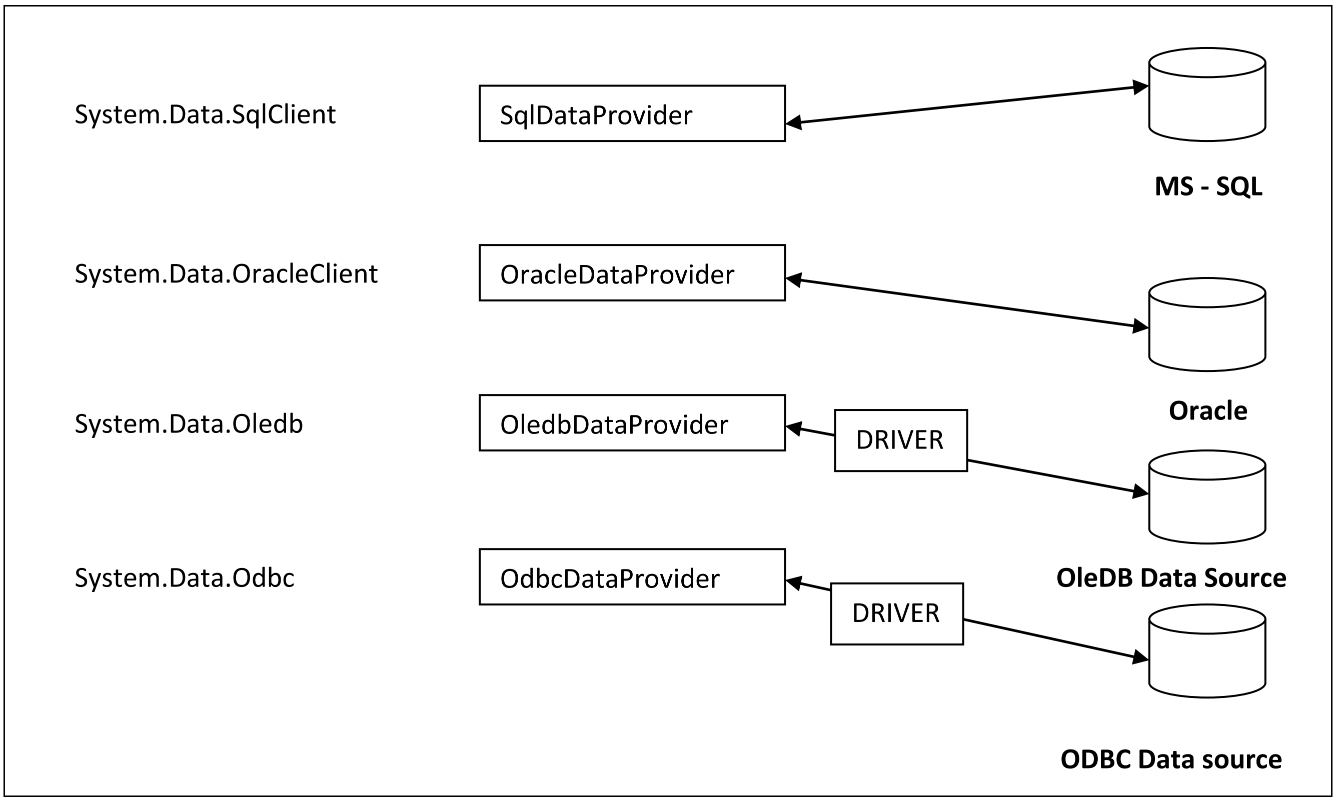
Here in the below diagram you can see SqldataProvider and OracledataProvider directly communicate with your database. It means if you are sure that you want to connect to SQL or Oracle then you can use this provider to get good performance.

For OleDbProvider and OdbcProvider, both require a driver between ADO.Net and your actual database. It is as these providers are meant to work any type of database so you require a driver to which works as a connection between both.

All these data providers provide common component which makes syntax easy to remember even though you change the provider.

The four main components provided by these data providers are:

1. **Connection**: Used for establishing connection with database.
2. **DataReader:** Used to read the data from the database.
3. **Command:** Used to specify the SQL query for retrieving data from the database.
4. **DataAdapter :** Used to fetch the data from database to local memory of client (i.e. DataSet) and also update the back to database.



Differences Between DataSet And DataReader

DATASET	DATAREADER
It is disconnected object and can provide access to data even when connection to database was closed.	It is connected object and cannot provide access to data when connection to database was closed.
It can store data from multiple tables.	It can store data from only one table.
It allows insert, update and delete on data	It is read only and it doesn't allow insert, update and delete on data.
It allows navigation between record either forward or backward.	It allows only forward navigation that also only to immediate next record.
It can contain multiple records.	It can contain only one record at a time.
All the data of a dataset will be on client system.	All the data of a DataReader will be on server and one record at a time is retrieved and stored in datareader when you call the Read() method of datareader.

Command Object In Ado.Net

Command is used to execute almost any SQL command from within the .net application. The SQL command like insert, update, delete, select, create, alter, drop can be executed with command object and you can also call stored procedures with the command object. Command object has the following important properties.

- **Connection :** used to specify the connection to be used by the command object.
- **CommandType :** Used to specify the type of SQL command you want to execute. To assign a value to this property, use the enumeration CommandType that has the members Text, StoredProcedure and TableDirect. Text is the default and is set when you want to execute ant SQL command with command object. StoredProcedure is set when you want to call a

stored procedure or function and TableDirect is set when you want to retrieve data from the table directly by specifying the table name without writing a select statement.

- **CommandText :** Used to specify the SQL statement you want to execute.
- **Transaction :** Used to associate a transaction object to the command object so that the changes made to the database with command object can be committed or rollback.

Command object has the following important methods.

- **ExecuteNonQuery() :** Used to execute an SQL statement that doesn't return any value like insert, update and delete. Return type of this method is int and it returns the no. of rows effected by the given statement.
- **ExecuteScalar() :** Used to execute an SQL statement and return a single value. When the select statement executed by executescalar() method returns a row and multiple rows, then the method will return the value of first column of first row returned by the query. Return type of this method is object.
- **ExecuteReader() :** Used to execute a select a statement and return the rows returned by the select statement as a DataReader. Return type of this method is DataReader.

CONNECTION OBJECT IN ADO.NET

The Connection Object is a part of ADO.NET Data Provider and it is a unique session with the Data Source. In .Net Framework the Connection Object is Handling the part of physical communication between the application and the Data Source. Depends on the parameter specified in the Connection String , ADO.NET Connection Object connect to the specified Database and open a connection between the application and the Database . When the connection is established, SQL Commands may be executed, with the help of the Connection Object, to retrieve or manipulate data in the Database. Once the Database activity is over , Connection should be closed and release the resources .

Some useful properties:

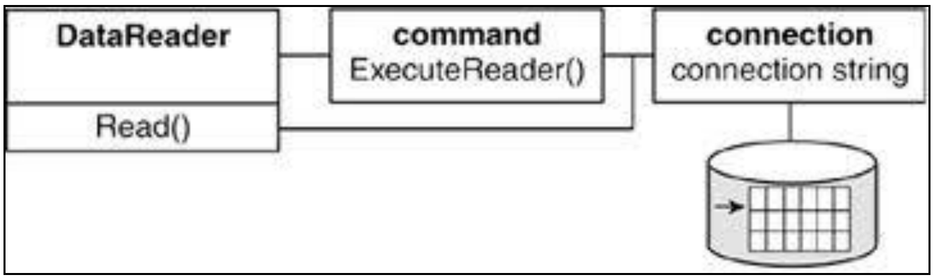
Property	Description
ConnectionString	Sets or returns the details used to create a connection to a data source
ConnectionTimeout	Sets or returns the number of seconds to wait for a connection to open
Provider	Sets or returns the provider name
State	Returns a value describing if the connection is open or closed
Version	Returns the ADO version number

Some useful methods:

Method	Description
BeginTrans	Begins a new transaction
Cancel	Cancels an execution
Close	Closes a connection
CommitTrans	Saves any changes and ends the current transaction
Execute	Executes a query, statement, procedure or provider specific text
Open	Opens a connection
OpenSchema	Returns schema information from the provider about the data source

DATAREADER:

the ADO.NET DataReader to retrieve a read-only, forward-only stream of data from a database. Results are returned as the query executes, and are stored in the network buffer on the client until you request them using the Read method of the DataReader. Using the DataReader can increase application performance both by retrieving data as soon as it is available, and (by default) storing only one row at a time in memory, reducing system overhead.



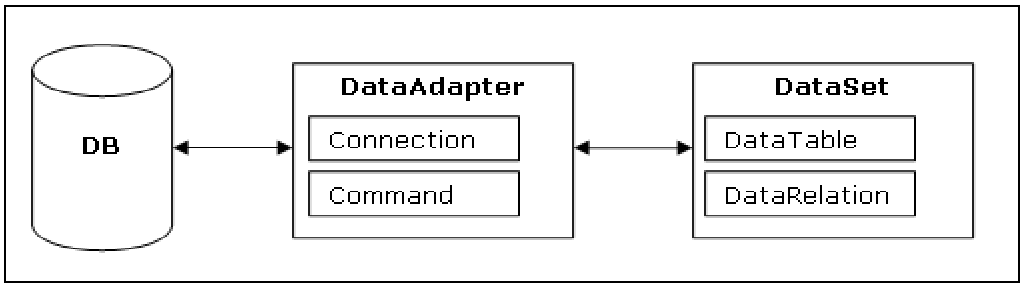
After creating an instance of the Command object, you have to create a DataReader by calling Command.ExecuteReader to retrieve rows from a data source.

```
SqlDataReader reader = cmd.ExecuteReader();
```

When the ExecuteReader method in the SqlCommand Object execute, it will instantiate a SqlClient.SqlDataReader Object. When we started to read from a DataReader it should always be open and positioned prior to the first record. The Read() method in the DataReader is used to read the rows from DataReader and it always moves forward to a new valid row, if any row exist . You should always call the Close method when you have finished using the DataReader object.

DATAADAPTER:

DataAdapter serves as a bridge between a DataSet and SQL Server for retrieving and saving data. We can use SqlDataAdapter Object in combination with Dataset Object. DataAdapter provides this combination by mapping Fill method, which changes the data in the DataSet to match the data in the data source, and Update, which changes the data in the data source to match the data in the DataSet.

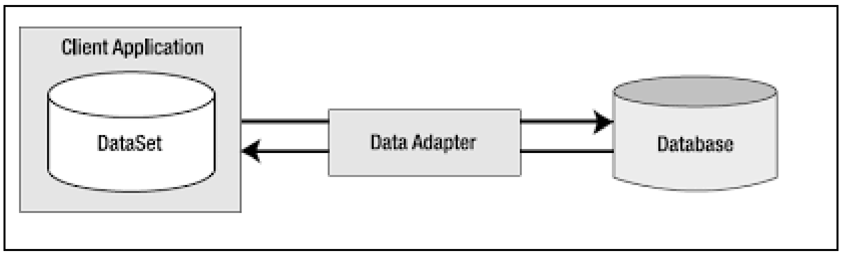


```
SqlDataAdapter adapter = new SqlDataAdapter(sql,connection );
adapter.Fill(ds);
```

The SqlDataAdapter Object and DataSet objects are combine to perform both data access and data manipulation operations in the SQL Server Database. When the user perform the SQL operations like Select , Insert etc. in the data containing in the Dataset Object , it won't directly affect the Database, until the user invoke the Update method in the SqlDataAdapter.

DATASET:

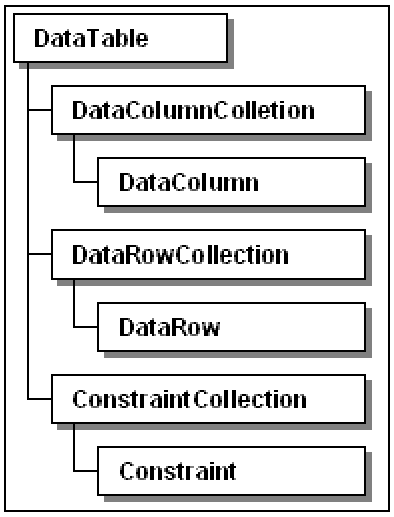
The ADO.NET DataSet contains DataTableCollection and their DataRelationCollection . It represents a complete set of data including the tables that contain, order, and constrain the data, as well as the relationships between the tables.



We can use Dataset in combination with DataAdapter Class . Build and fill each DataTable in a DataSet with data from a data source using a DataAdapter. The DataSet object offers disconnected data source architecture.

DataColumn :

DataColumn is the generalized object which is not related to any specific data provider. It also provided by System.Data namespace.



dataColumn contain a single column of Datatable or DataSet. It has information regarding column like name, data type, default value, maximum length etc.

DataRow:

In **ASP.**Net, **DataRow** class provides the functionality to add a new row i.e. new record into the **DataTable**. **DataRow object** inherits the schema of ASP.Net DataTable and allows you to add the values for each **DataColumn** according to the specified **DataType** for a data column of DataTable. You can use the zero based index number of DataColumn or column name to pass the value for specific column in the new DataRow. Index of the DataColumn depends upon the order of adding a DataColumn to the DataTable.

DataView:

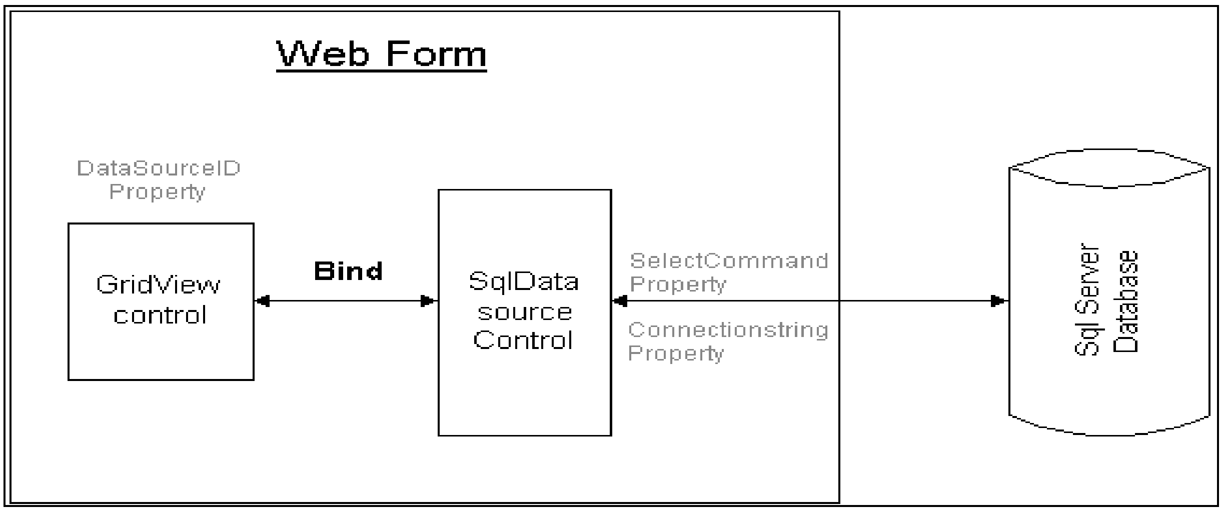
A DataView enables you to create different views of the data stored in a DataTable, a capability that is often used in data-binding applications. Using a DataView, you can expose the data in a table with different sort orders, and you can filter the data by row state or based on a filter expression.

A DataView provides a dynamic view of data in the underlying DataTable: the content, ordering, and membership reflect changes as they occur. This behavior differs from the Select method of the DataTable, which returns a DataRow array from a table based on a particular filter and/or sort order: thiscontent reflects changes to the underlying table, but its membership and ordering remain static. The dynamic capabilities of the DataView make it ideal for data-binding applications.

A DataView provides you with a dynamic view of a single set of data, much like a database view, to which you can apply different sorting and filtering criteria. Unlike a database view, however, a DataView cannot be treated as a table and cannot provide a view of joined tables. You also cannot exclude columns that exist in the source table, nor can you append columns, such as computational columns, that do not exist in the source table.

GridView Control:

GridView control is a successor to ASP.NET DataGrid control. It provides more flexibility for displaying and working with data from your database in comparison with any other controls available. The GridView control enables you to connect to data source and display the data in a tabular format. It also provides options to customize the look and feel.



The GridView view mode is used to display a list of data items by bindings the data fields to columns and by displaying a column header to identify the field. Default GridView Style implements button as column headers. This column header is used for interaction capabilities for example for sorting GridView data according to a specific column.

Repeater Control:

The Repeater control is used to display a repeated list of items that are bound to the control. The Repeater control may be bound to a database table, an XML file, or another list of items.

Repeater is a Data Bind Control. Data Bind Controls are container controls. Data binding is the process of creating a link between the data source and the presentation UI to display the data. ASP .Net provides rich and wide variety of controls, which can be bound to the data.

Repeater has 5 inline template to format it:
1. <HeaderTemplate>
2. <FooterTemplate>
3. <ItemTemplate>
4. <AlternatingItemTemplate>
5. <SeperatorTemplate>
6. <AlternatingItemTemplate>

HeaderTemplate: This template is used for elements that you want to render once before your ItemTemplate section.

FooterTemplate: - This template is used for elements that you want to render once after your ItemTemplate section.

ItemTemplate: This template is used for elements that are rendered once per row of data. It is used to display records

AlternatingItemTemplate: This template is used for elements that are rendered every second row of data. This allows you to alternate background colors. It works on even number of records only.

SeperatorTemplate: It is used for elements to render between each row, such as line breaks.

Binding Repeater Control with Data Source:

Repeater control does not have any specific view or any visual effect. You need to bind repeater control with some data source like dataset or datatable, but for that also we need to specify which column should be displayed manually.

To bind the content of particular column to repeater control column, you need to use DataBinder class as follows:

```
<% DataBinder.Eval(Container.Dataitem,"Name" %>
```

dataBinder class provides an eval() method. This method is used to specify the particular column of Data source with the repeater control's column would be bounded. For example,

```
<% DataBinder.Eval(Container.Dataitem,"EmpID" %>
```

The above code would bind the EmpID from dataset or datatable which contains the data bounded from employee table.

DATABINDING

Data Binding is new concept which is becoming popular these days. Data Binding simply means you are connecting your control with any of the table or table column or data of a particular row. Technically, it means data is retrieved from sources content and then it is connected with the control to property of visual element which we could see.

In ASP.NET when we bind our data with any of the control, it gets you data from specified data source. Data source could be anything such as a dataset, DataTable or even DataRow. But in most of the cases it is either DataSets or dataTable.

The control which is binded with some specific DataSets or DataTable is called Bounded control as it bounded with specific column of the table.

There are basically two types of Binding:

1. Simple Data Binding
2. Complex Data Binding

In simple Data Binding we are connected to any single piece of information, or you can say we are bounded to single column of single row. For example, if textbox is bounded with single column of table, it is called simple data binding.

In complex Data Binding we are connected to entire table itself, or you can also bounded to single column of single row of table in complex data binding. For example, if drop down list is bounded to a single column of table or if DataGrid is bounded to particular table then it is called complex databinding.

The control like datGrid GridView or Repeater control are also known as complex data Binding controls which are bounded to entire table.

Consider the following code for complex data binding

```
GridView1.DataSource=dt;
GridView1.dataBind();
```

In the above code our GridView would be bounded with datatable Object dt. GridView will show the table contained inside datatable dt. For this first we need to specify the datasource property of GridView. Here this is done by assigning DataSource property to dt. Then DataBind property is used to bind the data from datatable dt to gridView.

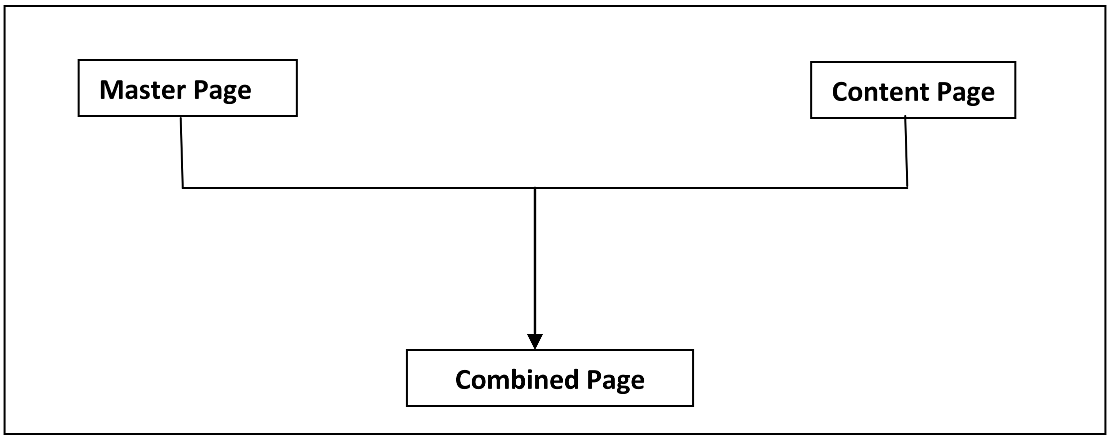
INTRODUCTION TO MASTER PAGE:

When we want to create different web pages that contain common elements and then customize the properties of the control, we need to use master page and themes. For example, if we want the header of all pages of the web application or websites to be same, then we need to use master page.

Master page is a feature in ASP.NET which helps to define the overall layout of the web application and reuse the defined layout in all the different pages which are derived from it. A master page contains markups and controls that can be shared across different web pages of website. This would in turn make the website more manageable and also avoids unnecessary duplication of code.

A master page would define the overall layout of a web application and all the pages would be derived from this master page. A master page can contain markups, controls, banners, navigation menus, and other elements that would be included all the pages of website. The web pages that inherit the properties defined in master pages are called content pages. The content pages displays their own controls as well as the controls inherited from master pages.

To create a master page, first we need to identify the controls that we need to display on all the pages and then add these controls to the master page. Then we need create a ContentPlaceHolder control for the master page to place the content of the web pages. When we execute this website, then the layout of master page and the content in ContentPlaceHolder control are merged to display the out of a web page as shown below in the figure.



REQUIREMENT OF A MASTER PAGE:

It provides centralize common functionality of various pages used in web application such that if we change in one place it would be updated everywhere.

It makes it easy to create a set of controls and code which applied to a set of a pages. For example, we can create a common menu which is applied to all pages.

The master page object models allows us to make changes to the master page from individual content page which in turn allows us to have fine grained control over the rendering of the final page.

Since most of UI code is in the master page, the size of content pages will be considerably small.

CREATING MASTER PAGE:

A master page is an ASP.NET file with extension **.master.** It has a predefined layout that includes static text, HTML elements and server controls. The Master page is identified by a special @master directive. For an ordinary .aspx page we have @page directive.

The @master directive looks like the following:

```
 <%@ Master Language="C#" AutoEventWireup="true"
CodeFile="Masterpage.master.cs" inherits="MasterPage" %>
```

The above property indicates:

1. Language indicates the language in which the page is written. It may be C# or VB.
2. AutoEventWireUp indicates that ASP.NET pages are automatically connected to the event handling functions.
3. CodeFile indicates the code behind attached to the Master page.
4. Inherits indicates the class file inherited by the Master Page.

CREATING CONTENT PAGE:

The content for the Master page's placeholder controls are created by placing individual content pages, which are ASP.NET pages. These pages are bounded to a specific master page. The binding is established by using the @Page directive of content page.

The @Page Directive can be given for master page names MasterPage.mastr can be given as:

```
<%@ Page Langauge="C#" MasterPageFile=~/MasterPage.master
AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default %>
```

The above properties are same as that defined in the @master page directive except the MasterPageFile property. It indicates the name of MasterPage file created by us.

INTRODUCTION TO THEME:

A theme is a collection of property setting which allows us to define the look of pages and control, and then apply that look consistently through all the web pages in the web application. Themes are made of different set of elements such as skins, cascading style sheets (CSS), Images and other resources. Themes are mostly defined in special directories.

ELEMENTS OF THEME:

Skins
Cascading Style Sheets (CSS)
Images and Other Resources

Skins

A skin file has the file name extension .skin and contains property settings for individual controls such as Button, Label, TextBox, or Calendar controls. Control skin settings are like the control markup itself, but contain only the properties you want to set as part of the theme. For example, the following is a control skin for a Button control:

```
<asp:button runat="server" BackColor="lightblue" ForeColor="black" />
```

There are two types of control skins, default skins and named skins:

- A default skin automatically applies to all controls of the same type when a theme is applied to a page. A control skin is a default skin if it does not have a SkinID attribute. For example, if you create a default skin for a Calendar control, the control skin applies to all Calendar controls on pages that use the theme. (Default skins are matched exactly by control type, so that a Button control skin applies to all Button controls, but not to LinkButton controls or to controls that derive from the Button object.)

- A named skin is a control skin with a SkinID property set. Named skins do not automatically apply to controls by type. Instead, you explicitly apply a named skin to a control by setting the control's SkinID property. Creating named skins allows you to set different skins for different instances of the same control in an application.

Cascading Style Sheets(CSS):
A theme can also include a cascading style sheet (.css file). When you put a .css file in the theme folder, the style sheet is applied automatically as part of the theme. You define a style sheet using the file name extension .css in the theme folder.

Images and Other Resources:
Themes can also include graphics and other resources, such as script files or sound files. Mostly the resources files are located in the same folder as the skin flies for that theme.

TYPES OF THEME:

There are two types of Themes:
Page Themes:
A page theme is a theme folder with control skins, style sheets, graphics files and other resources created as a subfolder of the \App_Themes folder in your Web site. Each theme is a different subfolder of the \App_Themes folder.
Global Themes
A global theme is a theme that you can apply to all the Web sites on a server. Global themes allow you to define an overall look for your domain when you maintain multiple Web sites on the same server.
Global themes are like page themes in that they include property settings, style sheet settings, and graphics. However, global themes are stored in a folder named Themes that is global to the Web server. Any Web site on the server, and any page in any Web site, can reference a global theme.

APPLYING THEMES:

We can apply themes to various pages r web applications by using Theme or StylesheetTheme attribute.
Global Themes can be applied setting the <page> elements in web.config file. A theme setting in web.config file is applied to all ASP.NET web paes in the web application. It can be done by adding the following code in web.config file.

```
<configuration>
<system.web>
<pages theme="ThemeName"/>
</system.web>
</configuration>

<configuration>
<system.web>
<pages stylesheetTheme="ThemeName"/>
</system.web>
</configuration>
```

Page Themes can be applied to a web page in web application by using StyleSheetTheme or Theme attributes of the Page Directive. It can be given as:

```
<%@ Page Theme="Themename" %>
<%@ page StyleSheetTheme="ThemeName"%>
```

Difference between Theme and StyleSheetTheme:

The StyleSheetTheme attribute works same as the Theme Attribute. The difference is when the attributes of a particular control are set locally on page; the attributes are overridden by the global theme if we use the theme attribute. While the using StyleSheetTheme, the attribute of a particular control are not overridden they remain same as that defined in the local page.
Suppose we have a textbox control like the following:

```
<asp:textbox ID="TextBox1" runat="server" ForeColor="#ffffff/>
```

In this example, the forecolor attribute is overridden by the theme if we apply theme using the Theme attribute in the page directive. If, we have apllied the theme using the StyleSheeTheme attribute in the page directive, the forecolor attribute remain in place, even if they are explicitly defined in the theme.

Themes vs Cascading StyleSheets:

Themes are quite similar to CSS. By defaults, the themes are defined in CSS file. In both themes and CSS we can define a set of common attributes that can be applied to any web page. However still there are some differences between both of them. Some of these differences can be given as:

1. Themes can be used to define many properties of a control or a web page, it not just only define style properties. For example, themes can use to specify the graphics for a TreeView control, template layout of GridView control, and many others.
2. Themes can include graphics.
3. Themes do not cascade the way the style sheets does. We can define the property values locally as well as globally. This is done by using either Theme or StyleSheetTheme attributes.
4. We can apply only one theme to each page. We cannot apply multiple themes to a page, while multiple style sheets can be applied to one page.

CACHING APPLICATION PAGES AND DATA

INTRODUCTION:

Caching, as the name suggests, is used by applications to store data temporarily either on Web Server or on the Client Side. Caching is mostly used in situations where several programs continuously access the same set of data. So make data access faster, this data is stored in a cache and can be used for future purpose. Usually the data accessing in ASP.NET is quite slow and time consuming as it one of the best alternatives to improve the performance of web application, as it one of the best of connecting to the database and retrieving the data. Caching is one of the best alternatives to improve the performance of web application, as it allows to access data from cache itself, not from the server database or from of application. Caching not reduces the processing time but it also reduces network traffic, there by improves the performance of the web application and enhancing user experience.

CATCHING IN ASP.NET:

The execution of applications in ASP.NET. With ASP.NET, caching techniques has improved dramatically. A number of classes are defined in ASP.NET which specifically deals with issues related to caching of data. Before going into the implementation of caching in ASP.NET.

What is caching?:

Caching is the technique for storing the frequently used data in memory for immediate access to the request program calls. Caching is one of the important features available to improve performance of web applications. It minimizes the usage of server resources for fetching frequently used data, and thus by improving the performance of the application.

Why Caching?:

Every time the page requests some information, it has to go through cycle consisting of client, server and database. Again in the response it has to go through another cycle consisting of database, server and client. This might increase the time, which in turn affect the performance**.**

When multiple users want to access the same information again and again.

In above scenarios caching is considered as one of important aspect as here data would be cached in memory and thus avoiding the database access with every page request, which in turn increases the performance of the applications.

ADVANTAGES / DISADVANTAGES OF CACHING:

ADVANTAGES:

(1) Cache improves the performance of the website.
(2) It takes heavy load or execution form the server for the repeated operations.
(3) It reduces load on database or web services.
(4) Storing data using cache is quite reliable.

DISADVANTAGES:

(1) It will create a delay while accessing website as the data needs to written before accessing.
(2) It would be an overhead for the client to carry out this activity again and again.
(3) It would lead to an increase in maintenance, as the cached data need to get updated regularly.
(4) Again there would also scalability issues.

DIFFERENT TYPES OF CACHING:

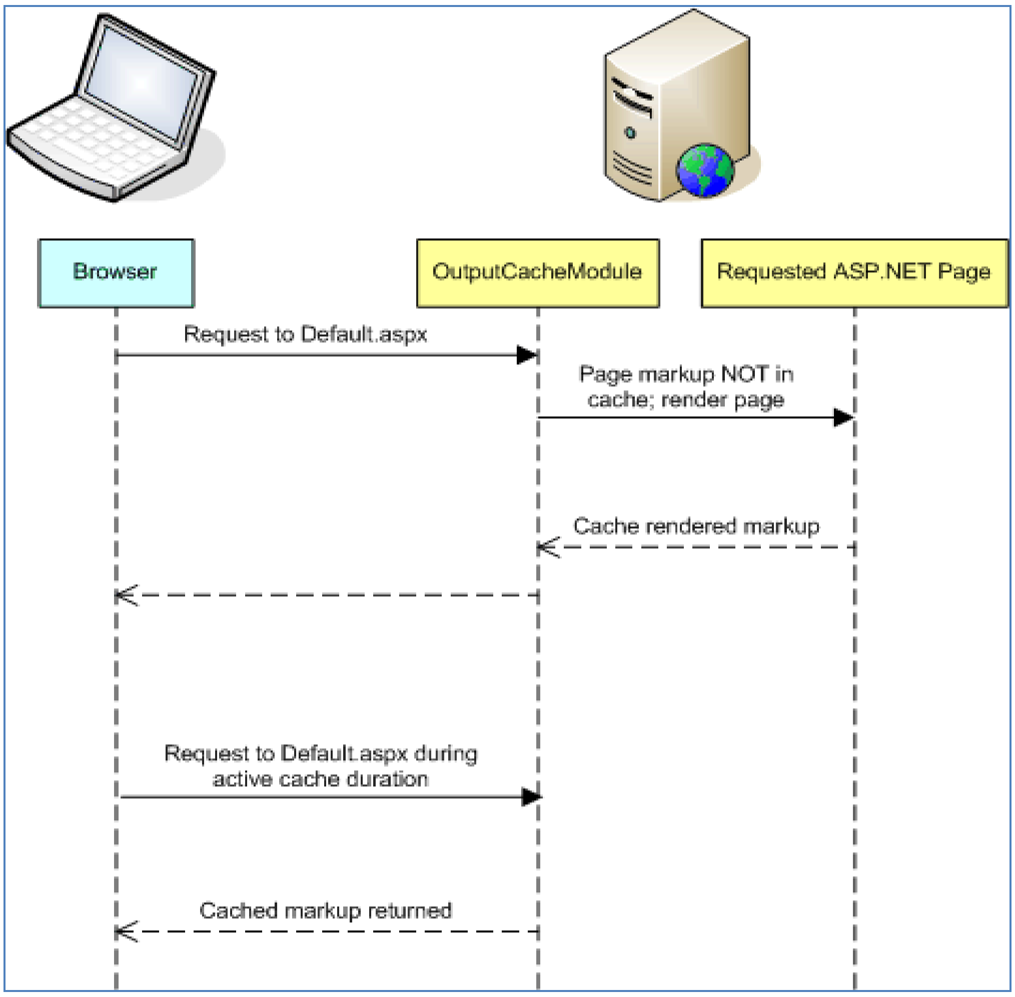
ASP.NET provides different kinds of caching techniques to cache a completely rendered web page. These techniques can be given as:

(1) Page output Caching
(2) Partial Page Caching
(3) Data Caching

(1) PAGE OUTPUT CACHING:

Output caching improves the performance of an ASP.NET application by caching the rendered markup of an ASP.NET web page and serving this content instead of re-rendering the page. For example, imagine that you had an ASP.NET page that displayed the records from a database table named Employees that listed the current employee information. Without caching, each time this page is accessed a connection is made to the database, the table queried, and the results returned to the requesting client. But how often does the employee information change? Probably not more than once a day or so, making many of those requests to the database superfluous. With output caching, when this page is first visited the rendered HTML will be cached for a specified duration. During that time period, if a user requests this page, the cached markup will be returned, thereby saving the database access and page rendering costs.

Cache output is very useful when we have static pages. As with static pages the content of the pages does not change frequently. So cache server would cache the HTML output once and for subsequent requests, the engine sends cached output. Thus it reduces the load on the web server as the page is not rendered each time a request is made; it is cached from the cache server.



As we can see in the diagram, for the first browser makes a request to ASP.NET web page. Since the web is not stored in the cache for the first time, the cache server passes the requested ASP.NET page. Now the request page would be cache and stored on the cache server.

Now for the next request to the same page, the output would be rendered from the cache server it would not go to the requested ASP.NET. Here the cached output would be active for some duration only. So if in that duration if the output is requested then the output would be rendered from the cache server, else it would go the requested web page.

We can impalement page output caching by using @OutputCache directive.

To implement page output caching, add the following code at top of the web page to implement page output caching.

<@OutputCache Duration="10" VaryByParam="none" %>
We can also store the cached data, on client side or on the server side or on proxy by using location attribute in the @OutputCache directive.

We can also store the cahed data, on client side or on the server side or on proxy by using Location attribute in the @OutputCache directive.
<@OutputCache Duration="90" Location ="Any | Client |Downstream |Server |None" VaryByParma="none" %>

<u>(2) PARTIAL PAGE CACHING:</u>
Caching of a complete requested web page is not required always. We may need to implement caching on specific parts of a web page. The remaining content of the web page would be refreshed each time the page is requested. This is called partial page caching or fragment caching.

Steps to apply partial page caching can be given as:

(2) Identify the parts of web page that requires more processing than the other parts, while rendering the page.

(3) Create user control and decide the caching policies for them. Caching policies would check the validity of the cached data on each user request.

(4) Now add the user controls in the web application. Now caching of he defined user controls would only be done. Other controls would be refreshed for each request.

(3) <u>DATA CACHING:</u>

Data caching is a technique used to store the data in the cache memory. Here we would retrieve the data from the database and store the data in the data cache. It is quite efficient to retrieve data from the cache rather than from the database or other sources. For this we need to use Syste.Web.Caching namespace available in ASP.NET. The scope of this data cache is within the application domain. Every user who accesses the website can have access to this object.

Data caching deals with the database. So when the client sends a request to the server, the server would execute the stored procedure, functions to the browser. Again if the request comes the same procedure would be executed.

So at the very first time we would retrieve the records from the database and store them in the cache memory. IIS would provide the cache memory to store the cached data. If the requests come again then data would be cached from cache Memory (IIS). Here also we can set the expiration period time after which the data would be again cache from database and stored in the cache memory.

For creating a data cache, we have to use cache keyboard. It is available in the System.Caching namespace. The basic syntax can be given as:

Cache ["VaraibleName"] ="value"

Here VariableName indicates the name of the variable to which the caching would be done. Value indicate the assigned to cached variable name specified in the brackets.

There are three main things needed to perform data caching:

1. Inserting Value in Cache
2. Retrieve the Cached Value
3. Deleting the cached Value

1. Inserting Value in cache:

For inserting a value in Cache, ASP.NET provides 4 methods. These methods can be given as:

Method Name	Description
Cache.Insert(Key, Value)	This method inserts a value into cache with key name and value with default priority and expiration.
Cache.Insert(Key, Value, Dependencies)	This method inserts a value into cache with key name and value with default priority and expiration, with a CacheDependency. When any of the dependency changes, then the object becomes invalid and is removed from the cache. If there are no dependencies, then this parameter has a null value.
Cache.Insert(Key, Value, Dependencies, AbsoluteExpiration, SlidingExpiration)	This method adds an expiration policy along with all things mentioned above
Cache.Insert(key,Value,	This method adds a priority to inserted value

Dependencies,AbsoluteExpiration,SlidingExpiration, Priority	in the cache. The key name with low priority is deleted before the key name with higher priority. Priority is an object of CacheItem priority Enumeration. It has the following possible values: 1. AboveNormal 2. BelowNormal 3. Default 4. High 5. Low 6. Normal 7. NotRemovable

2. Retrieve the in cache Data:

Retrieving the cached data from web page is one of the simplest tasks in ASP.NET. we need to simply specify the key and value that represent the cached data. For this, we need to write he following line of the code:

Value=Cache("key");

Here value would now contain the data cached from the variable named as key. Each time before using the retrieved value from the cache we need to check whether the retrieved value is null or not.

3. Retrieve the in cache Data:

Basically data would be automatically deleted from the cache on the basis of its expiration policy. To delete the data from cache, we need to write the following line of code:

Cache.Remove("key");

Here key indicates the cached value to be deleted from the cache.

XML parser:
As we know hat in XML we can define our own tags. As we define our own tags we have our own set of rules. Now we need to check whether the tags which we define are following the standards provided by XML or not. This is done by using XML parser.
Xml parser is small software which would read XML file for two purposes:

1. It checks whether the XML file is well formed and verified or not.
2. It also makes sure that XML file meets the defined structure, validation and constraints or not.

WRITING DATASETS TO XML:

.NET provides a namespace which is used to work with XML file called Syste.XML namespace. You can write data from DataSet or DataTable in XML file using two methods. They can be defined as:
1. WriteXml () method: it is used to read data from Dataset and write data as XML data. When we write a data from DataSet we must follow following rules:
- All tables of datasets are treated as main elements of XML file.
- All rows of tables are treated as Sub Elements of XML file.
- All columns of tables are treated as Data Elements which contain the actual data of table.
- Also there are only one root elements as XML rule.

2. WriteXmlSchema() method: it is small bit of XML Schema Definition(XSD) file of XML basics part. XML Schema is a file which contains rules contains the rules for defining XML elements and their data.
If we use only WriteXml() method, the only data would be written from datasets to XML file. But with WriteXmlSchema() method, the schema information would be write with the XML file.

Example:

```
using System.Data.Oledb;
protected  void addbtn_click(object sender,EventArgs e)
{
        Oldbconnection con=new
oledbconnection("provider=Microsoft.jet.oledb.4.0;= Data Source="
+Server.MapPath("database\\staff.mdb"));
oledbDataAdapter da=new oledbdataAdapter("select * from student",con);
dataset ds=new dataset();
da.fill(ds);
ds.WriteXml(Mappath("abc.xml"));
Gridview1.dataSource=ds;
Gridview1.dataBind();
}
```

READING DATASETS WITH XML:

You can read XML file and store the information in form of dataset or DataTable by using two methods. These methods can be given as:
ReadXml()method:
It is used to read XML data from XML file to a dataset or datatable. When we reas dataset or dataTable it follows following rules:
- All main elements under root elements eelemnets are treated as different tables.
- It would automatically store data in XML way.

ReadXmlSchema():

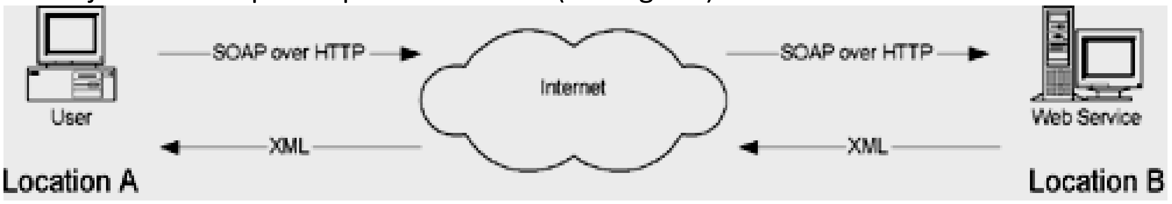
It is used to read XML Schema definition file from XML files. XML Schema is a file which has rules for defining XML element and their data.
If we read data using ReadXml() method then it would just read the entire data from the XML file and store the data into datasets or DataTable. But before reading you need to specify some rules than it can be mentioned by using ReadXmlSchema() method.

EXAMPLE:

```
DataSet ds=new DataSet()
ds.ReadXml(MapPath("add.xml"));
Gridview1.DataSource=ds;
GridView1.DataBind();
```

WEB SERVICES INTRODUCTION:

Web Services provide a simple technique for accessing a method on an object that is running on a local or remote computer. Web Services can be accessed by an application written in any language and running on any operating system. They utilize HTTP as the underlying transport, which allows function requests to pass through corporate firewalls (as they move through port 80). In ASP.NET we can write a Web Service as simple as a normal business object and the only difference is that the functions are preceded with a special <WebMethod ()>attribute that makes them as Web Services. Typical implementation of Web Services might include shipment tracking, credit card verification, searching for airfare rates, Weather forecasting, etc.
Web Services are a very general model for building applications and can be implemented for any operation system that supports communication over the Internet. Web Services use the best of component-based development and the Web. Component-base object models like Distributed Component Object Model (DCOM), Remote Method Invocation (RMI), and Internet Inter-Orb Protocol (IIOP) have been around for some time. Unfortunately all these models depend on an object-model-specific protocol. Web Services extend these models a bit further to communicate with the Simple Object Access Protocol (SOAP) and Extensible Markup Language (XML) to eradicate the object-model-specific protocol barrier (see Figure).



How is the user at Location A aware of the semantics of the Web Service at Location B? This question is answered by conforming to a common standard. Service Description Language (SDL), SOAP Contract Language (SCL) and Network Accessible Specification Language (NASSL) are some XML-like languages built for this purpose. However, IBM and Microsoft recently agreed on the Web Service Description Language (WSDL) as the Web Service standard.

```
[WebMethod]
  public string Add(int a, int b)
  {
    return Convert.ToString(a + b);
  }

Service1 ws=new Service1();
int a=Convert.ToInt32(TextBox1.Text);
int b=Convert.ToInt32(TextBox2.Text);
TextBox3.Text=ws.Add(a,b);
```

SOAP (Simple Object Access Protocol):

SOAP is a simple XML-based protocol to let applications exchange information over HTTP.

What is SOAP?

- SOAP stands for Simple Object Access Protocol
- SOAP is a communication protocol
- SOAP is for communication between applications
- SOAP is a format for sending messages
- SOAP communicates via Internet
- SOAP is platform independent
- SOAP is language independent
- SOAP is based on XML
- SOAP is simple and extensible
- SOAP allows you to get around firewalls
- SOAP is a W3C recommendation

Why SOAP?

It is important for application development to allow Internet communication between programs. Today's applications communicate using Remote Procedure Calls (RPC) between objects like DCOM and CORBA, but HTTP was not designed for this. RPC represents a compatibility and security problem; firewalls and proxy servers will normally block this kind of traffic. A better way to communicate between applications is over HTTP, because HTTP is supported by all Internet browsers and servers. SOAP was created to accomplish this. SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.

WEB SERVICE DESCRIPTION LANGUAGE:

WSDL (Web Services Description Language) is an XML-based language for describing Web services and how to access them.

What is WSDL?

- WSDL stands for Web Services Description Language
- WSDL is written in XML
- WSDL is an XML document
- WSDL is used to describe Web services
- WSDL is also used to locate Web services
- WSDL is a W3C recommendation

To be able to use web service, the developers need to know the methods exposed by web service and the parameters to be passed to these methods. As we know that web service are platform independent. The parameters of web method, return value and body, everything also be in standard format so that it can be shared among all platforms.

This is achieved by using XML based description language called WSDL (Web Services Description Language). WSDL is a language which is used to describe web service and its web methods. In short, WSDL is a markup language that describes a web services and all its WebMethod.

UDDI (Universal Description Discovery and Integration):

UDDI (Universal Description, Discovery, and Integration) is a specification designed to allow businesses of all sizes to benefit in the new digital economy. There is a UDDI registry, which is open to everybody. Membership is free and members can enter details about themselves and the services they provide. Searches can be performed by company name, specific service, or types of service. This allows companies providing or needing web services to discover each other, define how they interact over the Internet and share such information in a truly global and standardized fashion.

BUILDING & CONSUMING A WEB SERVICE:

Web Services, as known as XML Web Services, are units of application logic that provide data and services to other applications using Internet (or Intranet). Applications access Web Services via different kinds of Web protocols and data formats such as XML, with no need to worry about how

Web Services are implemented. They can be implemented for any operation system that supports communication over the Internet and Intranet. They are the cornerstones of the Microsoft .NET programming model.

Web Services can be accessed by any language, using any component model, and running on any operating system. They utilize HTTP as the underlying transport, which allows function requests to pass through corporate firewalls. XML is used to format the input and output parameters of the request, so the request is not tied to any particular component technology or object calling convention. The Microsoft .NET Framework makes **Web Services** easy to write components that communicate using HTTP and SOAP.

Web Services are similar to the regular services in our life. For example, if you have a car that needs gasoline to run, you do not need to have your own gas pump at home. All you need to do is drive to a nearby gas station, fill up the gas tank, and you are on your way. You have received a service from the gas station, Web Services are like this as well. Now, imagine that you are going to build a web site or a Internet system for you company, you can use **Web Services'** "services" to get a complicated application. That should lighten your work load.

<u>Building Web Services</u>

Before you start to create your Web Services, there are some of concepts you have to know. The two most important things are the Simple Object Access Protocol (SOAP) and XML. SOAP is a way for applications to communicate with one another over the Internet, independent of platforms. Unlike Http-Get and Http-Post, only can send name/value, but SOAP can send various rich data type, classes, objects, and others. XML is a pared-down version of SGML, designed especially for Web documents. It allows designers to create their own customized tags, enable the definitions, transmission, validation, and interpret data between applications and between organizations. SOAP relies on XML.

Web Services relies on XML-formatted messages to send data and receive commands. Web Services support three stardard protocols, Http-Get, Http-Post, and SOAP. In Web Services , the Http protocol only can be use the simple data. We do not discuss it here. All Web Services exmaples in this artical use SOAP communication.

Now let's explore how to build a **Web Service** at server. There are two ways to do so. If your Web Service is quite simple, you can directly save it as a *.asmx* file, such as TempConvertorCsharp.asmx. In some complicated applications, especially for database related applications, I prefer to use components. For instance, The DB Query1/2-Web Service at my .NET tutorial, I created the classes as component first, then created a *.asmx* Web Service file that called the component (to create a component, see my .NET tutorial for details.) The whole structure of the service is "UI-Component(Web Service classes)-Database" known as 3-tier archecturre.

Now let's create a Web Service that converts degree Fahrenheit and Celsius each other using C#, and using directly save it as **.asmx**.

```
  [WebMethod]
Public string oddevenNo(int no)
{
  string str;
  if(no % 2 ==0)
  {
       str="No is even";
  }
  else
  {
        str="No is odd";
  }
 return str;
}
```

This method has one argument which contains integer number. If the specified number is even or odd it would return an appropriate message stating whether the number is odd or even.
Consuming Web Services

ASP.NET APPLICATION CONFIGURATION & DEPLOYMENT OF APPLICATION:

INTRODUCTION:

ASP.NET provides a security model which makes easy to protect our web application. Different configuration files provided by ASP.NET help us to make the web application more secure. All the files which are used for storing configuration information are XML based.

ASP.NET CONFIGURATION:

ASP.NET configuration is done using XML file format as XML is a platform independent so it could be applied to any machine and platform. ASP.NET configurations are created as soon as we create our web application. We can also add configuration even after the creation of web application.

Some of the benefits of XML based configuration files are as follows:

- XML is a platform independent so it can be stored and retrieved in any platform.
- When any of configuration settings are changed we do not need to restart web server again and again. ASP.NET configuration changes automatically and applies it to web application.
- Configuration files are in XML format so it can be opened with any simple text editor. We do not need special tools for configuring web file.

Configuration can be done at different levels. We can do configuration as follows:

- Configure IIS to configure all web applications under a machine.
- Configure ASP.NET web application and all its page under it.
- Configure individual web page of ASP.NET application.

If we have installed IIS, it provides MMC (Microsoft Management Console) interface to manage web sites that are stored under IIS. This interface is also known as ISM (Internet Service Manager) which is located in Administrative Tools under Control Panel.

COMMON CONFIGURATION

TRACING:

Tracing is a way to monitor the execution of your ASP.NET application. You can record exception details and program flow in a way that doesn't affect the program's output.

Page level Tracing:

ASP.NET tracing can be enabled on a page-by-page basis by adding "Trace=true" to the Page directive in any ASP.NET page:

```
<%@ Page Language="C#" Trace="true" TraceMode = "SortByCategory"
Inherits  = "System.Web.UI.Page" CodeFile="Default.aspx.cs" %>
```

Additionally, you can add the TraceMode attribute that sets SortByCategory or the default, SortByTime. You can use SortByTime to see the methods that take up the most CPU time for your application. You can enable tracing programmatically using the Trace.IsEnabled property.

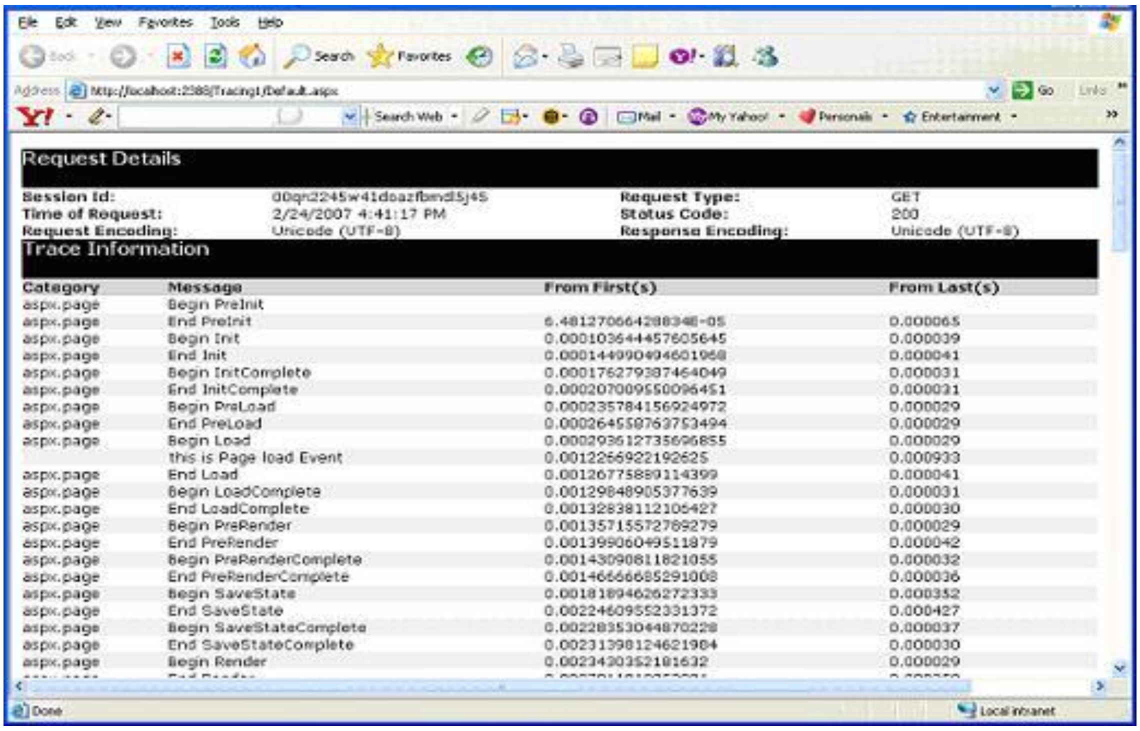
Application Tracing:

You can enable tracing for the entire application by adding tracing settings in web.config. In below example, pageOutput="false" and requestLimit="20" are used, so trace information is stored for 20 requests, but not displayed on the page because pageOutput attribute is set to false.

```
<configuration>
    <appSettings/>
    <connectionStrings/>
    <system.web>
        <compilation debug="false" />
        <authentication mode="Windows" />
      <trace enabled ="true" pageOutput ="false" requestLimit ="20"
traceMode ="SortByTime "  />
```

```
    </system.web>
</configuration>
```

The page-level settings take precedence over settings in Web.config, so if enabled="false" is set in Web.config but trace="true" is set on the page, tracing occurs. When you run the page, you can see a great deal of trace information in the browser



```
Address  http://localhost:2388/Tracing1/Default.aspx

Request Details
Session Id:          d0qn2245w41doazfbndl5j45      Request Type:        GET
Time of Request:     2/24/2007 4:41:17 PM          Status Code:         200
Request Encoding:    Unicode (UTF-8)               Response Encoding:   Unicode (UTF-8)
Trace Information
Category    Message                      From First(s)             From Last(s)
aspx.page   Begin PreInit
aspx.page   End PreInit                  6.48127066428834E-05      0.000065
aspx.page   Begin Init                   0.000103644457605645      0.000039
aspx.page   End Init                     0.000144990494601968      0.000041
aspx.page   Begin InitComplete           0.000176279387464049      0.000031
aspx.page   End InitComplete             0.000207009550096451      0.000031
aspx.page   Begin PreLoad                0.000235784156924972      0.000029
aspx.page   End PreLoad                  0.000264558763753494      0.000029
aspx.page   Begin Load                   0.000293612735696855      0.000029
            this is Page load Event      0.00122669221926 25       0.000933
aspx.page   End Load                     0.00126775889114399       0.000041
aspx.page   Begin LoadComplete           0.00129848905377639       0.000031
aspx.page   End LoadComplete             0.00132838112105427       0.000030
aspx.page   Begin PreRender              0.00135715572789279       0.000029
aspx.page   End PreRender                0.00139906049511879       0.000042
aspx.page   Begin PreRenderComplete      0.00143090811821055       0.000032
aspx.page   End PreRenderComplete        0.00146666685291008       0.000036
aspx.page   Begin SaveState              0.00181894626272333       0.000352
aspx.page   End SaveState                0.00224609552331372       0.000427
aspx.page   Begin SaveStateComplete      0.00228353044870228       0.000037
aspx.page   End SaveStateComplete        0.00231398124621984       0.000030
aspx.page   Begin Render                 0.00234303521816 32       0.000029
```

ERROR HANDLING:

Error handling is important part of an web application. Each error occurring must be caught and appropriate action must be taken to resolve the error. Web.config file in ASP.NET provides the facility to do what when an error occurs in application.

ASP.NET provides a simple and powerful way to deal with errors that occur in web applications. Every application must have error handling mechanism. For catching the handled exception we are using try catch block. For unhandled exception ASP.NET provides an error page.

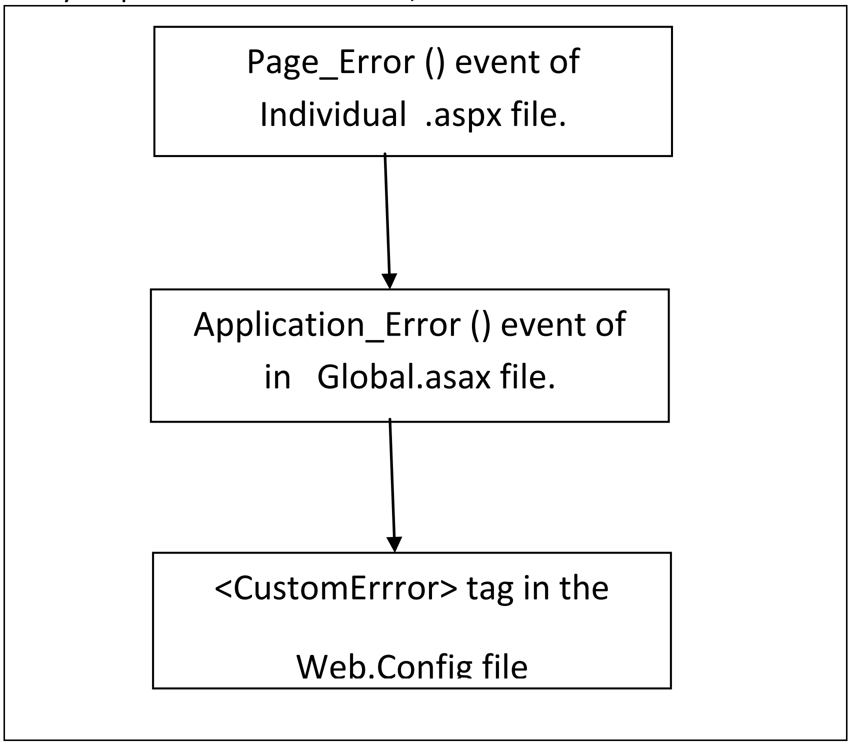
ASP.NET provides various levels at which we can handle and respond to error that might occur when we run an ASP.NET application. ASP.NET provides three main methods to trap and respond to error when they occur.

Page_Error: Occurs when an error occurs within the Web page. This event is in the Web form.
Application or Global_Error: Occurs when an error occurs within the application. This event is in the Gloabl.asax file.
Custom Errors:<Custom errors> section of the application configuration files(web.config).

Sequence of error events:

- First of all Page_Error() event of particular .aspx file (Web Form) is executed if you have written and handled it.
- After going into Page_Error() next, control goes to Application_Error() event which is written Global.asax file.
- As the last sequence, if you have written <CustomErrors> section in Web.Config , it goes to specified page.

It is very important to note that while executing these methods, it goes from Page_Error () of aspx to Application of Global .asax and finally to <CustomErrors> of Web.Config only if Error is not cleared. If in any sequence error is cleared, control is not transferred to next sequence.


Page_Error () event of Individual .aspx file.
Application_Error () event of in Global.asax file.
<CustomErrror> tag in the Web.Config file

Page Error The first line of unhandled error will go through your Page_Error handler event. Every unhandled error occured on your page will be trapped inside the Page_Error Event handler. To handle error in this event handler is quite simple. You can use built in ASP.NET Server Objects to get the LastError and then it is up to you how to handle the code.

```
protected void Page_Error(object sender, EventArgs e)
{
        Exception objErr = Server.GetLastError();
        string err = "Error in: " + Request.Url.ToString() +
             ". Error Message:" + objErr.Message.ToString();
        Response.Write("BCA6");
        System.Diagnostics.EventLog.WriteEntry("MyError", err);
        Server.ClearError();
}
```

Notice on the line we called *Server.ClearError().* This is very important if you do not want the error handler being routed or handled on the Global.asax errror handler or web.config error handler. The Page_Error event handler only covers the error that occured on your page only and not on the page that do not have the Page_Error event handler. If you want to trap the error on the whole website with just single place of code, then you might consider using Global.asax or web.config files.

Application or Global Error:

Global.asax is optional file. Your site can work without it, but it could be very useful. You can have only one Global.asax in the root folder of your web site. Global.asax contains Application_Error procedure which executes whenever some unhandled error occurs. By using Application_Error, you can **catch all unhandled errors produced by your web site**, and then write a code to save error messages to database, text file or Windows EventLog, send notificationn e-mail, write some message to user, redirect user to other page with Response.Redirect etc. If you used Server.ClearError() in Page_Error procedure, Application_Error in Global.asax will not execute. Implementation code is similar to Page_Error code above..

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

}
```

Custom Errors:

Structured exception handling is a fundamental part of the CLR and provides .Net programmers a great way of managing errors. In addition to CLR exception system, ASP.Net also provides ways of handling errors. When a runtime or design-time error occurs in an application, ASP.Net shows a default error page that gives a brief description of the error along with the line number on which the error occurred. A developer would wish to view this default error page, during the testing of the application since the description helps him in rectifying the error. But he would never want a user trying to access his application, to view this error page. The user would be least bothered to know about the error. Instead of showing the default error page, it would be more sensible to show a customized error page that would let the user send notification of the error to the administrator.
Web.config file contains a **customErrors section** inside **<System.web>**. By default, this section looks like this:

```
<customErrors mode="RemoteOnly" />
```

As you see, there is mode parameter inside customErrors tag which value is "RemoteOnly". This means that detailed messages will be shown only if site is accessed through a http://localhost.
<customErrors> section is responsible for handling the error on the basis of the setting made in that section if we do not call Server.ClearError, trap the error in the Page_Error or Application_Error event handler. In the <customErrors> section we can specify a redirect page as a default error page (defaultRedirect) or specify to a particular page based on the HTTP error code that is raised. We can use this method to customize the error message that the user receives.
If an error occurs that is not trapped at any of the previous levels in the application (i.e. either in page level or in application level), this custom page is displayed. In the above code listing the <customErrors> section includes a mode attribute that is set to On. The mode attribute is used to control how the error redirection occurs. For example, if anyone is developing the application, he or she most likely wants to see the actual ASP.NET error messages and does not want to be redirected to the more user-friendly error page. The mode attribute includes the following settings or options. On: This option specifies that custom errors are enabled. If no defaultRedirect is specified, users see a generic error. Off: This option specifies that custom errors are disabled. This allows the display of detailed errors. RemoteOnly: This option specifies that custom errors are shown only to remote clients and ASP.NET errors are shown to the local host. This is the default.
Note that defaultRedirect optional attribute specifies the default URL to direct a browser if an error occurs. When defaultRedirect is not specified, a generic error is displayed instead.

Handling specific errors:

The customErrors section of the web.config file allows us to handle individual errors differently. Errors are distinguished by their HTTP error codes; we can also specify a particular page to redirect to that is based on the HTTP error code that is raised.
The followings are some of the most common errors.

400: Bad Request - This means the request could not be understood by the server. The request was denied due to a syntax error in the request.

401: Unauthorized - This means the request requires authentication.

402: Payment required - This means the data is not accessible at the time. The owner of the space has not yet paid his or her service provider or the server was unable to serve the data that was requested.

403: Forbidden - This means the IP address or the username/password entered were not correct and the request was denied as there was no permission to access the data.

404: Not found - This means the document requested either no longer exists or has never existed on the server.

405: Method not allowed- Means the method you are using to access the document is not allowed.
408: Request Timeout - Means the server has closed the socket due to communications between the client and server taking too long.
414: The URL requested is too long.
500: Internal server error. The server encountered an error - This is most often caused by a scripting problem, a failed database access attempt, or similar reasons.
503: Service unavailable- This means the server is overloaded or down for maintenance and was unable to process the client request.

A complete HTTP status code listing is available online. We may expand the customErrors section of the web.config file to handle specific error codes. We can achieve the expansion by adding error elements under the customErrors element. The following sample shows how we may accomplish this.

```
<customErrors mode="RemoteOnly" defaultRedirect="CustomErrorPage.aspx">
<error statusCode="400" redirect="ErrorPage400.aspx" />
<error statusCode="401" redirect="ErrorPage401.aspx" />
<error statusCode="402" redirect="ErrorPage402.aspx" />
<error statusCode="403" redirect="ErrorPage403.aspx" />
<error statusCode="404" redirect="ErrorPage404.aspx" />
<error statusCode="408" redirect="ErrorPage408.aspx" />
<error statusCode="414" redirect="ErrorPage414.aspx" />
<error statusCode="500" redirect="ErrorPage500.aspx" />
<error statusCode="503" redirect="ErrorPage503.aspx" />
</customErrors>
```

The above sample handles each specific error with the corresponding page. The statusCode attribute of the error element designates the error and the redirect attribute defines the page to use. This allows us to customize the error message based upon the error encountered. The default error page handles any error that is not specified.

AUTHENTICATION & AUTHORIZATION:

Authentication and Authorization both are very important aspect of ASP.NET which allows our web site to work properly with proper user. Both of these form the basis of security in ASP.NET web application.

Also it is important to note that only ASP.NET does not work behind security but both authentication and authorization works for this. Any request which comes it is first authenticated and then it authorized by IIS.

Authentication is a process by which user's identity is checked. It is used to know based on user's credentials. Whenever a user logs on to any application, the user is first authenticated and then authorized. Web.config file contains the configuration settings for any ASP.NET application.

Authorization is a process by which user have right to access the resources. First user authenticates then authorization is given. It is used check whether the user is authenticated use a particular web page or not.

Types of Authentication:

Whenever any user visits any page of websites, the user is first authenticated then authorized. This is done by using web.config. Web.config file contains all the configuration setting for any ASP.NET application. In Web.Config file we can set authentication in 4 different ways. They can be stated as:

1. None
2. Forms Authentication

3. Windows Authentication
4. Passport Authentication

Authentication configuration is set by using <authentication> tag inside <configuration> <syste.web> element of the Web.Config file. It can be given as follows:

```
<configuration>
    <system.web>
      <authentication mode="None | Forms|Windows | PassPort">
      </authentication>
    </system.web>
</configuration>
```

Different types of authentication can be given as follows:

1. None Authentication:

This type of authentication is used when we do not want to perform any of authentication process. If authentication mode is specified as none, then ASP.NET will not go for any of security check for our web application. Generally this approach is used by web developers.

None authentication can be given as follows:

```
<configuration>
    <system.web>
      <authentication mode="None">
      </authentication>
    </system.web>
</configuration>
```

Here we do not need to specify the authorization element as authentication mode is set to none.

2. Form Authentication:

This authentication is used while developing web application. In which of authentication we would be redirected to Login Page, even when we requested any other page.

First we should authenticated using Login page, then only we would be forwarded to request web page or specified web page. Until we are authenticated correctly, we would not be able to access any other page.

Form authentication can be given as follows:

```
<configuration>
    <system.web>
      <authentication mode="Forms">
         <forms name="LoginFrm" LoginUrl="login.aspx">
            <credentials passwordFormat="Clear">
               <user name="jjkcc" password="bca"/>
               <user name="bca" password="test"/>
            </credentials>
         </forms>
      </authentication>
      <authorization>
            <allow user="jjkcc,bca"/>
            <deny usr="?"/>
      </authorization>
    </system.web>
</configuration>
```

The above describes the form authentication method. In this we would be redirected to login page URL even though we have specified some other page. Here the login URL is set to login.aspx. So if we request any page it would be redirected to login.aspx page. Along with this, we can also specify<credentials> which has list of <user> elements with their passwords.

In <authorization> element, we can specify two sub elements called allow and deny. <allow user="jjkcc,bca"/> sub element allows two user named jjkcc and bca <deny user="?"/> indicates other user are not allowed. It has two values? And *. ? denotes all other user and * denotes all the user.

3. Windows Authentication:

Windows is default authentication mode. If we set the Authentication mode to windows, then the user is authenticated based on his or her password.

While using windows authentication we do not need to specify the credentials as ASP.NET takes local users and groups for authentication.

Here is code example of windows authentication

```
<configuration>
    <system.web>
      <authentication mode="Windows">
      </authentication>
      <authorization>
         <allows  user="*"/>
      </authorization>
    </system.web>
</configuration>
```

The above code check for windows based authentication. Here all the users are authorized to use web application as we have specified * in the allow tag.

4. Passport Authentication:

Passport authentication is concerned with Microsoft service authentication. This method uses centralized authentication service provided directly by Microsoft's passport service developed by Microsoft.

User logins on the application by using his or her credential details which are then check by Microsoft's website. If the user is authenticated then he or she would get a token. If he or she does not get a token that means the user is not authenticated by Microsoft service.

Types of Authorization:

Authorization is used to determine whether a user should be granted access to specific resources. In ASP.NET, there are two ways authorize a user's access to given resource:

1. **File Authorization:**

It is performed by using AuthorizationModule. It checks the access control, list of the given file and then determine whether a user have an access to the given file or not. Access control list can be maintained for small number of files, but when there are large numbers of files then it become quite difficult to maintain. There we can use URL authorization.

2. **URL Authorization:**

It is performed by using UrlAuthorizationModule. In this module user and roles are mapped specific URLs in ASP.NET application. This module can be used to allow or deny access to various parts of an application for specific users or roles.

3. **Customized Authorization:**

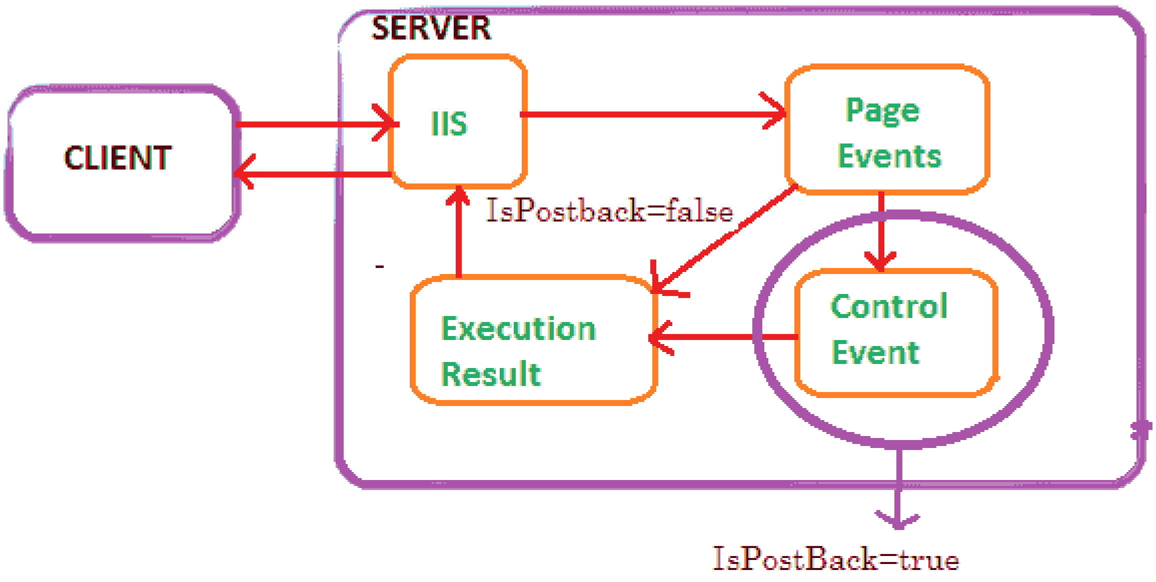
We can customize authorization by using this type. This can be done by using the location element inside the configuration element of Web.Config file.

Here is example of this type of authorization:

```
<configuration>
    <location path="Admin.aspx">
    </location>
    <system.web>
          <authorization>
             <allows  user="jjkcc"/>
             <deny user="*"/>
          </authorization>
    </system.web>
</configuration>
```

<u>What is IsPostback property?</u>

Postback is actually sending all the information from client to web server, then web server process all those contents and returns back to the client. Most of the time ASP control will cause a post back (e. g. buttonclick) but some don't unless you tell them to do In certain events (Listbox Index Changed,RadioButton Checked etc..) in an ASP.NET page upon which a PostBack might be needed.



IsPostBack is a property of the Asp.Net page that tells whether or not the page is on its initial load or if a user has perform a button on your web page that has caused the page to post back to itself. The value of the Page.IsPostBack property will be set to true when the page is executing after a postback, and false otherwise. We can check the value of this property based on the value and we can populate the controls on the page.

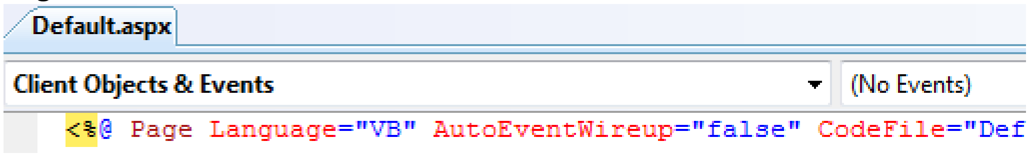
Is Postback is normally used on page _load event to detect if the web page is getting generated due to postback requested by a control on the page or if the page is getting loaded for the first time.

<u>Page Directives in Asp.Net</u>

Directives

The directives are instructions that specify optional settings in Asp.Net, but they are not rendered as part of the HTML page return to the client browser. These instructions include registering a custom control, page language etc. It describes how the .aspx pages (web forms) or .ascx pages (user controls) are processed by the .Net framework.

Page directive



```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Def
```

The most commonly used directive is the @ Page directive and it can be used only in Web Forms. Page directive allows you to specify many configuration options for the page. By default, Visual Studio creates a page directive as shown below:

You can include only one @ Page directive in your .aspx file. Also you should specify one language in the Language attribute. This can be any .NET Framework-supported language, including VB.Net, C#, or JScript.

AutoEventWireup controlled the automatic binding of page events based on the method naming convention. The default is true, which performs the automatic lookup and binding. If it is set to False then you should create methods with any name and bind them to page events explicitly

CodeFile specifies a path to the referenced code-behind file for the page. Inherits defines the name of the class from which to inherit. This can be any class derived from the Page class

Different types of directives in Asp.Net

@ Assembly

The @Assembly Directive attaches assemblies to the page or an ASP.NET user control thereby all the assembly classes and interfaces are available to the class. This directive supports the two attributes Name and src. The Name attribute defines the assembly name and the src attribute defines the source of the assembly.

@ Control

Defines control-specific attributes used by the ASP.NET page parser and compiler and can be included only in .ascx files (user controls).

@ Implements

The @Implements Directive gets the ASP.NET pages to implement .Net framework interfaces. This directive only supports a single attribute interface.

@ Import

The Import directive imports a namespace into a web page, user control page of application. If the Import directive is specified in the global.asax file, then it is applied to the entire application. If it is in a page of user control page, then it is applied to that page or control.

@ Master

Identifies a page as a master page and defines attributes used by the ASP.NET page parser and compiler and can be included only in .master files.

@ MasterType

Defines the class or virtual path used to type the Master property of a page.

@ OutputCache

The OutputCache directive controls the output caching policies of a web page or a user control.

@ Page

The @Page directive enables you to specify attributes and values for an Asp.Net Page to be used when the page is parsed and compiled. Every .aspx files should include this @Page directive to execute. There are many attributes belong to this directive.

@ PreviousPageType

Creates a strongly typed reference to the source page from the target of a cross-page posting.

@ Reference

Links a page, user control, or COM control to the current page or user control declaratively.

@ Register

Associates aliases with namespaces and classes, which allow user controls and custom server controls to be rendered when included in a requested page or user control.