ASP.NET Web Forms

ASP.NET is an Open-Source Web Technology from Microsoft for building Web Applications using .NET Languages.

Applications are divided into different categories like:

1. Desktop Applications
2. Web Applications
3. Mobile Applications

Desktop Applications means, these applications must be installed on our computer first to use them. Example: MS Office, Skype Messenger, Zoom, Browsers, etc.

Web Applications means, we install these applications first on a centralized machine known as Web Server and then provide access to clients, so that clients can connect to the Web Server using a browser thru internet and then access the application. Example: Facebook, Gmail, Amazon, Flipkart, etc.

Mobile Applications are also like Desktop Applications only i.e., we need to install them on our Mobile to consume but works with the help of Internet like a Web Application. Example: WhatsApp, Swiggy, Zomato, Uber, Ola, etc.

Desktop Applications Vs Web Applications:

- Web Applications need to be installed only once that to on 1 Computer only whereas Desktop App's are to be installed separately on each computer.
- When we need to update a Web App's it needs to be done only on the single computer where it is installed, whereas in case of Desktop App's it needs to be done on every computer.
- Desktop App's are confined to a particular location and they have usability constraints, whereas Web App's are convenient for the users to access them from any location using internet.
- Web Application's relies significantly on internet connectivity and speed, so absence of internet or poor connectivity can cause performance issues whereas Desktop App's are standalone in nature and hence do not face any hindrances resulting from Internet connectivity.
- Web Application is completely internet dependent, so they consume more bandwidth whereas Desktop App's are partially internet dependent, so they consume less bandwidth.
- To build Desktop Application with .NET Languages we are provided with technologies like Windows Forms Applications, WPF (Windows Presentation Foundation) and same as that to build Web Applications we are provided with a technology called ASP.NET.

.NET is Platform Independent i.e., applications that are developed by using .NET can run on multiple Platform's and to run .NET Applications on a machine that machine should be 1st installed with a software i.e., .NET Runtime. Microsoft provided us 3 different .NET Runtimes which are evolved over a period, like:

- .NET Framework Runtime
- .NET Core Runtime
- .NET Runtime

.NET Framework Runtime was launched into the market by Microsoft in the year 2002 and this is available only for Windows Platforms. The first version of this Runtime is 1.0 and the last version is 4.8.

.NET Core Runtime was launched into the market by Microsoft in the year 2016 and this is available for multiple platforms like Windows, Linux, and Mac. The first version of this Runtime is 1.0 and the last version is 3.1.

.NET Runtime is also same as .NET Core Runtime which was launched into the market by Microsoft in the year 2020 and this is also available for multiple platforms like Windows, Linux, and Mac. This is a combination of .NET Framework and .NET Core and evolved as "One .NET". The first version of .NET Runtime started with 5.0 and we call this as .NET 5 and the latest is .NET 6.

In .NET Framework we have been provided with ASP.NET Technology for building Web Applications and under this we have different options again, those are:

- ASP.NET Web Forms
- ASP.NET MVC
- ASP.NET Web API

In .NET CORE and .NET 5, .NET 6 we have been provided with ASP.NET Core Technology for building Web Applications and under this also we have different options again, those are:

- ASP.NET Core Web App (Razor Pages)
- ASP.NET Core Web App (Model-View-Controller)
- ASP.NET Core Web API

What is Internet?

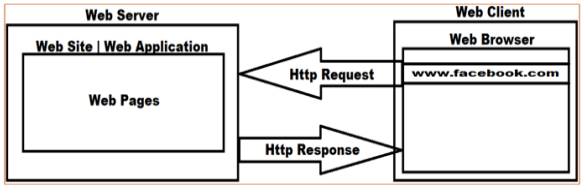
Internet is a global system of interconnected computer networks that use the standard Internet protocol suite (TCP/IP) to link several billion devices worldwide. It is a *network of networks* that consists of millions of private, public, academic, business, and government networks, of local to global scope, that are linked by a broad array of electronic, wireless, and optical networking technologies. The Internet carries an extensive range of information, resources, and services, such as the inter-linked hypertext documents and applications of the World Wide Web (WWW), the infrastructure to support email, and peer-to-peer networks for file sharing and telephony.

What is World Wide Web (WWW)?

- The Web is a network of computers all over the world.
- All the computers in the Web can communicate with each other.
- All the computers use a communication protocol called Http(s).

How does WWW work?

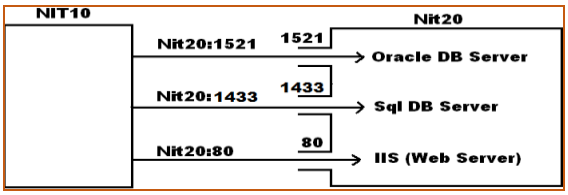
- Web information is stored in files called web pages.
- Web pages are files stored on computers called web servers.
- Devices reading the web pages are called web clients.
- Web clients view the pages with a program called as web browser.
- Popular browsers are Internet Explorer, Microsoft Edge, Mozilla Firefox, Google Chrome, and Opera.



How does a Browser Fetch a Web Page?

- A browser fetches a page from a web server by a request and this request is a standard "HTTP Request" containing a page address known as URL.
- An address or URL may look like this: **http(s)://www.google.com:80/default.html**

<Protocol>://<Domain Name>:<Port>/<Request Page>



How does a Browser Display a Web Page?

- All web pages contain instructions for display and the browser displays the page by reading those instructions.
- The most common display instructions are called HTML.

What is a Web Server?

- The collection of web pages is called as a web site or web application.
- To let others, view your web pages, you must publish your web site.
- To publish your site, you must copy it to a web server.
- Your own PC can act as a web server if it is connected to a network, but the most common practice is to use a Hosting Internet Service Provider (ISP).

What is an Internet Service Provider?

- An ISP provides Internet Services.
- A common Internet service is web hosting.
- Web hosting means storing your web site on a public server.
- Web hosting normally includes email services also.
- Web hosting often includes domain name registration.

Summary:

- If you want other people to view your web site, you must copy your site to a public server.
- Even if you can use your own PC as a web server, it is more common to let an Internet Service Provider (ISP) host your site.
- Included in a web hosting solution you can expect to find domain name registration and standard email services.
- Hosting your web site on your own server is always an option. Here are some points to consider:

Hardware Expenses: To run a "real" web site, you will have to buy some powerful server hardware. Don't expect that a low-cost PC will do the job. You will also need a permanent (24 hours a day) high-speed connection.
Software Expenses: Remember that server-licenses often are higher than client-licenses. Also note that server-licenses might have limit on number of users.
Labour Expenses: Don't expect low labour expenses. You must install your own hardware and software. You also must deal with bugs and viruses and keep your server constantly running.

Using an Internet Service Provider: renting a Web Server from an ISP is a very common option and most small companies store their web site on a server provided by an ISP only. Here are some advantages:

- **Connection Speed:** Most ISPs have very fast connections to the Internet.
- **Security and Stability:** ISPs are specialists on web hosting. Expect their servers to have more than 99% up time, the latest software patches, and the best virus protection.
- **Powerful Hardware:** ISPs often have powerful web servers that can be shared by several companies. You can also expect them to have an effective load balancing, and necessary backup servers.

Things to Consider with an ISP:

- **24-hour support:** Make sure your ISP offers 24-hours support. Don't put yourself in a situation where you cannot fix critical problems without having to wait until the next working day. Toll-free phone could be vital if you don't want to pay for long distance calls.
- **Daily Backup:** Make sure your ISP runs a daily backup routine; otherwise, you may lose valuable data.
- **Traffic Volume:** Study the ISP's traffic volume restrictions. Make sure that you don't have to pay a fortune for unexpected high traffic if your web site becomes popular.
- **Content Restrictions:** Study the ISP's content restrictions. If you plan to publish pictures or broadcast video or sound, make sure that you can.
- **E-mail Capabilities:** Make sure your ISP supports the e-mail capabilities you need.
- **Database Access:** If you plan to use data from databases on your web site, make sure your ISP supports the database access you need.

What is TCP/IP?

TCP/IP stands for Transmission Control Protocol / Internet Protocol, is a family of protocols for communication between computers. It defines how electronic devices should be connected over the network, and how data should be transmitted between them.

- **TCP** - Transmission Control Protocol is responsible for breaking down data into small packets before they can be sent over a network, and for assembling the packets again when they arrive.
- **IP** - Internet Protocol takes care of the communication between computers. It is responsible for addressing, sending, and receiving the data packets over the Internet.

TCP/IP Protocols for the Web: web browsers and servers use TCP/IP protocols to connect to the Internet. Common TCP/IP protocols are:

- **HTTP** - Hyper Text Transfer Protocol takes care of the communication between a web server and a web browser. HTTP is used for sending requests from a web client (browser) to a web server, returning web content (web pages) from the server back to the client.
- **HTTPS** - Secure HTTP, takes care of secure communication between a web server and a web browser. HTTPS typically handles credit card transactions and other sensitive data.
- **FTP** - File Transfer Protocol, takes care of transmission of files between computers.

IP is Connection-Less:

- IP is a “connection-less” communication protocol.
- IP does not occupy the communication line between two computers. This reduces the need for network lines. Each line can be used for communication between many different computers at the same time.
- With IP, messages are broken up into small independent “packets” and sent between computers via the Internet. IP is responsible for “routing” each packet to the correct destination.

IP Routers:

- When an IP packet is sent from a computer, it arrives at an IP Router.
- The IP router is responsible for “routing” the packet to the correct destination, directly or via another router.
- Communicating via IP is like sending a long letter as many small postcards, each finding its own way to the receiver.

IP Addresses:

- This is a unique identification for every device in the network.

- IP uses 32 bits or four bytes where each number should be ranging between 0 to 255; to address a computer.
- IP addresses are normally written as four numbers separated by a period, like this: 192.168.1.50.
- Each device must have a unique IP address before it can connect to the Internet.
- Each IP packet must have an address before it can be sent to another computer.
- This is an IP address: 192.68.20.50. This might be the same address: www.w3schools.com

Domain Names:

- A name is much easier to remember than a 12-digit number.
- Names used for IP addresses are called domain names; for example "w3schools.com" is a domain name.
- When you address a web site, like "www.w3schools.com", the name is translated to a number by a Domain Name Server (DNS).
- All over the world, DNS servers are connected to the Internet. DNS servers are responsible for translating domain names into IP Addresses.
- When a new domain name is registered together with an IP Address, DNS servers all over the world are updated with this information.

What is a Domain Name?

- A domain name is a unique name for a web site, like google.com, facebook.com.
- Choosing a hosting solution should include domain name registration.
- Domain names must be registered. When domain names are registered, they are added to a large domain name register. In addition, information about the web site, including the IP Address, is stored on a DNS Server.
- DNS stands for Domain Name System. A DNS server is responsible for informing all other computers on the Internet about the domain name and the web site address.

Registering a Domain

- Domains can be registered from domain name registration companies.
- These companies provide an interface to search for available domain names, and they offer a variety of domain name extensions that can be registered at the same time.

Choosing a Domain Name

- Choosing a domain name is a major step for any individual or organization.
- New domain name extensions and creative thinking offer thousands of excellent domain names!
- When choosing a name, it is important to consider the purpose of a domain name, which is to provide an easy way to reach your web site.
- The best domains have the following characteristics: Short, Meaningful, Clear and Exposure

Short - People don't like to type! A short domain name is easier to type, read, and remember.

Meaningful - A short domain is nothing without meaning, "34i4nh.com" is not easy to enter or to remember. Select a domain that relates to your site in a way that people will understand about you or your site.

Clear - Clarity is important when selecting a domain name. Avoid a name that is difficult to spell or pronounce.

Exposure - Names that are short and easy to remember are an asset. In addition to visitors, also consider search engines. Search engines index your site and rank it for relevance against terms people search for your sites, consider including a relevant search term in your domain.

What is web hosting?

Ans: Web hosting is a service provided by companies (the web host) that sell or lease space on a server where you can store the files that make your website accessible on the internet. This typically requires that you own a domain, and these companies may help you in purchase one.

How does web hosting work?

Ans: Once you purchase a Web hosting plan, web host companies store your site on their servers and assign it a unique DNS. The DNS is the address that allows people around the world to access your website. This unique address is required for people to view your site. By purchasing a website hosting package, you're buying space on their servers which allows your website files to be accessed from anywhere.

What is shared hosting?

Ans: Shared hosting is a popular hosting option where a provider hosts multiple websites on one physical web server. Typically, most websites don't use many server resources, so shared hosting lets providers offer stable service at a low cost. Shared hosting allows you to share hosting space and costs with others, while benefitting from the speed and space you need for your small business website.

Domain Name Servers (DNS): it is the Internet's equivalent of a phone book. They maintain a directory of domain names and translate them to IP Addresses. This is necessary because, although domain names are easy for people to remember, computers or machines, access websites based on IP Addresses only.

Information from all the domain name servers across the Internet are gathered and housed at the Central Registry. Host companies and Internet Service Providers interact with the Central Registry on a regular schedule to get updated DNS information.

When you type in a web address, e.g., www.facebook.com, your Internet Service Provider views the DNS associated with the domain name, translates it into a machine friendly IP address (for example 157.240.228.35 is the IP for www.facebook.com) and directs your Internet connection to the correct website. After you register a new domain name or when you update the DNS servers on your domain name, it usually takes about 12-36 hours for the domain name servers world-wide to be updated and able to access the information.

The Internet Assigned Numbers Authority (IANA) - manages the IP address space allocations globally and delegates five Regional Internet Registries (RIRs) to allocate IP address blocks to local Internet registries (Internet Service Providers) and other entities.

A **Regional Internet Registry (RIR)** is an organization that manages the allocation and registration of Internet number resources within a particular region of the world. The Regional Internet Registry system eventually dividing the world into five RIR's:

- African Network Information Center (AFRINIC) for Africa.
- American Registry for Internet Numbers (ARIN) for the United States, Canada, several parts of the Caribbean region, and Antarctica.
- Asia-Pacific Network Information Centre (APNIC) for Asia, Australia, New Zealand, and neighboring countries.
- Latin America and Caribbean Network Information Centre (LACNIC) for Latin America and parts of the Caribbean region.
- Réseaux IP Européens Network Coordination Centre (RIPE NCC) for Europe, Russia, the Middle East, and Central Asia.

Till now you have created some Web Pages by using HTML, Java Script, and CSS, and are able to access those pages from your local machines by using their physical path or address in the browser, but how can we access those pages from remote machines within a network?

Ans: If we want to provide access to Web Pages, we have developed to remote machines we need to take the help of a Server known as "Web Server".

What is a Server?

Ans: It is software which works on 2 principles like request and response. There are so many Server software's in the industry, like DB Servers (Oracle, SQL Server, and My SQL etc.), Application Servers, Web Servers, etc.

What is the need of a Web Server?

Ans: this software is used for taking request (HTTP Request) from clients in the form of a "URL (Uniform Resource Locator)" and then sends Web Pages as response (HTTP Response).

What Web Server software's are available for us to consume?

Ans: There are so many Web Server software's that are available in the market like Apache Web Server from Apache, Nginx Web Server from NGINX, IIS Web Server from Microsoft, GWS Web Server from Google, LiteSpeed Web Server from LiteSpeed Technologies.

Working with IIS Web Server:

- IIS stands for Internet Information Services which is formerly known as Internet Information Server.
- IIS Web Server is a product of Microsoft and more compatible for our ASP.NET Applications.
- IIS is a part of Windows OS which is generally installed on our machines along with OS, and to verify whether it is installed on your machine or not, open any Browser and type in the URL => http://localhost.

Note: If IIS is not installed on our machine, we get the below error: "HTTP Error 404. The requested resource is not found."

Installing IIS on our machine if not installed: Go to Control Panel => Click on Programs and Features => In the window opened click on "Turn Windows features on or off", which opens another window called "Windows Features" => In the window opened select the CheckBox "Internet Information Services" and also select each and every Sub-Item (Checkbox's) inside of it and click on the Ok button which will install IIS on your machine.

How to access a Web Page using IIS?

Ans: When we install IIS Web Server on our Machine it will provide us an admin console for managing IIS and we call it as "IIS Manager", which can be launched by searching for "inetmgr" in the window search. Once IIS Manager is opened, in LHS of the window we find "Connections Panel" and in that Panel we find our Computer Name as Server Name and when we expand it, we find a node called as "Sites" and under that we find a Website with the name "Default Web Site".

What is a Web Site?

Ans: Web Site is a collection of Web Pages, and a Web Server is a collection of Web Sites, i.e., a Web Server can contain 1 or more Web Sites under it and by default when IIS is installed on a machine there will be 1 Website already created, with the name as "Default Web Site".

To access Web Pages thru IIS Web Server do the following:

Step 1: Create a folder on your PC in any drive naming it as "ASP".

Step 2: Create an HTML Page with the name "Login.html" with the below code and save it into "ASP" folder.

```
<!DOCTYPE html>
<html>
 <head>
  <title>Login Form</title>
 </head>
 <body>
  <table align="center">
   <caption>Login Form</caption>
   <tr>
    <td>User Name:</td>
    <td><input type="text" id="txtName" name="txtName" /></td>
   </tr>
   <tr>
    <td>Password:</td>
    <td><input type="password" id="txtPwd" name="txtPwd" /></td>
   </tr>
   <tr>
    <td align="center" colspan=2>
     <input type="submit" value="Login" id="btnLogin" name="btnLogin" />
     <input type="reset" />
    </td>
   </tr>
  </table>
 </body>
</html>
```

If we want to access Web Pages thru Web Server, we are provided with 3 different options:

Option 1: Accessing thru "Default Web Site": because this Website is already created under IIS Web Server we can access our Web Pages thru the Site and to do that we need to copy our Web Pages to a folder that is linked with this Website i.e., "<OS Drive>:\inetpub\wwwroot" and if we copy our Web Pages into this folder we can access them thru IIS using their "Virtual Path" either from a local or a remote machine also. To test this let's copy our "Login.html" file into this folder and then access it thru the Virtual Path of the file as following:

Local Machine => http://localhost/Login.html or http://Server/Login.html

Remote Machine => http://Server/Login.html

Note: for every Web Site that is created on IIS, there will be 1 associated folder on the Hard Disk and all the Web Pages of that Site should be placed into that folder, right now "Default Web Site" is mapped with the "wwwroot" folder and that is the reason why we placed our "Login.html" Page into that folder. To view the mapping folder of "Default Web Site", right click on the "Default Web Site" in "IIS Manager" Window => select "Manage Website" and under that select "Advanced Settings" which opens a window and in that we find "Physical Path" property and beside that we find the folder that is mapped to this Web Site.

Option 2: Accessing them thru an Application or Virtual Directory created under Default Web Site: in this case without copying all the Web Pages into "inetpub/wwwroot" folder as we did in the previous case, we can create an

"Application or Virtual Directory" under "Default Web Site" and map them to our physical folder where we have saved our Web Pages i.e., "ASP" Folder.

Note: An Application or Virtual Directory is just a Sub-Item under the Default Web Site mapped with a physical folder.

To create an Application or Virtual Directory, Right Click on "Default Web Site" in "IIS Manager", select the option "Add Application" or "Add Virtual Directory" which opens a Window and in that, in "Alias" TextBox enter a name of your choice and under "Physical Path" TextBox enter the physical path of your folder i.e., "<drive>:\ASP".

Now following the above process, create 1 Application with the name "Site1" and create 1 Virtual Directory with the name "Site2" and map both to our physical folder i.e., "ASP". From now we can access the page of this folder in any of the following ways:

Local Machine: http://localhost/Site1/Login.html Or http://Server/Site1/Login.html
Local Machine: http://localhost/Site2/Login.html Or http://Server/Site2/Login.html

Remote Machine: http://localhost/Site1/Login.html Or http://Server/Site1/Login.html
Remote Machine: http://localhost/Site2/Login.html Or http://Server/Site2/Login.html

Option 3: Accessing the pages by creating a Site. In this case we create a new site under IIS just like the existing site i.e., (Default Web Site) and map it to our physical folder, but the difference is; in this case we can give our own Host Name or Domain Name just like "localhost" is the Host Name or Domain Name for "Default Web Site".

To create a new Site on IIS Web Server, right click on "Sites" node in "Connections" panel and select the option "Add Website", which opens a new window and in that window under "Site name" TextBox specify a name to the site for example: "MySite" and under "Physical path" TextBox specify location of our folder i.e., "<drive>:\ASP", now under "Host Name" TextBox specify a Host Name or Domain Name and click on the "Ok" button, which will create a new Site under Sites.

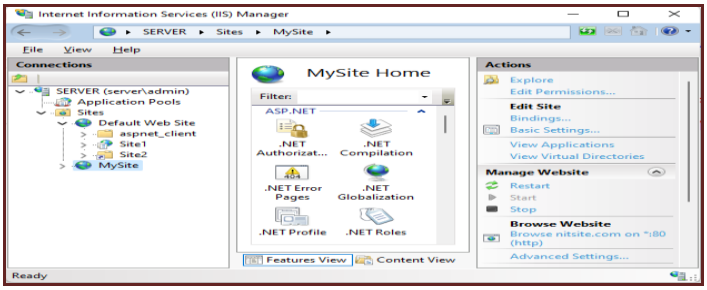
Now to use the site we need to specify that name under a file "hosts.txt" and the file is located at: "C:\Windows\System32\drivers\etc". Open the file and in the last line write the below code:

127.0.0.1 nitsite.com

Now we can access the Web Pages in our folder using the following URL:

Local Machine => http://nitsite.com/Login.html
Remote Machine => http://nitsite.com/Login.html

Note: till now we have created 1 Application and 1 Virtual Directory under "Default Web Site" and alos 1 Site under the Server with the name "MySite", so all these 3 will be present under "IIS Manager" as below:



Creating Dynamic Web Pages: These are pages that are created based on a client's request, and the content of these pages will change based on the request i.e., the output of the page will change time to time. For example, a transaction report in a Bank's Web Site.

Staring Date: 01/04/2021
Ending Date: 31/03/2022
Transaction Value:* >= 15000
Transaction Value:* <= 30000 *Optional

How to create a dynamic Web Page?

Ans: To create a dynamic Web Page along with HTML, Client-Side Scripting Languages (Java Script) and CSS we also require taking the help of some Server-Side Technologies. There are so many Server-Side Technologies that are available in the industry for creating dynamic Web Pages like: ColdFusion, Perl, PHP, Classical ASP, JSP, Servlets, ASP.NET, ASP.NET Core, DJango, etc.

1. HTML => Static Web Pages
2. HTML + CSS => Static Web Pages (Beauty)
3. HTML + CSS + Java Script => Static Web Pages (Interactive)
4. HTML + CSS + Java Script + Server Side Technologies => Dynamic Web Pages

Every Server-Side Technology uses some programming Language for implementing the logic, for example:

- ColdFusion => CFML
- Perl => Perl Script
- PHP => PHP Script
- Classical ASP => VB Script or JScript
- JSP and Servlets => Java
- ASP.NET & ASP.NET Core => .NET Languages
- DJango => Python

ASP.NET is again divided into 2 parts like:

1. ASP.Net Web Forms
2. ASP.Net MVC

ASP.NET Web Forms:

- It is an Open-Source Server-Side Web Application Framework designed by Microsoft for web development to produce dynamic Web Pages.
- It's designed to allow programmers to build Web Sites/Web Applications and Web Services.
- It was 1st released in Jan 5, 2002, with 1.0 version of .NET Framework and this is a successor to Microsoft's Classical ASP technology.
- The current version of ASP.NET Web Forms is 4.8 released on October 2019 which will be the last version and it is succeeded by ASP.NET Core.
- ASP.NET is built on Common Language Runtime (CLR), allowing programmers to write ASP.NET code by using any of the supported .NET Languages and while coding an ASP.NET Application, we will have access to all Libraries of ".NET Framework" which enable us to develop Web Applications and then benefit from CLR, Type Safety, Inheritance, and all Object Oriented Programming features.
- ASP.NET is a unified Web Development Framework which includes all the services that are necessary for us to build an Enterprise Class Web Application's with minimum volume of coding.

How to create Dynamic Web Pages by using ASP.NET?

Ans: To create a Dynamic Web Page by using ASP.NET we need to follow the below process:

1. Save the Page with ".aspx" extension and that Page can contain HTML, Java Script, CSS and ASPX Code.
2. Implement the logic (ASPX Code) that must be executed on the Server in any of the following ways:

```
<script language="language" runat="server">
  -Server-Side Logic (ASPX Code)
</script>
```

Or

```
<%
 -Server-Side Logic (ASPX Code)
%>
```

"Language" attribute is to specify in which language we want to implement the logic and it can either be "VB or CS", but the default is "VB". "Runat" attribute is to specify that this logic should be executed by the "Web Server".

Note: in-case we are implementing the logic by using second approach we don't require to specify "runat" attribute and the default language will be "VB".

The extension of a Web Page plays a very crucial role while creating dynamic web pages i.e., if the extension of Web Page is ".html", Web Server will not process the logic that we have implemented by ASP.NET and to test that, write the below code in a notepad, save it as "Test.html" and access it thru Web Server.

```
<!DOCTYPE html>
<html>
  <head>
    <title>First Dynamic Web Page</title>
  </head>
  <body>
    <h1 style="text-align:center;color:red;background-color:yellow">Naresh I Technologies</h1>
    Server Date: <% Response.Write(DateTime.Now.ToShortDateString()) %>
    <br />
    Server Time: <% Response.Write(DateTime.Now.ToLongTimeString()) %>
  </body>
</html>
```

Note: save the above page in our "ASP" folder and access it from browser in any of the following ways:

http://localhost/Site1/Test.html //Accessing thru Application Created
http://localhost/Site2/Test.html //Accessing thru Virtual Directory Created
http://nitsite.com/Test.html //Accessing thru Site Created

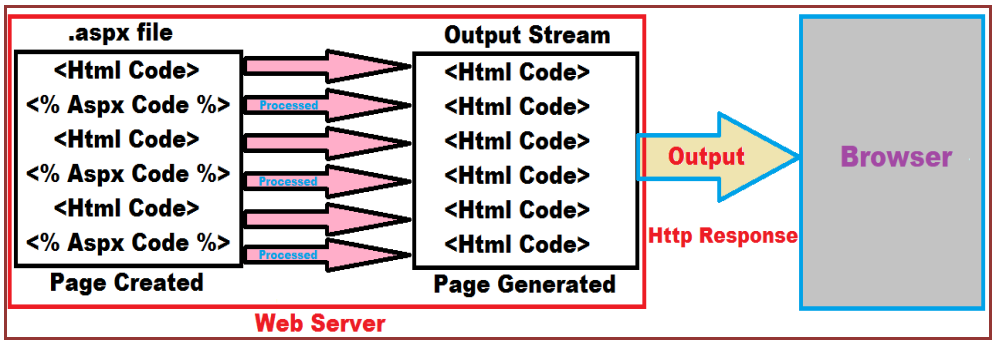
In the above case even if the Web Page contains ASPX Code in it, Web Server did not process that code because of the page extension, so it will send the content of page "as is" to the Browser for publishing. When we send a request from any Browser to Web Server for any ".html" page, Web Server will immediately send content of that page as Response to the Browser without checking what is present in the file, so the ASPX Code we implemented in the file is not processed by the server but displayed directly on the Browser.

Now use "Save As" option in notepad and save "Test.html" as "Test.aspx" in the same folder i.e., "ASP", and when we access the Page from Browser, we don't see the logic, but what we see is the result when accessed:

http://localhost/Site1/Test.aspx
http://localhost/Site2/Test.aspx
http://nitsite.com/Test.aspx

When we send request from a Web Browser to Web Server for any ".aspx" page, Web Server will open the file, reads line by line from the file and writes the "html" code "as is" into the Output Stream, whereas if it finds any "aspx code", it will process that logic and writes the results that are obtained by processing, into the Output Stream as text (html), so that after processing the whole page, content in the Output Stream will be sent to Browser as Response.

ASP.NET Renders HTML: Unfortunately, internet still has bandwidth limitations and not every person is running on the same OS, same Web Browser or same Device, and these issues make it necessary to stick with HTML as our mark-up language of choice. So, in all the Server-Side technologies including ASP.NET; Web Server will process all the logic implemented by us using the technology and converts the result into Text (HTML) which we call it as **Rendering** and then that HTML will be sent to Clients as Response.



What is the default language used in the creation of ASPX Pages?

Ans: The default language that is used in the creation of ASPX Pages is VB (VB.NET).

How to use C# as the language for creation of ASPX Pages?

Ans: If we want to use C# language for creation of ASPX Pages we need specify that by using "Page Directive" on top of the page and by using this we can specify the language we want to use for development of the page with the help of its language attribute as following:

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

To test this, write the above statement on the top of our "Test.aspx" file we have created earlier and immediately we get an error when we run the page stating that ";" is expected and to resolve the problem end the below 2 statements with ";":

```
<% Response.Write(DateTime.Now.ToShortDateString()); %>
<% Response.Write(DateTime.Now.ToLongTimeString()); %>
```

Note: a directive is an instruction that we give to the Web Server. By using Page directive we are telling the Web Server that the content in this page is implemented by using the specified language i.e., "C#" so that Web Server will use the appropriate language compiler to compile the page.

What is "Response" in our above code?

Ans: Response is an instance of a pre-defined class "HttpResponse" and we call it as an intrinsic object and apart from this Response we also have 7 other intrinsic objects like: Request, Server, Session, Application, Trace, User and Cache, which can be directly consumed in our Web Pages. Each intrinsic object is internally an instance of some class like "Request" is an instance of class "HttpRequest", "Response" is an instance of class "HttpResponse", and "Server" is an instance of class "HttpServerUtility" and so on.

How intrinsic objects are accessible to our ASPX Pages?

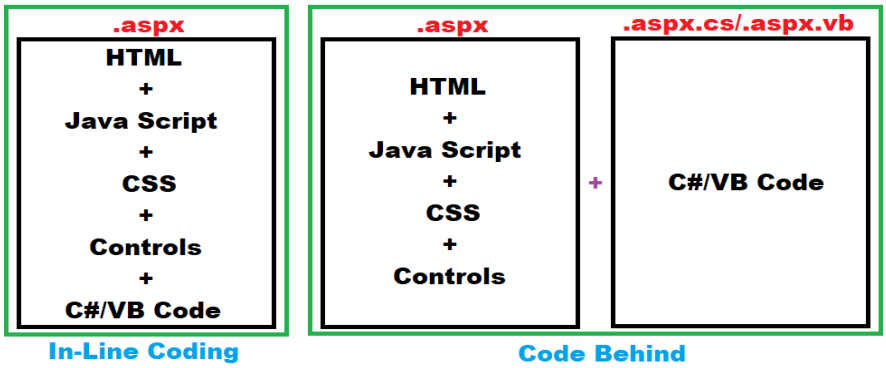
Ans: Intrinsic objects are accessible to our ASPX Pages from its parent class i.e., every ASPX Page after its compilation gets converted into a class and that class will internally inherit from a pre-defined class known as "Page", which is defined in "System.Web.UI" namespace of FCL (Framework Class Libraries) and all the above 8 intrinsic objects are inherited from that "Page" class to our ASPX Pages.

Coding Techniques in ASP.NET: ASP.NET supports 2 different coding techniques for creating Web Pages, those are:

1. Inline coding technique
2. Code behind technique

Inline Coding: in this case HTML, Java Script, CSS, Controls and Business Logic (implemented by C# or VB) will be present in 1 file which is saved with ".aspx" extension. The "Test.aspx" page we have created above is an example for "Inline Coding".

Code Behind: in this case HTML, Java Script, CSS and Controls will be present under 1 file which is saved with ".aspx" extension and Business Logic (implemented by C# or VB) will be present in a separate file which is saved with ".aspx.cs" or ".aspx.vb" based on the language in which we are implementing the Business Logic.



Developing Web Pages using Visual Studio: Open Visual Studio => select "Create a new project" option => now under the Window opened, choose "C#" under "All _Languages DropDownList", choose "Windows" under "All _Platforms DropDownList", choose "Web" under "All Project _Types" DropDownList and now in the below choose "ASP.Net Web Application (.NET Framework)" and click "Next", which launches a new window, enter Project name as "FirstWebApp", Location as "<drive>:\ASP", under Framework choose the latest Framework Version and click on the "Create" button which launches a new Window and in that choose "Empty" Project Template and in the RHS select "WebForms" CheckBox and make sure all the other Checkbox's are un-checked and click on "Create" button which will create and opens the new project.

Right now, the project which is opened is empty which doesn't contain any WebPages in it and if we open the Solution Explorer and watch, we will find 2 empty folders here with the names "App_Data" & "Models" which can either be deleted or leave them as is, and under the project we will also find 3 items with the names "Global.asax", "Packages.config" & "Web.Config" which are required under every Asp.Net Web Application.

Note: Every “ASP.Net Web Application” project is a collection of “Web Pages” and in “ASP.Net”; “Web Pages” are called as “Web Forms”.

Adding a Web Form in our project: right now our project doesn't contain any Web Form’s in it, and if we want to add a new Web Form, open Solution Explorer right click on the project and Select Add => “New Item” which opens the “Add New Item” window, choose Web Form in it, which will ask for a name, default will be “WebForm1.aspx”, change it as “First.aspx” and click on the Add button.

Note: in Visual Studio we are provided with support for “code behind” model only.

By default, the page “First.aspx” contains “Page Directive” on top of the page with few attributes like: “Language”, “AutoEventWireup”, “CodeBehind” and “Inherits”, below that we find some Html Code containing Head and Body sections.

To view the Code Behind file associated with this Web Form right click on the Web Form in document window and select “View Code” option which will launch the file “First.aspx.cs” that contain a class in it with the name “First” inheriting from the “Page” class and notice that class First is a partial class i.e., it is defined on more than one source files.

1. First.aspx.cs
2. First.aspx.designer.cs

Note: by default, “First.aspx.cs” file is only visible to us and if we want to view “First.aspx.designer.cs” file, open Solution Explorer and expand “First.aspx” node and under that we find both the files. “Designer.cs” files are used by Visual Studio to auto-generate code i.e., the code in these files will be generated by Visual Studio, so never edit the content of this files.

Writing code in a WebForm: because our page is created in Code Behind model, we can implement our logic either in “.aspx” file or “.aspx.cs” file also. For example, if we want to implement logic in “First.aspx”, then write the below code under “ ” tag which we find under the “<body>” tag:

```
<% Response.Write("Server Date: " + DateTime.Now.ToShortDateString()); %>
<br />
<% Response.Write("Server Time: " + DateTime.Now.ToLongTimeString()); %>
```

Note: to run the Web Page hit F5 and it executes the page and displays the output on browser.

If we want to implement logic in “aspx.cs” file, then add another new WebForm in the project naming it as “Second.aspx”, go to its Code View and write the below code under the method “Page_Load” which we find in the file “Second.aspx.cs”:

```
Response.Write("Server Date: " + DateTime.Now.ToShortDateString() + "<br />");
Response.Write("Server Time: " + DateTime.Now.ToLongTimeString());
```

How does a Web Application run under Visual Studio?

Ans: To run a Web Application we require a Web Server and without that Web Server we can’t run any Web Application, so Visual Studio by default provides a built-in Web Server to run the Web Application’s that we

develop under Visual Studio i.e., "IIS Express" and right now our project "FirstWebApp" is running thru that "IIS Express Web Server" only.

VS will add each Web Application to IIS Express as 1 Site by allocating 1 random Port to each Application, which will be different from Project to Project and Machine to Machine. To view the Port that is allocated for our Web Application run any WebForm in our project and watch the URL in browsers Address bar which will be as following:

http://localhost:YourPort/First.aspx
http://localhost:YourPort/Second.aspx

Note: IIS Express is provided for testing our Web Applications at the time of development and we call this as a "Development Web Server" also and once the Application development is completed, we can host our Web Application either on the "Local IIS" of our machine or "IIS" on remote machines or on some Public Server also.

How to host our Web Application into Local IIS?

Ans: as discussed above our Web Application is right now running under IIS Express of Visual Studio, and it's possible to host the Application on Local IIS of our machine and to do that first close your Visual Studio and re-open it in Administrator mode and to do that, right click on the Visual Studio and select the option "Run as Administrator". Now open Solution Explorer, right click on the project and select "Properties" option which will launch Project Property Window, in that window on the LHS we find "Web" tab, click on it and now on the right scroll down to "Servers" section, there we find a "Drop Down List" control and in that by default "IIS Express" is selected and in the Project URL TextBox it will be showing the current Project's URL i.e.,: http://localhost:Port/.

Now in the "DropDownList" select "Local IIS" option which will change the Project URL as following: http://localhost/FirstWebApp, and beside the Project URL TextBox we find a button "Create Virtual Directory" click on it, which will create an "Application" under the "Default Web Site" of "Local IIS", now click on the save button in "Visual Studio Standard ToolBar" and from now our Web Application will be running thru Local IIS and the URL when we run our Web Forms will be as following:

http://localhost/FirstWebApp/First.aspx
http://localhost/FirstWebApp/Second.aspx

Working with Web Forms: every WebForm in an ASP.NET Web Application project will be having 3 places to work with, those are:

1. Source View 2. Design View 3. Code View

To understand all these, add a new WebForm in our existing project with the name "Third.aspx".

Source View: when a new WebForm is added by default it display's the Source View only, which is nothing but the ".aspx" file and we use this for writing Html, CSS, Java Script as well as all the logic for creation of Controls.

Design View: This is a visual representation of Source View i.e. whatever we write in Source View will be displayed as output in Design View just like how output is displayed on Browser and because of this feature Visual Studio IDE is known as WYSIWYG (What You See Is What You Get) IDE. We can also use this for designing the UI's, by a feature called "Drag and Drop".

Note: To navigate from "Source View" to "Design View", in the bottom of ".aspx" file we find 3 options: Design, Split & Source. By default, it will be pointing to Source, click on Design which will take us to the Design View or click on Split to see both at a time in top and bottom.

Code View: this is nothing but ".aspx.cs" file of the Web Form and we use this for implementing all business logic using C# Language.

ASP.NET Server Controls: in ASP.NET to design the UI's we don't use Html Controls at all, and in that place we are provided with "ASP.NET Server Controls" for designing UI's. The advantages of using ASP.NET Server Controls over Html Controls are:

1. Html Controls will lose the state of their values once the Web Page is submitted to server, where as Asp.Net Server Controls will maintain the state of their values even after page submission.
2. Html Controls can't be accessed directly in C# Code whereas ASP.Net Server Controls are directly accessible in the C# Code and also, we have the advantage of intellisense listing members of that control.

We are provided with various Controls in ASP.NET for designing UI's and all those Controls are pre-defined Classes in our Libraries, which are defined under the namespace "System.Web.UI.WebControls". Asp.Net Server Controls are divided into different categories like:

1. Standard Controls
2. Data Controls
3. Validation Controls
4. Navigation Controls
5. Login Controls
6. Web Parts Controls
7. Ajax Extension Controls
8. Dynamic Data Controls

Note: we have been provided with number of controls that are implemented by using C# Language, but all those controls at the time of page processing will be rendering required Html and the logic for rendering is implemented in a method of that Control class i.e., "RenderControl" and this method will be called by ASP.NET Framework while processing the page.

To check ASP.Net Server controls renders HTML, add a new WebForm in the existing project i.e., "FirstWebApp", naming it as "Fourth.aspx" and write the below code under the " " tag we find in source view:

```
Enter Name: <input name="txtName1" type="text" id="txtName1" />
<input type="submit" name="btnSubmit1" value="Save" id="btnSubmit1" />
<br /> <br />
Enter Name: <asp:TextBox ID="txtName2" runat="server" />
<asp:Button ID="btnSubmit2" runat="server" Text="Save" />
```

Run the above page, we see 2 TextBox's and 2 Button's on the page, now right click on the Browser and select the option "View Page Source" which will display the source code of this page and notice in that code; ASP.Net Server Controls also will be showing code in Html format only because they are rendered and this is how we see the code there, under the last tag.

```
Enter Name: <input name="txtName1" type="text" id="txtName1" />
<input type="submit" name="btnSubmit1" value="Save" id="btnSubmit1"  />
<br /> <br />
```

Enter Name: <input name="txtName2" type="text" id="txtName2" />
<input type="submit" name="btnSubmit2" value="Save" id="btnSubmit2" />

Working with ASP.NET Server Controls: every ASP.NET Server Control is associated with 3 things in it:

1. Properties
2. Methods
3. Events

- Properties are nothing but attributes of a control which defines the look of a control. E.g.: Width, Height, BackColor, ForeColor, BorderColor, BorderStyle, Font, etc.

- Methods are actions performed by the control when we call them. E.g.: Focus(), Clear(), etc.

- An Event is a time which tells when an action has to be performed. E.g.: Click of a Button, Load of a Page, etc.

Note: The parent class for all ASP.NET Server Controls is class Control of "System.Web.UI" namespace, which contains all the common Properties Methods and Events of all ASP.NET Server Controls.

Placing an ASP.NET Server Control on a Web Form: to work with ASP.NET Server Controls create a new empty "ASP.NET Web Application (.NET Framework) Project, naming it as "ControlsDemo", choose Empty Project Template and select the "Web Forms" Check Box, uncheck all the other CheckBox's and click on Create button. In the project first delete the existing folders, and then add a WebForm, naming it as "ControlCreation.aspx" and delete the " " tag that is present inside the "<form>" tag. We can place a control on our Web Form either thru "Design View" or "Source View" or "Code View" also.

Design View: go to Design View of our Web Form and in the LHS we find a window called "Toolbox", open it and in that window under the "Standard Tab" we find a list of controls, either double click on the Button control or Drag and Drag the Button control on to the Design View.

Note: this action will generate all the required "ASPX Code" for creation of the Button and to view that code go to Source View of the page and there we find the below code:

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

Source View: same as the above we can also write code for creation of a control just like what VS has generated and to test that write the below code under existing "Button1" in Source View:

```
<asp:Button ID="Button2" runat="server" Text="Button" />
```

Note: Every control we place on a WebForm is an instance of an appropriate control class, so in the above case "Button1" and "Button2" are 2 instances of Button Class.

Code View: we can also explicitly create the instance of any control class thru C# code and add it on the Form and to test this process go to "aspx.cs" file of our WebForm and write the below code in "Page_Load" method which is present there:

```
Button Button3 = new Button();
Button3.ID = "Button3";
Button3.Text = "Button";
form1.Controls.Add(Button3);
```

Note: now run the WebForm and you will see all the 3 buttons on the Browser Window.
Working with properties of Controls: every ASP.Net Server control is associated with "n" no. of properties and all those properties will define the look of the control. We can set properties to a control also in 3 different ways.

Using Design View: go to Design View of the "ControlCreation.aspx" WebForm, select Button1 and hit F4 which opens the Property Window on RHS, listing all the properties of Button1 and in that window we find Control properties like BackColor, BorderColor, BorderStyle, BorderWidth, Font, ForeColor, etc, change any of the required property values and view the output directly in Design View.

Note: above action will generate all the required code in Source View as per our settings, to verify that go to source view and watch the code added to Button1.

Using Source View: same as Visual Studio generated the code for settings of Button1 in source view, we can also write manual code to perform property settings and to test that add the following code for Button2 tag, which should now look as below:

```
<asp:Button ID="Button2" runat="server" Text="Button" BackColor="Pink" BorderColor="Turquoise"
        BorderStyle="Dotted" BorderWidth="5" Font-Size="X-Large" ForeColor="SpringGreen" />
```

Using Code View: we can also set properties to controls by writing "C# Code", to test that go to "aspx.cs" file and write the below code in "Page_Load" method above the statement "form1.Controls.Add(Button3);":

```
Button3.BackColor = System.Drawing.Color.AliceBlue;
Button3.BorderColor = System.Drawing.Color.CadetBlue;
Button3.BorderStyle = BorderStyle.Dashed;
Button3.BorderWidth = Unit.Pixel(5);
Button3.Font.Size = FontUnit.XLarge;
Button3.ForeColor = System.Drawing.Color.Chocolate;
```

Note: run the web page again to watch the output.

Working with Events of Control: an event is a time period which tells when an action has to be performed i.e. when a method has to be executed. Every control is associated with a set of events and each event will fire at some point of time which is based on user interactions, so we can bind methods with these events and those methods will execute whenever the event fires.

Note: in GUI (Graphical User Interface) programming we don't call methods directly, but we bind the methods with events and those methods gets executed whenever the event fires and every event will fire at some point of time. Events are already defined under the Control classes, so we need to define methods in our page class and then bind those methods with events.

We can define methods for events in 3 different ways:

1. **Using Design View:** go to Design View of "ControlCreation.aspx", select Button1, hit F4, in the property window opened we find "Events" option on the top, select it which will show events of Button and every Event will fire at some point of time which can be identified by watching the description below. Now double click on Click Event which will generate a method in "aspx.cs" file for implementing the logic that should be executed when the End User clicks on the Button and the name of that method will be "Button1_Click" which follows a pattern i.e., "<Control Name>_<Event>", now write the below code in that method:

Response.Write("<script>alert('Button1 is clicked.')</script>");

Note: now if we go to Source View and watch we find "Button1_Click" method bound to the Button's click event as following:

<asp:Button ID="Button1" ..... OnClick="Button1_Click" />

2. Using Source View: we can generate methods for events thru source view also and to do that go to Source View and add the following code to Button2 tag in the end:

OnClick="Button2_Click"

Note: when we type "OnClick=", Intellisense will display the option "<Create New Event>" select it, which will automatically generate the above code. Now in Code View we find our method "Button2_Click" for implementing the logic, write the below code in it:

Response.Write("<script>alert('Button2 is clicked.')</script>");

3. Using Code View: we can generate methods for events in Code View also, and to do that write the following statement in "Page_Load" method above the statement "form1.Controls.Add("Button1");":

Type Button3.Click += and hit the tab key => which will change as Button3.Click += Button3_Click;

This action will generate a method with the name "Button3_Click", write the below code under that method:

Response.Write("<script>alert('Button3 is clicked.')</script>");

Note: the methods under which we are implementing the logic for an event are known as "Event Handlers" and all these methods are non-value returning and every method takes 2 parameters in common, those are:

1. Object
2. EventArgs or a child class of EventArgs

Default Events: every control is associated with multiple events with it and to generate a "Event Handler" for any of those Events we need to follow any of the 3 processes that are described above, whereas for every control there will be 1 default Event and in case we want to generate "Event Handler" to that default event without following any of the options that are described above, we can directly double click on the Control in Design View which will generate the required Event Handler.

Control	Default Event
Button, LinkButton, ImageButton	Click
TextBox	TextChanged
CheckBox and RadioButton	CheckedChanged
DropDownList, ListBox, CheckBoxList and RadioButtonList	SelectedIndexChanged
Calendar	SelectionChanged

Label Control: this control is used for displaying static text on the UI and we set the text value by using the "Text" property of the control.

Button Control: this control is designed for submitting a page to the Server and under this family we have 3 Classes (Controls): Button, LinkButton and ImageButton, and when we place any of the above 3 button on a WebForm they will render all the required Html Code and converts as following:

Button => Html Input-Type Submit
LinkButton => Html Hyperlink

ImageButton => Html Input-Type Image

Note: to use ImageButton we need to set the ImageUrl property and to do that open Solution Explorer, right click on the Project, select "Add" => "New Folder", which will add a new Folder in the project, name it as "Images". Now right click on the Images folder and select "Add" => "Existing Item" which will open a dialog box, select an image from your hard disk and it will add into the folder.

To work with "Button" controls add a new WebForm in the project naming it as "ButtonDemo1.aspx" and write the below code under " " tag:

```
<asp:Button ID="Button1" runat="server" Text="Button" /><br />
<asp:LinkButton ID="LinkButton1" runat="server" Text="LinkButton" /><br />
<asp:ImageButton ID="ImageButton1" runat="server" Width="50" Height="50" ImageUrl="~/Images/Nike.jpg" />
```

When we run the page, it will render the below code and we can view that by using "View Page Source" option of browser:

```
<input type="submit" name="Button1" value="Button" id="Button1" /><br />
<a id="LinkButton1" href="javascript:__doPostBack(&#39;LinkButton1&#39;,&#39;&#39;)">LinkButton</a><br />
<input type="image" name="ImageButton1" id="ImageButton1" src="Images/Nike.jpg"
                    style="height:50px;width:50px;" />
```

What is meant by submitting a page to Server?

Ans: submitting a page to server means transferring all the values of a page that are entered into the controls from Client's Browser to the Web Server.

Page Submission is of 2 types:

1. Post back Submission
2. Cross Page Submission

Note: Post back submission is a process of submitting a page back to itself whereas cross page submission means it is a process of submitting a page to another page.

Page1 => Submitting to Page1 //Postback Submission
Page1 => Submitting to Page2 //Crosspage Submission

By default, Button control submits a page back to itself (Post back) and it is also capable of submitting the page to other pages also (Cross Page).

Understanding "Post Back Submission" and "Cross Page Submission": add a new WebForm in our project naming it as "ButtonDemo2.aspx" and write the below code under its " " tag:

```
<asp:Button ID="B1" runat="server" Text="Post Back Submission" />
<asp:Button ID="B2" runat="server" Text="Cross Page Submission" />
```

Right now, both the Buttons will perform PostBack Submission only, and to check that, go to "aspx.cs" (Code View) file and write the below code under "Page_Load" Event Handler:

```
if (IsPostBack)
  Response.Write("This is a post or post back request.");
else
```

```
Response.Write("This is a first or get request.");
```

Note: "IsPostBack" is a property of our parent class (Page) which returns true if the current request is a Post or Post back request and false if it is a First or Get request. First request to a page is called as "Get Request" and the next requests to a page are called as "Post Request".

Now run the WebForm and watch the output which will initially display the value as "This is a first or get Request." which indicates it's a first request and when we click on any of the 2 buttons it will display the value as "This is a post or post back Request." which indicates it's a post back request.

Right now, both buttons are performing post back submission only whereas if we want the 2nd Button to perform a cross page submission then add a new WebForm in the project naming it as "ButtonDemo3.aspx" and write the following code under its "Page_Load" Event Handler in Code View:

```
Response.Write("<font color='red'>Cross Page Submission.</font>");
```

Now come to Design View of "ButtonDemo2.aspx", select 2nd Button, go to its properties, identify the "PostBackUrl" property, select it, which will display a Button beside, click on it which will launch a new screen and in that select "ButtonDemo3.aspx" and then run "ButtonDemo2.aspx" form, and when we watch the output i.e. when we click on "Button1" it will perform Post Back Submission and when we click on "Button2" it will perform Cross Page Submission and launches "ButtonDemo3.aspx".

Important members of Button Control:

1. **Text:** It is a property using which we can associate some data with the control that is visible to end users and this works as a caption of the button.
2. **PostBackUrl:** It is a property using which we can specify the Address of target page to whom the current page has to be submitted when the button is clicked.
3. **OnClientClick:** It is a property using which we can specify the name of a JavaScript function which has to be executed when the button is clicked, i.e., before submitting the page to server.
4. **CommandName & CommandArgument:** These 2 properties are used for associating some additional data with the control apart from the Text Property but these are not visible to end users and these values can be used in code under Command Event.
5. **Click:** It is an event which fires when the button is clicked.
6. **Command:** This is also an event which fires when the button is clicked and there are associated CommandName or CommandArgument values.

To understand about the "OnClientClick" property, add a new WebForm in the project naming it as "ButtonDemo4.aspx" and write the below code under its " " tag:

```
<asp:Button ID="Button1" runat="server" Text="Click Me" />
<asp:Label ID="Label1" runat="server" ForeColor="Red" />
```

In this WebForm when we click on Button it has to display a message in Label Control telling that, the page has been submitted to server. To achieve it generate a "Click Event Handler" for Button Control and write the below code in it:

```
Label1.Text = "Page is submitted to server.";
```

Now when we run the Form and click on the Button Control, Label Control will display the message. But when the user clicks on the Button first it must ask for a confirmation about submitting the page to server and if the user accepts it, then only page should be submitted to server. This can be done by using JavaScript code and to

do that, go to Source View of “ButtonDemo4.aspx” and write the following code under “<head>” section of the page:

```
<script>
 function Confirmation()
 {
  var Result = confirm("Are you sure of submitting the page to server?");
  return Result;
 }
</script>
```

Now to execute this code when Button is clicked, go to Design View of the page, select the Button, go to its properties, select “OnClientClick” property and type the name of JavaScript function we defined, as following: "return Confirmation()". Now run the WebForm again and check the output.

Understanding about Object and EventArgs parameters of Event Handlers: as discussed earlier every Event Handler will be taking 2 parameters i.e., “sender” of type “object” and “e” of type “EventArgs” or child class of “EventArgs”. The use of these parameters is we can implement logic for multiple controls in a single Event Handler.

1. **“object sender”**: to understand about this parameter, add a new WebForm in the project naming it as “Calculator1.aspx” and write the below code in its “ ” tag:

```
<table align="center">
 <caption>Calculator</caption>
 <tr>
  <td>Enter 1<sup>st</sup> number:</td>
  <td><asp:TextBox ID="txtNum1" runat="server" /></td>
 </tr>
 <tr>
  <td>Enter 2<sup>nd</sup> number:</td>
  <td><asp:TextBox ID="txtNum2" runat="server" /></td>
 </tr>
 <tr>
  <td>Result Generated:</td>
  <td><asp:TextBox ID="txtResult" runat="server" /></td>
 </tr>
 <tr>
  <td colspan="2" align="center">
   <asp:Button ID="btnAdd" runat="server" Text="Add" />
   <asp:Button ID="btnSub" runat="server" Text="Sub" />
   <asp:Button ID="btnMul" runat="server" Text="Mul" />
   <asp:Button ID="btnDiv" runat="server" Text="Div" />
   <asp:Button ID="btnMod" runat="server" Text="Mod" />
  </td>
 </tr>
</table>
```

Now let's write 1 Event Handler method for all the 5 buttons and to do that go to design view of the form, select “Add” button, open property window, select Events, select Click Event and beside it specify a name to the Event Handler “Buttons_Click” and click on it which will generate the Event Handler with the given name.

Now to bind the Event Handler with remaining 4 buttons, go to design view again, select “Sub”, “Mul”, “Div” and “Mod” buttons at a time, open property window, select Events, select Click Event and now we will find a drop down besides, click on it which will list the Event Handler “Buttons_Click” select it which will bind the “Event Handler” with all 4 buttons.

Now go to “Calculator1.aspx.cs” file and write the below code over there:

Code under Page_Load:

```
if (!IsPostBack)
  txtNum1.Focus();
```

Code under Button_Click:

```
Button b = (Button)sender;
int Num1 = int.Parse(txtNum1.Text);
int Num2 = int.Parse(txtNum2.Text);
int Result = 0;
switch(b.ID)
{
  case "btnAdd":
    Result = Num1 + Num2;
    break;
  case "btnSub":
    Result = Num1 - Num2;
    break;
  case "btnMul":
    Result = Num1 * Num2;
    break;
  case "btnDiv":
    Result = Num1 / Num2;
    break;
  case "btnMod":
    Result = Num1 % Num2;
    break;
  }
txtResult.Text = Result.ToString();
```

When an Event Handler is bound with multiple controls we can identify the control which is raising the Event (Click in our case) by using the “sender” parameter of Event Handler, because we are already aware that every Control is a class and a control that is placed on the Form is an Instance of that class, so whenever a Control Instance raises an Event immediately that instance is captured in the “sender” parameter which of type “object” (default parent class of all classes).

Because “sender” is of type “object” it is not capable of accessing any of the child class members, for example in the above case “Button” is the child class, and to overcome this problem we need to convert “sender” back into button as we have performed in our above code and then we can read it’s ID or Text or any other property and implement the logic specific to the control.

2. **“EventArgs (or child class of EventArgs) e”:** to understand about this parameter, add a new WebForm in the project naming it as “Calculator2.aspx” and write the below code in its “ ” tag:

```
<table align="center">
 <caption>Calculator</caption>
 <tr>
  <td>Enter 1<sup>st</sup> number:</td>
  <td><asp:TextBox ID="txtNum1" runat="server" /></td>
 </tr>
 <tr>
  <td>Enter 2<sup>nd</sup> number:</td>
  <td><asp:TextBox ID="txtNum2" runat="server" /></td>
 </tr>
 <tr>
  <td>Result Generated:</td>
  <td><asp:TextBox ID="txtResult" runat="server" /></td>
 </tr>
 <tr>
  <td colspan="2" align="center">
   <asp:Button ID="btnAdd" runat="server" Text="Add" CommandName="+" />
   <asp:Button ID="btnSub" runat="server" Text="Sub" CommandName="-" />
   <asp:Button ID="btnMul" runat="server" Text="Mul" CommandName="*" />
   <asp:Button ID="btnDiv" runat="server" Text="Div" CommandName="/" />
   <asp:Button ID="btnMod" runat="server" Text="Mod" CommandName="%" />
  </td>
 </tr>
</table>
```

In the above case we have set a CommandName property for each button and this value can be accessed and used in the Command Event of button. Now let’s write 1 Event Handler method for all the 5 buttons and to do that go to design view of the form, select “Add” button, open property window, select Events, select Command Event and beside it specifies a name to the Event Handler i.e. “Buttons_Command” and click on it which will generate an Event Handler with the given name.

Now to bind the Event Handler with remaining 4 buttons, go to design view again, select “Sub”, “Mul”, “Div” and “Mod” buttons at a time, open property window, select Events, select Command Event and now we will find a drop down besides, click on it which will list the Event Handler i.e. “Buttons_Command” select it which will bind the “Event Handler” with all 4 buttons. Now go to “Calculator2.aspx.cs” file and write the below code there:

Code under Page_Load:

```
if (!IsPostBack)
  txtNum1.Focus();
```

Code under Button_Command:

```
int x = int.Parse(txtNum1.Text);
int y = int.Parse(txtNum2.Text);
int z = 0;
switch(e.CommandName)
{
 case "+":
   z = x + y;
   break;
 case "-":
   z = x - y;
   break;
 case "*":
   z = x * y;
   break;
 case "/":
   z = x / y;
   break;
 case "%":
   z = x % y;
   break;
}
txtResult.Text = z.ToString();
```

In the above case we have specified a CommandName to each Button and we can access that CommandName value under the Event Handler by using "e" of type CommandEventArgs which is nothing but the child class of the EventArgs class, so once we can access the CommandName we can easily identify the Control which is raising the Event and implement the logic as we have done in the above case.

In the above pages whenever we click on a button the whole page is submitted to the server and this happens every time we click on the button and re-loads the whole page again. To avoid this problem of reloading the whole page again, use "AJAX".

AJAX: ASynchronous JavaScript and XML

- AJAX is a technique for creating fast and dynamic web pages. It is a cross platform technology which speeds up the response time.
- AJAX allows Web Pages to be updated asynchronously by exchanging small amount of data with the server behind the screen. This means it is possible to update parts of a Web Page with-out the whole page being updated.
- Ajax is not a single technology, but rather a group of technologies combined together, like:
  1. HTML (or XHTML) and CSS for presentation.
  2. The Document Object Model (DOM) for dynamic display of data and interaction with data.
  3. JSON or XML for the interchange of data, and XSLT for its manipulation.
  4. The XMLHttpRequest object for asynchronous communication.
  5. Java Script to bring all these technologies together.

Note: a variety of popular Java Script libraries like JQuery provides built-in support and assistance in executing Ajax requests. Classical Web Pages which do not use AJAX used to reload the entire page if any content has to change. ASP.NET provides us various controls for using AJAX in ASP.NET App's and those controls will add script to the page which is executed and processed by the browser without us implementing all the required logic.

In the toolbox we find a "Tab" called "AJAX Entensions" and under that we find all the AJAX controls provided by ASP.NET for us to consume.

1. ScriptManager Control: this is the most important control and must be present on the Web Page for other AJAX Controls to work which takes care of the client-side script for all the Server Side AJAX Controls.

2. ScriptManagerProxy Control: this control lets you add scripts and services to Content Pages and User Controls if the Master Page or Host Page already contains a ScriptManager control and by using this control, you can add to the script and services collections which is defined by the ScriptManager control in Master Page.

3. UpdatePanel Control: this control is a container control. It just act's as an interface for holding other controls on it and doesn't have any UI. When a control inside of it triggers a "Call back" then the UpdatePanel intervenes to initiate the post asynchronously and updates just that portion of the page.

4. Timer: this control allows you to do Call backs at certain intervals, if used together with UpdatePanels (which is the most common approach) performs timed partial updates of your page, but it can also be used for posting back the entire page as well. This control uses the interval attribute to define the number of milliseconds for firing the Tick event.

5. UpdateProgress: this control provides status information about partial-page updates in UpdatePanel controls because one of the big problems with Ajax is since it's asynchronous and performs actions in background, the browser will not show us any status whereas with fast servers and fast methods this is not a big problem, but whenever we have a method which takes up a bit of time, the user is very likely to get impatient which can be resolved with UpdateProgress.

To resolve the problem in our previous pages use ASP.NET AJAX Controls and to do that rewrite the code that is present in the " " tag of our 2 pages "Calculator1" & "Calculator2" as below:

```
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
  <ContentTemplate>
   -Put the above table creation code here
  </ContentTemplate>
</asp:UpdatePanel>
```

TextBox Control: we use this control for taking input from the users and we can use this Control in 16 different ways like Single Line Text, Multi Line Text, Password, etc. Important members of TextBox Control are:

1. **TextMode:** By using this property we can set the behavior for the control and this property can be set with any of the following values:

SingleLine [d]	Date	Month	Number
MultiLine	DateTime	Time	Range
Password	DateTimeLocal	URL	Search
Color	Week	Email	Phone

2. **ReadOnly:** This is a Boolean property with the default value as false and if set as true the control becomes non editable.
3. **Rows:** This property is to specify the no. of rows we want in case the TextMode is set as MultiLine.
4. **MaxLength:** This is to specify the maximum no. of char's that has to be accepted into the control.
5. **AutoPostBack:** This is a Boolean property with default value as false, where as if we set it as true this control gets the capability of submitting the page to server i.e. it can perform a Postback.
6. **TextChanged:** This is the default Event of the control which fires when the text property has been changed.

To work with TextChanged event of the TextBox add a new WebForm under the project naming it as "ColorDialogs.aspx" and write the below code under "<form>" tag by deleting existing " " tag over there:

```
<div id="div1" runat="server">
  <br />Change Color: <asp:TextBox ID="txtColor1" runat="server" TextMode="Color" />
  <br /><br />
</div>
<div id="div2" runat="server">
  <br />Change Color: <asp:TextBox ID="txtColor2" runat="server" TextMode="Color" />
  <br /><br />
</div>
<div id="div3" runat="server">
  <br />Change Color: <asp:TextBox ID="txtColor3" runat="server" TextMode="Color" />
  <br /><br />
</div>
<asp:Button ID="Button1" runat="server" Text="Button" />
```

Now go to design view of WebForm and double click on each TextBox to generate a "TextChanged" "EventHandler" for each TextBox and write the below code in corresponding "EventHandlers" under "aspx.cs" file:

Code under txtColor1_TextChanged:

```
div1.Attributes.Add("style", "background-color:" + txtColor1.Text);
```

Code under txtColor2_TextChanged:

```
div2.Attributes.Add("style", "background-color:" + txtColor2.Text);
```

Code under txtColor3_TextChanged:

```
div3.Attributes.Add("style", "background-color:" + txtColor3.Text);
```

Now run the WebForm and select a color under any "ColorDialog", but the background color of corresponding " " control will not change even if we implemented logic under the "TextChanged" event of TextBox, because TextBox control is not capable of Submitting the page to server and to overcome this problem, after selecting a Color in ColorDialog's we need to click on the Button we placed below which will Submit the page to server and then the logic we implemented under "TextChanged" event of TextBox's will execute.

How the logic is executing when the Button is clicked?

Ans: When a control raises an event but not capable of submitting the page to server, until the page is submitted to Server the event which is raised gets cached and executes when the page is submitted to the server next time and we call them as "Cached Events".

Cached Event: this event will store page data, which gets processed when the page is submitted to the server by a Postback.

In our previous example, if we want the logic to be executed immediately after changing the color, without waiting for the Button to click, do the following:

Step 1: Delete the Button on the form.
Step 2: Set the AutoPostBack property of all the 3 TextBox's as true so that these controls will get the capability of submitting the page to server immediately when the Text (Color) is changed.

The drawback now is, first time when we load the page, all the 3 TextBox's will be set with a default color that is "Black" and internally "TextChanged" event will fire but the page will not be performing a Postback because in the page load Postback will not be performed until the user interaction starts. So the events gets cached, and when we select a color in any TextBox, it will set the background color of corresponding div control and this action will also change the background color of other 2 div's with black (default color) and to overcome this problem we need to take the help of "AJAX". To overcome these problems in our page first add a "ScriptManager" to form and to do that write the following code just below the <form> tag:

```
<asp:ScriptManager ID="ScriptManager1" runat="server" />
```

Now place each " " tag in "UpdatePanel" which should look as following:

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Conditional">
 <ContentTemplate>
  <div id="div1" runat="server"><br />
   Change Color: <asp:TextBox ID="txtColor1" runat="server" TextMode="Color" AutoPostBack="True"
                        OnTextChanged="txtColor1_TextChanged" /><br /><br />
  </div>
 </ContentTemplate>
</asp:UpdatePanel>
<asp:UpdatePanel ID="UpdatePanel2" runat="server" UpdateMode="Conditional">
 <ContentTemplate>
  <div id="div2" runat="server"><br />
  Change Color: <asp:TextBox ID="txtColor2" runat="server" TextMode="Color" AutoPostBack="True"
                        OnTextChanged="txtColor2_TextChanged" /> <br /><br />
  </div>
 </ContentTemplate>
</asp:UpdatePanel>
<asp:UpdatePanel ID="UpdatePanel3" runat="server" UpdateMode="Conditional">
 <ContentTemplate>
  <div id="div3" runat="server"><br />
   Change Color: <asp:TextBox ID="txtColor3" runat="server" TextMode="Color" AutoPostBack="True"
                        OnTextChanged="txtColor3_TextChanged" /><br /><br />
  </div>
```

```
 </ContentTemplate>
</asp:UpdatePanel>
```

Note: When we have a single UpdatePanel it will update itself when there is a partial Postback, whereas when we have multiple Update Panels all those Update Panels will be updated even if we want 1 to be updated because the UpdateMode property of UpdatePanel has a default value "Always", so now also when we change the color in 1 ColorDialog all the 3 div's controls will change their background color and to overcome this problem we need to set UpdateMode property of UpdatePanel with the value "Conditional". Now run the WebForm again and watch the difference in output where we notice that whatever TextBox we change the color that corresponding controls background color only gets changed.

Understanding Page Submission: as learnt earlier the process of sending values that are entered by a user in controls of a page to the Web Server is known as Page Submission and it is of 2 types:

1. **Postback Submission**
2. **Cross Page Submission**

Postback Submission: To understand this in detail let's create a Login Form and to do that add a new WebForm in the project naming it as "LoginForm.aspx" and write the below code under " " tag:

```
<table align="center">
 <caption>Login Form</caption>
 <tr>
  <td>User Name:</td>
  <td><asp:TextBox ID="txtName" runat="server" /></td>
 </tr>
 <tr>
  <td>Password:</td>
  <td><asp:TextBox ID="txtPwd" runat="server" TextMode="Password" MaxLength="16" /></td>
 </tr>
 <tr>
  <td>Email Id</td>
  <td><asp:TextBox ID="txtEmail" runat="server" TextMode="Email" /></td>
 </tr>
 <tr>
  <td colspan="2" align="center">
   <asp:Button ID="btnLogin" runat="server" Text="Login" />
   <asp:Button ID="btnReset" runat="server" Text="Reset" />
  </td>
 </tr>
 <tr>
  <td colspan="2" align="center">
   <asp:Label ID="lblStatus" runat="server" ForeColor="Red" />
  </td>
 </tr>
</table>
```

Now go to design view, double click on the "Login Button" & "Reset Button" to generate required event handlers and write the below code in "LoginPage.aspx.cs" file:

Code under Page_Load:

```
if(!IsPostBack)
  txtName.Focus();
```

Code Under Reset Button Click:

```
txtName.Text = txtPwd.Text = txtEmail.Text = lblStatus.Text = ""; txtName.Focus();
```

Code under Login Button Click:

```
if (txtName.Text == "admin" && txtPwd.Text == "admin#123" && txtEmail.Text == "admin@nareshit.com")
  lblStatus.Text = "Valid User";
else
  lblStatus.Text = "Invalid User";
```

Note: in the above page we have performed a Postback Submission i.e. the page has submitted the data back to itself and was validated.

In a Postback Submission we can read values of controls directly on server by using their associated properties, for example we have read the value of TextBox by using it's Text property is our above code.

Cross Page Submission: this is a process of submitting a page to another page and to understand Cross Page Submission add 2 new WebForms in the project naming them as "Contact.aspx" and "Respond.aspx". Write the below code under " " tag of "Contact.aspx":

```
<table align="center">
  <caption>Contact Details</caption>
  <tr>
    <td>Name:</td><td><asp:TextBox ID="txtName" runat="server" /></td>
  </tr>
  <tr>
    <td>Phone No:</td><td><asp:TextBox ID="txtPhone" runat="server" MaxLength="10" /></td>
  </tr>
  <tr>
    <td>Email Id:</td><td><asp:TextBox ID="txtEmail" runat="server" TextMode="Email" /></td>
  </tr>
  <tr>
    <td colspan="2" align="center">
      <asp:Button ID="btnSubmit" runat="server" Text="Submit" PostBackUrl="~/Respond.aspx" />
      <asp:Button ID="btnReset" runat="server" Text="Reset" />
    </td>
  </tr>
</table>
```

Now go to design view, double click on the "Reset Button" to generate the event handler and write the below code in "Contact.aspx.cs" file:

Code under Page_Load:

```
if (!IsPostBack)
  txtName.Focus();
```

Code under Reset Button:

```
txtName.Text = txtPhone.Text = txtEmail.Text = ""; txtName.Focus();
```

Run “Contact.aspx”, fill in the details and click on Login button, which will send a request from the browser to “Respond.aspx” becuase for Submit Button we have set “PostBackUrl” Property with the value “Respond.aspx” and we call this as “Cross Page Submission”.

In the process of “Cross Page Submission” all the values that are entered into the Controls of “Contact.aspx” page will be submitted to “Respond.aspx” page and at this time, lot of information is transferred from the browser to server like, Session State, Application State, Request Cookie Collection, Response Cookie Collection, Header Collection, Form Collection, Query String Collection, Server Variable Collection, etc.

Form Collection will contain the information of controls and their values in a “name/value” pair combination, so in “Respond.aspx” page we can read the Form Collection values with the help of “name” using the request object and consume the values. To test that go to “Respond.aspx.cs” file and write the following code under “Page_Load” Method:

```
string Name = Request.Form["txtName"];
string Phone = Request.Form["txtPhone"];
string Email = Request.Form["txtEmail"];
Response.Write("<h3>Hello " + Name + ", we have received your contact details.</h3>");
Response.Write("Contact Phone: " + Phone + "<br />");
Response.Write("Contact Email: " + Email + "<br />");
```

Note: in the above, “Request.Form” refers to “Form Collection” and the parameter we pass to it is the “name” corresponding to the value we want to read.

When data is submitted from 1 page to another page, data transfer from source page to target page will be done in any of the 2 methods:

1. **Get Method**
2. **Post Method**

For Html Form’s the default method is “get” and if we want to change it we require to use “method” attribute on the “<form>” element and specify it as “post” as following:

```
<form id="f1" name="f1" method="post">
```

For ASPX Form’s the default method is “post” and if we want to change it we require to use “method” attribute on the “<form>” element and specify it as “get” as following:

```
<form id="f1" runat="server" method="get">
```

Currently “Contact.aspx” is using an ASPX Form and so the submit method is “post” and if we want to change it as “get”, go to “Contact.aspx” file and add “method” attribute to “<form>” tag with value as “get” and it should look now as following:

```
<form id="form1" runat="server" method="get">
```

Run “Contact.aspx” Form again, enter values into the controls and click on “Submit” button which will Submit Control values to “Respond.aspx” but the difference is when the submit method is Post, values will go to target page as “Form Collection”, whereas when the Submit Method is Get, values will go to target page as “Query

String Collection", so right now "Respond.aspx" page will not display any values that we entered in "Contact.aspx" page. To resolve the problem, go to "Respond.aspx.cs" file and change "Request.Form" as "Request.QueryString" in "Page_Load" method which should not look as below:

```
string Name = Request.QueryString["txtName"];
string Phone = Request.QueryString["txtPhone"];
string Email = Request.QueryString["txtEmail"];
```

Note: when we are not sure whether the submit method is "Get" or "Post" then we can directly read the values using Request object without specifying the Query String or Form as below:

```
string Name = Request["txtName"];
string Phone = Request["txtPhone"];
string Email = Request["txtEmail"];
```

Differences between Get and Post:

1. In case of Get the data that has to be transferred from browser to server will be concatenated to the URL of page we are requesting (target page) in the form of a "Query String" and we can view those values in the Address Bar of our browser, whereas in case of Post, data will be passed through "Http Headers" which is not at all visible to the end user.
   Note: Http Headers let the client and server pass information with an Http Request or Http Response.
2. Get is faster in transfer when compared to post whereas post is secured compared to get.
3. Get is not recommended for transporting sensitive info like password, credit card details, etc as the values are visible in address bar where as post is secured method of transporting because, data is transferred through HTTP Headers by using secure HTTP protocol, so we can make sure that data is secured.
4. In case of Get we have limitations on the data being submitted i.e., the maximum length of a URL can be 2048 characters only whereas we don't have any size limitations for data in Post.
5. In case of get we access data from "Query String Collection" whereas in case of post we access data from "Form Collection" on the target pages.

How do we specify the URL of target page for a Cross Page Submission?

Ans: we specified the URL of target page for "Cross Page Submission" using the "PostBackUrl" property of Button whereas in other web technologies we specify that by using "action" attribute of form. We can use that in ASP.Net also and to test that delete the "PostBackUrl" property associated with "Submit" Button, then go to "<form>" tag and add action attribute to it, which should now look as following:

```
<form id="form1" runat="server" method="get" action="Respond.aspx">
```

Run the page again and watch, now also when we click on "Submit" button, page will be submitted to "Respond.aspx". The drawback in this approach is not only "Submit" button of our page, "Reset" button of our page also will do "Page Submission" and to avoid this problem, without using form's "action" attribute we use button's "PostBackUrl" attribute, so that "Submit" button performs "Cross Page Submission" and "Reset" button will perform "Postback Submission".

Transferring control from 1 page to another page: we can transfer control from 1 page to another in a Web Application and that can be performed in 2 different ways:

1. **Server.Transfer**
2. **Response.Redirect**

To check transferring of the control from 1 page to another page add 3 new Web Forms in the project naming them as: LoginPage.aspx, SuccessPage.aspx and FailurePage.aspx. Now go to LoginPage.aspx, define style for body tag and set a background-color to it: <body style="background-color:lawngreen"> and then write the below code in " " tag:

```
<table align="center">
 <caption>Login Page</caption>
 <tr>
  <td>User Name:</td><td><asp:TextBox ID="txtName" runat="server" /></td>
 </tr>
 <tr>
  <td>Password:</td><td><asp:TextBox ID="txtPwd" runat="server" TextMode="Password" /></td>
 </tr>
 <tr>
  <td colspan="2" align="center">
   <asp:Button ID="btnReset" runat="server" Text="Reset" />
   <asp:Button ID="btnLogin" runat="server" Text="Login" />
  </td>
 </tr>
</table>
```

Now generate Click Event Handlers for Login & Reset Button, and write the below code in "LoginPage.aspx.cs" file:

Code under Page_Load:

```
if (!IsPostBack)
 txtName.Focus();
```

Code under Reset Button:

```
txtName.Text = txtPwd.Text = "";
txtName.Focus();
```

Code under Login Button:

```
if (txtName.Text == "admin" && txtPwd.Text == "admin#123")
 Server.Transfer("SuccessPage.aspx");
else
 Response.Redirect("FailurePage.aspx");
```

Now go to "SuccessPage.aspx" and set "style" attribute for "<body>" tag which should look as below:

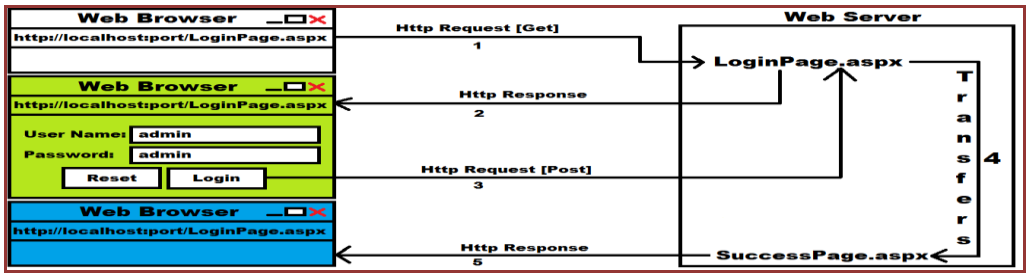
```
<body style="background-color:deepskyblue">
```

Now go to "FailurePage.aspx" and set "style" attribute for "<body>" tag which should look as below:

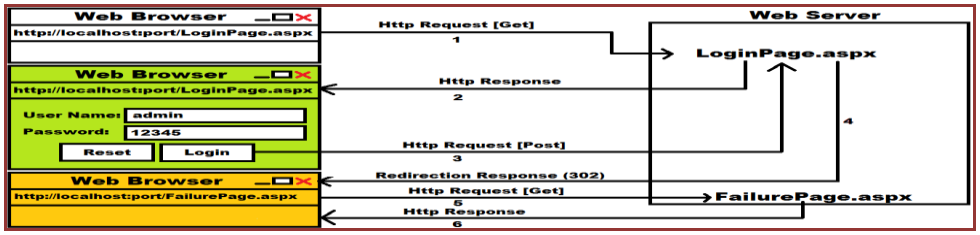
```
<body style="background-color:gold">
```

Now run "LoginPage.aspx" enter valid credentials and click on "Login" button which will take you to "SuccessPage.aspx" and when we enter any in-valid credentials and click on "Login" button it will take you to "FailurePage.aspx".

Server.Transfer: in this case the transfer between source and target pages will be performed within the server only i.e., when we enter valid credentials and click on the Login button first it goes to "LoginPage" because it is a "Postback" and from there "Server.Transfer" will transfer the control to SuccessPage as following:



Response.Redirect: in this case the transfer between source and target pages will be performed thru the browser i.e., when we enter in-valid credentials and click on the Login button first it goes to "LoginPage" because it is a "PostBack" and from there "Response.Redirect" will send a re-direction (302) response to the browser, asking the browser to make a new request to server for "FailurePage", so browser will make a request to the server for "FailurePage" and recieves the page as response as following:



Accessing Source Page values in Target Pages when the control is transferred from one Page to another Page:

In case of Server.Transfer, target page i.e., "SuccessPage.aspx" can directly access values from source page i.e., "LoginPage.aspx" and this can be done in 2 different ways:

1. Target pages can directly read Form Collection or Query String Collection values that are submitted to Source Page by using Request object and to test this write the below code in Page_Load method of "SuccessPage.aspx.cs" file:

```
string Name = Request["txtName"];
Response.Write("Hello " + Name + ", welcome to the site.");
```

Note: by using the above approach we can only read Text Property Value of Input Controls that are present in Source Page, on the Target Page but we can't read any other members of that Control as well as any other Member of the PreviousPage.

2. By using PreviousPage property in Target Page we can get the reference of Source Page and then access all the members of Source Page i.e., Control Members as well as any other member of the Page like Properties, Methods, Events etc. To test this, re-write the code present in Page_Load method of SuccessPage.aspx.cs file as following:

```
Page pp = PreviousPage;
Control ctrl = pp.FindControl("txtName");
TextBox tb = (TextBox)ctrl;
string Name = tb.Text;
Response.Write("Hello " + Name + ", welcome to the site.");
```

or

```
Control ctrl = PreviousPage.FindControl("txtName");
TextBox tb = (TextBox)ctrl;
string Name = tb.Text;
```

```
Response.Write("Hello " + Name + ", welcome to the site.");
                                or
TextBox tb = (TextBox)PreviousPage.FindControl("txtName");
string Name = tb.Text;
Response.Write("Hello " + Name + ", welcome to the site.");
                                or
string Name = ((TextBox)PreviousPage.FindControl("txtName")).Text;
Response.Write("Hello " + Name + ", welcome to the site.");
```

In case of Response.Redirect target page i.e., "FailurePage" can't access any values of source page i.e., "LoginPage" directly and if we want the values of source page in target page then we need to explicitly transfer those values as a "Query String" by concatenating to the Url of the Page to where we are transferring the control. To test this, first go to "LoginPage.aspx.cs" file and re-write the statement "Response.Redirect" which is present in the else part of Login Button - Click Event Handler as following:

Old Code:

```
Response.Redirect("FailurePage.aspx");
```

New Code:

```
Response.Redirect("FailurePage.aspx?Name=" + txtName.Text);
```

Now we can read this value in "FailurePage" from the "QueryString" Collection by using the Request object and to test that write this code in "Page_Load" Event Handler of "FailurePage.aspx.cs" file:

```
string Name = Request.QueryString["Name"];
Response.Write("Hello " + Name + ", your credentials are invalid.");
```

Differences between Server.Transfer and Response.Redirect methods:

1. In case of Server.Transfer, transfer between pages is performed directly on the Web Server it-self, whereas in case of Response.Redirect, transfer between pages is performed from the browser.

2. Server.Transfer is faster in execution because it will not have any extra round trips between the browser and server whereas in case of the second i.e., Response.Redirect there will be extra round trips between the browser and server, so execution is not faster.

3. In case of Server.Transfer by using the PreviousPage property on target page we can access the source page and its values, whereas in case of Response.Redirect this will not be possible.

4. In case of Server.Transfer the Url in browser's Address bar will not be updated with new page Address, so we can't bookmark the page whereas it happens in the case of Response.Redirect, so we can bookmark the page.

5. In case of Server.Transfer we can transfer only to the pages of same site whereas in the case of Response.Redirect we can transfer to pages of same site as well as other sites on the same server as well as other servers also.

RadioButton and CheckBox Controls: we use these controls to provide the users with a list of values to choose from, so that users can select from those values. Radio Buttons are used in-case we want to provide an option for single-selection and Checkbox's are used in-case we want to provide an option for multi-selection. Both Radio

Button and Check Box Controls provides a boolean property called "Checked" using which we can recognize whether the control is selected or not, which returns true if selected and false if not selected.

Note: in-case of Radio Button control we need to explicitly tell from how many Radio Button's 1 value should be selected and to do that we need to use "Group Name" attribute and group the Radio Button's so that 1 Radio Button can be selected from the group.

To work with "Check Box" and "Radio Button" controls add a new Web Form in our project naming it as "RadioAndCheck.aspx" and write the below code under " " tag:

```
Name:
<asp:TextBox ID="txtName" runat="server" /> <br />
Gender:
<asp:RadioButton GroupName="Gender" ID="rbMale" runat="server" Text="Male" />
<asp:RadioButton GroupName="Gender" ID="rbFemale" runat="server" Text="Female" />
<asp:RadioButton GroupName="Gender" ID="rbTrans" runat="server" Text="Transgender" /> <br />
Eating Habbit:
<asp:RadioButton GroupName="Habbit" ID="rbVeg" runat="server" Text="Vegetarian" />
<asp:RadioButton GroupName="Habbit" ID="rbNonVeg" runat="server" Text="Non Vegetarian" />
<asp:RadioButton GroupName="Habbit" ID="rbVegan" runat="server" Text="Vegan" /> <br />
Hobbies:
<asp:CheckBox ID="cbReading" runat="server" Text="Reading Books" />
<asp:CheckBox ID="cbPlaying" runat="server" Text="Playing Games" />
<asp:CheckBox ID="cbWatching" runat="server" Text="Watching Movies" /> <br />
<asp:Button ID="btnSubmit" runat="server" Text="Submit Values" /> <br />
<asp:Label ID="lblMsg" runat="server" ForeColor="Red" />
```

Now generate a Click Event Handler form "Display Values" button, import the namespace "System.Text" and write the below code in "RadioAndCheck.aspx.cs" file:

Code under Page Load Event:

```
if (!IsPostBack)
  txtName.Focus();
```

Code under Display Button Click Event:

```
StringBuilder sb = new StringBuilder();

if (txtName.Text.Trim().Length > 0)
  sb.Append("Name: " + txtName.Text + "<br />");

if (rbMale.Checked)
  sb.Append("Gender: Male <br />");
else if (rbFemale.Checked)
  sb.Append("Gender: Female <br />");
else if(rbTrans.Checked)
  sb.Append("Gender: Transgender <br />");

if(rbVeg.Checked)
```

```
  sb.Append("Eating Habbit: Vegetarian <br />");
else if(rbNonVeg.Checked)
  sb.Append("Eating Habbit: Non-Vegetarian <br />");
else if(rbVegan.Checked)
  sb.Append("Eating Habbit: Vegan <br />");

List<string> values = new List<string>();
if (cbReading.Checked)
  values.Add("Reading Books");
if (cbPlaying.Checked)
  values.Add("Playing Games");
if (cbWatching.Checked)
  values.Add("Watches Movies");

if (values.Count > 0)
  sb.Append("Hobbies: " + String.Join(", ", values));

lblMsg.Text = sb.ToString();
```

List Controls

These controls are also used for providing the users with a list of values to choose from. We have 4 List Controls in ASP.NET and those are:

1. DropDownList
2. ListBox
3. RadioButtonList
4. CheckBoxList

- **DropDownList:** default visible items are 1 and we can select only 1 item.
- **ListBox:** default visible items are multiple, and we can select 1 or more items.
- **RadioButtonList:** default visible items are multiple, but we can select only 1 item.
- **CheckBoxList:** default visible items are multiple, and we can select 1 or more items.

To work with these controls first we need to add values into them, and every value of a List Control is known as a ListItem (class) and each value is an instance of this class. Every ListItem is associated with 4 attributes:

- Text
- Value
- Enabled
- Selected

- Text refers to the "Display Member" of the control i.e., what user views.
- Value refers to the "Value Member" of the control i.e., this is not visible to end users but used for implementing the background logic.
- Enabled is a boolean property with default value true and if set as false this ListItem is not visible to end users.
- Selected is a boolean property with default value false and if set as true this ListItem is selected by default.

To work with List Controls, add a new Web Form in our project naming it as "ListControls1.aspx", add 1 DropDownList, ListBox, RadioButtonList and CheckBoxList controls on the Form.

Adding List Item's into List Controls: to add List Item's into a List Controls we are provided with 5 different options, those are:

1. By using the Items property in properties window we can add List Item's into a List Control and to do that, in the design view of Web Form select "DropDownList", go to its properties, select Items property and click on the button beside it which open "ListItem Collection Editor" window, click on the "Add Button" which will add a new ListItem in the LHS and now on the RHS it will display the 4 attributes where we can enter "Text & Value" for each ListItem. Follow this process and add 6 Countries into the Control with below Text & Value.

Text	Value
Italy	C1
India	C2
Japan	C3
France	C4
America	C5
Australia	C6

Note: This action will write all the necessary code in Source file and to view that code go to Source View and watch the "Inner Html" of "DropDownList" which will be as following:

```
<asp:ListItem Value="C1">India</asp:ListItem>
<asp:ListItem Value="C2">Japan</asp:ListItem>
<asp:ListItem Value="C3">China</asp:ListItem>
<asp:ListItem Value="C4">France</asp:ListItem>
<asp:ListItem Value="C5">America</asp:ListItem>
<asp:ListItem Value="C6">Australia</asp:ListItem>
```

2. Same as the above code what VS has generated for "DropDownList", we can also manually write that code in Source View and to test that write the following code as "Inner Html" to "ListBox":

```
<asp:ListItem Text="Kerala" Value="S1" />
<asp:ListItem Text="Tamilnadu" Value="S2" />
<asp:ListItem Text="Karnataka" Value="S3" />
<asp:ListItem Text="Telangana" Value="S4" />
<asp:ListItem Text="Maharastra" Value="S5" />
<asp:ListItem Text="Andhra Pradesh" Value="S6" />
```

3. We can also add Item's to List Controls thru "C# Code" in Code View by calling "Items.Add" method on the "ListControl" and this method is an overloaded method which is provided with 2 overloads:

```
<ListControl>.Items.Add(string Item)
<ListControl>.Items.Add(ListItem Item)
```

Note: in case of the first overload, we are passing string as a parameter and internally it creates a "ListItem" and the "Text & Value" will be same i.e., the string what we passed over there only will be taken as "Text & Value" also, whereas in case of the second overload we need to explicitly create a "ListItem" with a "Text & Value", and add it to the Control. To test the above go to "ListControl1.aspx.cs" file and write the below code in "Page_Load" Event Handler:

```
RadioButtonList1.Items.Add("Delhi");
RadioButtonList1.Items.Add("Kolkata");
RadioButtonList1.Items.Add("Mumbai");
RadioButtonList1.Items.Add(new ListItem("Chennai", "City4"));
RadioButtonList1.Items.Add(new ListItem("Bengaluru", "City5"));
RadioButtonList1.Items.Add(new ListItem("Hyderabad", "City6"));
```

4. Same as the above we have another method "Items.AddRange" and by using this we can add an Array of List Item's at a time into the control.

```
<ListControl>.Items.AddRange(ListItem[] Items)
```

To test the above go to "ListControls1.aspx.cs" file and write the below code in "Page_Load" Event Handler:

```
ListItem color1 = new ListItem("Red", "Color1");
ListItem color2 = new ListItem("Blue", "Color2");
ListItem color3 = new ListItem("White", "Color3");
ListItem color4 = new ListItem("Green", "Color4");
ListItem color5 = new ListItem("Purple", "Color5");
ListItem color6 = new ListItem("Magenta", "Color6");
ListItem[] colors = new ListItem[] { color1, color2, color3, color4, color5, color6 };
CheckBoxList1.Items.AddRange(colors);
                    or
ListItem[] colors = new ListItem[]
{
  new ListItem("Red", "Color1"),
  new ListItem("Blue", "Color2"),
  new ListItem("White", "Color3"),
  new ListItem("Green", "Color4"),
  new ListItem("Purple", "Color5"),
  new ListItem("Magenta", "Color6")
};
CheckBoxList1.Items.AddRange(colors);
                    or
CheckBoxList1.Items.AddRange(new ListItem[]
{
  new ListItem("Red", "Color1"),
  new ListItem("Blue", "Color2"),
  new ListItem("White", "Color3"),
  new ListItem("Green", "Color4"),
  new ListItem("Purple", "Color5"),
  new ListItem("Magenta", "Color6")
});
```

5. By using "DataSource" property of List Control's we can assign values from any DataSource like a File or Database also.

```
<ListControl>.DataSource = <Table>;
<ListControl>.DataTextField = <Column_Name>;
<ListControl>.DataValueField = <Column_Name>;
<ListControl>.DataBind();
```

Behavior of each List Control:

1. **DropDownList:** This control is used in-case we want to provide the option for single selection.
2. **ListBox:** This control by default supports single selection only whereas if we want multi-selection then we need to set "SelectionMode" property of the control as "Multiple" because, default is "Single".
3. **RadioButtonList:** This control is also like DropDownList control, providing support for single selection only whereas in case of RadioButtonList we find a Radio Button beside each item for selection.
4. **CheckBoxList:** this control works same as a RadioButtonList, but the only difference is it supports Multi-Selection with a CheckBox beside each item for selection.

Note: RadioButtonList and CheckBoxList provides a property "RepeatDirection" and by using it we can specify the direction in which "List Items" are laid out, default is "Vertical" direction and can be changed to "Horizontal". By using the "RepeatLayout" property we can set layout for "List Items" which is by default "Table" but can be changed to "Flow", "OrderedList" and "UnOrderedList" (OrderedList and UnOrderList layout will not work in case the "RepeatDirection" is set as Horizontal i.e., will work only in case of Vertical). By using "RepeatColumns" property we can specify the number of columns to layout the "List Items".

Accessing values from ListControls: to access values from ListControls we are provided with a set of Members under them as following:

1. **Items:** by using this property we can access all the Items of a ListControl which returns them in the form of ListItem Array.

   <ListControl>.Items => ListItem[]

2. **SelectedItem:** by using this property we can access the "SelectedItem" form the ListControl which will return the value as a ListItem.

   <ListControl>.SelectedItem => ListItem (Text and Value)

3. **SelectedValue:** this property returns the "Value" that is associated with a selected "ListItem" in the form of a string.

   <ListControl>.SelectedValue => String (Only Value)

4. **Text:** this property is also same as "SelectedValue" only which returns the "Value" that is associated with a selected ListItem in the form of a string.

   <ListControl>.Text => String (Only Value)

5. **SelectedIndex:** this property returns the index position of the selected "ListItem" in the form of int.

   <ListControl>.SelectedIndex => int

6. **GetSelectedIndices():** this is a method present under "ListBox" control that returns an array of indices corresponding to the SelectedItems.

   <ListBox>.GetSelectedIdices => int[]

Note: the first 5 members are available under all the controls whereas the last member i.e., "GetSelectedIndices" is specific to "ListBox" control.

XML: Extensible Markup Language

- This is also a markup language like HTML but used for describing the content of a Text Document.
- This is used in the industry for transporting the data from 1 machine to another machine apart from Excel and CSV (Comma Separated Values).
- This can also be used as a Temporary Database in scenarios where we don't have access to a Live Database.
- XML is Platform Independent which can be used in any O.S. and well as every application provides options to read data from XML.
- Just like HTML, XML data is also present under Tags, but the difference is HTML tags are pre-defined whereas XML tags are user-defined.
- An XML document should be saved with ".xml" extension.
- An XML document can have 1 & only 1 root element.
- A simple XML documents looks as following:

```
<Employees>
  <Employee SerialNo="1">
    <Id>1001</Id>
    <Name>Scott</Name>
    <Job>Manager</Job>
    <Salary>50000</Salary>
  </Employee>
  <Employee SerialNo="2">
    <Id>1002</Id>
    <Name>Smith</Name>
    <Job>Salasman</Job>
    <Salary>15000</Salary>
  </Employee>
  <!--Enter multiple employees data-->
</Employees>
```

- XML is case sensitive, so we need to strictly follow the same case for starting and ending tag.

  `<Id>101</Id>` => Valid

  `<Id>101</ID>` => Invalid
- To understand XML, we need a software known as XML Parser and this is built into every browser.
- XML tags also can have attributes, for example we have defined "SerialNo" as an attribute for "Employee" tag, but values to those attributes must be enclosed in double quotes (Mandatory).

Note: our "aspx" code in ASP.NET completely follows XML syntaxes only.

Loading Data into List Control's from an XML File: to load Data into List Controls from XML File do the below.

Step 1: Add a Web Form naming it is as "ListControls2.aspx" and write the following code under its " " tag:

```
<table align="center" border="1">
  <caption>Product Catalog</caption>
  <tr>
    <td align="center"><asp:DropDownList ID="DropDownList1" runat="server" /></td>
    <td align="center"><asp:ListBox ID="ListBox1" runat="server" SelectionMode="Multiple" /></td>
    <td align="center"><asp:RadioButtonList ID="RadioButtonList1" runat="server" /></td>
    <td align="center"><asp:CheckBoxList ID="CheckBoxList1" runat="server" /></td>
  </tr>
  <tr>
    <td align="center"><asp:Button ID="Button1" runat="server" Text="Show Selected Product" /></td>
    <td align="center"><asp:Button ID="Button2" runat="server" Text="Show Selected Products" /></td>
    <td align="center"><asp:Button ID="Button3" runat="server" Text="Show Selected Product" /></td>
    <td align="center"><asp:Button ID="Button4" runat="server" Text="Show Selected Products" /></td>
  </tr>
  <tr>
    <td><asp:Label ID="Label1" runat="server" ForeColor="Red" /></td>
    <td><asp:Label ID="Label2" runat="server" ForeColor="Red" /></td>
    <td><asp:Label ID="Label3" runat="server" ForeColor="Red" /></td>
    <td><asp:Label ID="Label4" runat="server" ForeColor="Red" /></td>
  </tr>
</table>
```

Step 2: In "Add New Item" window, choose XML File item, name it as "Products.xml" and write the below code:

```
<Products>
  <Product>
    <Id>101</Id>
    <Name>Television</Name>
  </Product>
  <Product>
    <Id>102</Id>
    <Name>Microwave</Name>
  </Product>
  <Product>
    <Id>103</Id>
    <Name>Washing Machine</Name>
  </Product>
  <Product>
    <Id>104</Id>
    <Name>Refrigerator</Name>
  </Product>
  <Product>
    <Id>105</Id>
    <Name>Home Theatre</Name>
  </Product>
  <Product>
    <Id>106</Id>
    <Name>Mixer Grinder</Name>
  </Product>
  <Product>
    <Id>107</Id>
    <Name>Blu-ray Player</Name>
  </Product>
  <Product>
    <Id>108</Id>
    <Name>Split A/C</Name>
  </Product>
  <Product>
    <Id>109</Id>
    <Name>Air Cooler</Name>
  </Product>
  <Product>
    <Id>110</Id>
    <Name>Window A/C</Name>
  </Product>
</Products>
```

Step 3: Now go to design view of the WebForm, generate Event Handlers for all the 4 Buttons and write the below code in "ListControls2.aspx.cs" file:

```
using System.Data;
```

Code under Page Load Event:

```
if (!IsPostBack) {
  string xmlFilePath = Server.MapPath("~/Products.xml");
  DataSet ds = new DataSet();
  ds.ReadXml(xmlFilePath);
  DropDownList1.DataSource = ds;
  DropDownList1.DataTextField = "Name";
  DropDownList1.DataValueField = "Id";
  DropDownList1.DataBind();
  DropDownList1.Items.Insert(0, "-Select Product-");
  ListBox1.DataSource = ds;
  ListBox1.DataTextField = "Name";
  ListBox1.DataValueField = "Id";
  ListBox1.DataBind();
  RadioButtonList1.DataSource = ds;
  RadioButtonList1.DataTextField = "Name";
  RadioButtonList1.DataValueField = "Id";
  RadioButtonList1.DataBind();
  CheckBoxList1.DataSource = ds;
  CheckBoxList1.DataTextField = "Name";
  CheckBoxList1.DataValueField = "Id";
  CheckBoxList1.DataBind();
}
```

Code under Button1 Click Event:

```
if (DropDownList1.SelectedIndex > 0) {
  ListItem li = DropDownList1.SelectedItem;
  Label1.Text = li.Value + ": " + li.Text;
}
else {
  Label1.Text = "";
}
```

Code under Button2 Click Event:

```
Label2.Text = "";
foreach(int index in ListBox1.GetSelectedIndices()) {
  ListItem li = ListBox1.Items[index];
  Label2.Text += li.Value + ": " + li.Text + "<br />";
}
```

Code under Button3 Click Event:

```
if (RadioButtonList1.SelectedItem != null) {
  ListItem li = RadioButtonList1.SelectedItem;
  Label3.Text = li.Value + ": " + li.Text;
}
```

Code under Button4 Click Event:

```
Label4.Text = "";
foreach(ListItem li in CheckBoxList1.Items) {
  if(li.Selected) {
    Label4.Text += li.Value + ": " + li.Text + "<br />";
  }
}
```

DataSet is a class, present under "System.Data" namespace which can read the data from an XML File and hold it in a Table format.

MapPath is a method of "HttpServerUtility" class, and this method returns the physical path of a file when we pass the virtual path of the file.

BulletedList: this is also a List Control which can be used for displaying the values but doesn't support selection of values. This control by default displays ListItem's that are added to it as an "UnOrdered" List, which can be changed to different styles by using the "BulletStyle" Property of the control and this "BulletStyle" property can be set with any of the below values:

- Not Set [d]
- Numbered
- Lower Alpha
- Upper Alpha
- Lower Roman
- Upper Roman
- Disc
- Circle
- Square
- CustomImage

Note: if we want the "Bullet Style" to be a "Custom Image" then do the below:

- Add a new folder under the project and name it as "Icons".
- Add an image into the "Icons" folder which we want to set as Custom Image.
- Set the "BulletStyle" property values as "CustomImage".
- Use the "BulletImageUrl" property of the control and select the Image from Icons folder.

To test this, add a new Web Form in the project naming it as "ListControls3.aspx" and write the below code under the " " tag of Form:

```
<asp:BulletedList ID="BulletedList1" runat="server" BulletStyle="CustomImage"
                  BulletImageUrl="~/Icons/arrow_right.png" >
  <asp:ListItem Value="Color1" Text="Red" />
  <asp:ListItem Value="Color2" Text="Blue" />
  <asp:ListItem Value="Color3" Text="Black" />
  <asp:ListItem Value="Color4" Text="Green" />
  <asp:ListItem Value="Color5" Text="White" />
  <asp:ListItem Value="Color6" Text="Purple" />
  <asp:ListItem Value="Color7" Text="Yellow" />
```

```
  <asp:ListItem Value="Color8" Text="Magenta" />
</asp:BulletedList>
```

Now use the "BulletStyle" property and set the style that has to be used for the bullets and by using the "DisplayMode" property of the control we can specify how the Text should be displayed by the control and this can be set with any of the below values:

1. Text [d]
2. HyperLink
3. LinkButton

FileUpload Control

This control provides an interface to the end user for selecting a file to upload. Once the file is selected by the user, we can call members of "HttpPostedFile" class to upload the selected file to Web Server. To work with "FileUpload" control add a new Web Form in our project naming it as "FileDemo1.aspx" and write the following code under its " " tag:

```
<asp:FileUpload ID="FileUpload1" runat="server" />
<br />
<br />
<asp:Button ID="btnUpload" runat="server" Text="Upload File" />
<br />
<asp:Label ID="lblMsg" runat="server" ForeColor="Red" />
```

Now go to "FileDemo1.aspx.cs" file and write the below code under "Upload File" button Click Event Handler by import "System.IO" namespace:

```
//Code for creating a new folder under the project if it does not exist:
string physicalPath = Server.MapPath("~/Uploads/");
if (!Directory.Exists(physicalPath)) {
  Directory.CreateDirectory(physicalPath);
}
//Code for uploading and saving the file selected by user in FileUpload control:
HttpPostedFile selectedFile = FileUpload1.PostedFile;
selectedFile.SaveAs(physicalPath + selectedFile.FileName);
lblMsg.Text = selectedFile.FileName + " uploaded to the server.";
```

"Directory" is a class under "System.IO" namespace and "Exists" is a static method under the class which determines whether the given path refers to an existing folder or file on the hard disk and returns a Boolean value. "CreateDirectory" is also a static method of the "Directory" class which will create a directory at the specified path.

"PostedFile" is a property of "FileUpload" class which will return the "HttpPostedFile" class's object for the file that is selected under "FileUpload" control. "HttpPostedFile" class provides the complete info of the file that is selected under "FileUpload" control like, "FileName", "ContentLength", "ContentType" etc, and provides a method known "SaveAs" to save the file at the desired location.

Uploading and Saving a File with conditions: in the previous example we have uploaded and saved the file without any conditions, whereas let us also check some conditions before uploading and save the file, like:

1. Before the user clicks on Upload File button there should be a file selected or else it should display a user-friendly error message.
2. The file being uploaded should be an image file of the following extensions like: "jpg, png and bmp" only.
3. The file being uploaded should be less than or equal to 512 KB size.

Add a new Web Form in the project naming it as "FileDemo2.aspx" and write the below code in its " " tag:

```
<asp:FileUpload ID="FileUpload1" runat="server" /><br /><br />
<asp:Button ID="btnUpload" runat="server" Text="Upload File" /><br />
<asp:Label ID="lblMsg" runat="server" ForeColor="Red" />
```

Now go to **"FileDemo2.aspx.cs"** file and write the following code under "Upload File" button Click Event Handler by importing "System.IO" namespace:

```
if (FileUpload1.HasFile) {
  HttpPostedFile selectedFile = FileUpload1.PostedFile;
  string fileType = selectedFile.ContentType.Substring(0, selectedFile.ContentType.IndexOf("/"));
  if (fileType == "image") {
    if (selectedFile.ContentLength <= 524288) {
      string folderPath = Server.MapPath("~/Uploads/");
      if (!Directory.Exists(folderPath)) {
        Directory.CreateDirectory(folderPath);
      }
      selectedFile.SaveAs(folderPath + selectedFile.FileName);
      lblMsg.Text = selectedFile.FileName + " is uploaded to the server.";
    }
    else {
      lblMsg.Text = "Upload image can't be greater than 512 kb size.";
    }
  }
  else {
    lblMsg.Text = "Please select only image files to upload.";
  }
}
else {
  lblMsg.Text = "Please select an image file to upload.";
}
```

"HasFile" and "HasFiles" are properties of FileUpload class which returns true if a file(s) has been selected by end user under the FileUpload control.

"ContentType" is a property of HttpPostedFile class which gets the Content Type (MIME) of the file sent by client and "ContentLength" is a property of HttpPostedFile class which gets the size of an uploaded file in bytes.

Uploading multiple files using FileUpload control: By default, "FileUpload" control provides an option to select only a single file for uploading, whereas if we want to provide an option for multi-selection then we need to set the "AllowMultiple" attribute of "FileUpload" control as true because by default it is false.

Add a new Web Form in the project naming it as "FileDemo3.aspx" and write the below code in its " " tag:

```
<asp:FileUpload ID="FileUpload1" runat="server" AllowMultiple="true" />
<br /><br />
<asp:Button ID="btnUpload" runat="server" Text="Upload Files" />
<br />
<asp:Label ID="lblMsg" runat="server" ForeColor="Red" />
```

Now go to **"FileDemo3.aspx.cs"** file and write the below code under "Upload Files" button Click Event Handler by importing "System.IO" namespace:

```
string physicalPath = Server.MapPath("~/Uploads/");
if (!Directory.Exists(physicalPath)) {
  Directory.CreateDirectory(physicalPath);
}
if (FileUpload1.HasFiles) {
  int Count = 0;
  foreach (HttpPostedFile selectedFile in FileUpload1.PostedFiles) {
    Count += 1;
    selectedFile.SaveAs(physicalPath + selectedFile.FileName);
  }
  lblMsg.Text = Count + " file(s) uploaded to the server.";
}
else {
  lblMsg.Text = "Please select a file(s) for uploading.";
}
```

"PostedFiles" is a property of "FileUpload" class which returns a List of "HttpPostedFile" class objects representing each file that is selected under "FileUpload" control.

Calendar Control

This control is used for providing the users an option for selecting a Date. Calendar control will render an Html table. This control is provided with the following set of members to work with it:

1. **SelectedDate:** this property can be used for setting or getting the Date selected under the control and the type of this property is DateTime.
2. **SelectedDates:** this property can be used for getting the List of Dates that are selected under the control in the form of an Array.
3. **Caption:** this property is used for setting a caption to the Calendar control.
4. **DayNameFormat:** this property is to specify format for day "Header Text" which can be set with any of the following values: Full, Short [d], FirstLetter, FirstTwoLetters or Shortest.
5. **FirstDayOfWeek:** this property is to specify which day of the week has to be displayed first in the Calendar; default is Sunday and can be changed to any day of the week.
6. **NextPrevFormat:** this property is to specify format for month navigation buttons and this property can be set with any of the following values: ShortMonth, FullMonth & CustomText [d]. If we are using CustomText then the text (caption) for "Next Month Button" and "Previous Month Button" can be specified by using the properties "NextMonthText, who's default value is: >" and "PreviousMonthText, who's default value is: <".
7. **SelectionMode:** this property determines whether the selection should be Day [d] or DayWeek or DayWeekMonth or None.

   Note: when the "SelectionMode" is set as "DayWeek" or "DayWeekMonth" then we can specify the text that has to displayed for select week button and select month button by using the properties "SelectWeekText, who's default value is: >" and "SelectMonthText, who's default value is: >>".

8. **SelectionChanged**: this is an event which fires when the Date selection is changed, and this is the default event of Calendar.
9. **DayRender:** this is also an event which fires for each day being rendered by the Calendar i.e., if we launch the Calendar for 1 time this event will fire for 42 times.
10. **VisibleMonthChanged:** this is also an event which fires when we navigate to Next or Previous months by clicking on "Next Month Button" and "Previous Month Button".

To work with a Calendar, first download any Calendar image on to your computer and add to the Icons folder in our project. Now add a new WebForm in the project naming it as "CalendarDemo1.aspx" and write the below code in its " " tag:

```
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
  <ContentTemplate>
    Enter Date: <asp:TextBox ID="txtDate" runat="server" />
    <asp:ImageButton ID="imgDate" runat="server" ImageUrl="~/Icons/Calendar.ico" ImageAlign="AbsMiddle"
                                        Width="20" Height="20" />
    <asp:Calendar ID="cldDate" runat="server" Visible="false" />
  </ContentTemplate>
</asp:UpdatePanel>
```

Now go to design view of the Web Form and then generate a "Click" Event Handler for "ImageButton" and "SelectionChanged" Event Handler for "Calendar" control and write the below code under "CalendarDemo1.aspx.cs" file:

Code under Page Load Event:

```
if (!IsPostBack) {
  txtDate.Focus();
}
```

Code under Image Button Click Event:

```
if (cldDate.Visible == false) {
  cldDate.Visible = true;
}
else {
  cldDate.Visible = false;
}
```

Code under Calendar SelectionChanged Event:

```
txtDate.Text = cldDate.SelectedDate.ToShortDateString();
cldDate.Visible = false;
```

Customizing a Calendar Control: in the previous example end user can select any date from the Calendar, but let us customize the Calendar to restrict user selection based on the following rules:

- Should not allow selection of previous dates and today's date in the Calendar.
- Should not allow selection of Sundays in the current year.
- Should not allow selection of second Saturdays in the current year.
- Should not allow selection of any Dates that are listed in the "HolidayList.xml" file.
- Should not allow selection of any date in the previous year or next year.
- Should not allow navigation to previous months to the current month and navigation to next months after December of the current year.

To perform all the above do the following:

Step 1: Add an XML File in the project naming it as "HolidayList.xml" and enter the list of holidays for the current year as below.

```
<Holidays>
  <Holiday>
    <Date>1/1/2021</Date>
    <Reason>New Year</Reason>
  </Holiday>
  <Holiday>
    <Date>1/14/2021</Date>
    <Reason>Pongal</Reason>
  </Holiday>
  <Holiday>
    <Date>1/26/2020</Date>
    <Reason>Republic Day</Reason>
  </Holiday>
  <Holiday>
    <Date>3/28/2021</Date>
    <Reason>Holi</Reason>
  </Holiday>
  <Holiday>
    <Date>4/13/2021</Date>
    <Reason>Ugadi</Reason>
  </Holiday>
  <Holiday>
    <Date>5/12/2021</Date>
    <Reason>Ramzan</Reason>
  </Holiday>
  <Holiday>
    <Date>8/15/2021</Date>
    <Reason>Independence Day</Reason>
  </Holiday>
  <Holiday>
    <Date>9/10/2021</Date>
    <Reason>Ganesh Chaturdhi</Reason>
  </Holiday>
  <Holiday>
    <Date>10/2/2021</Date>
    <Reason>Gandhi Jayanthi</Reason>
  </Holiday>
<Holiday>
   <Date>10/5/2021</Date>
   <Reason>Dussara</Reason>
  </Holiday>
  <Holiday>
```

```
  <Date>11/4/2021</Date>
  <Reason>Diwali</Reason>
 </Holiday>
 <Holiday>
  <Date>12/25/2021</Date>
  <Reason>Christmas</Reason>
 </Holiday>
</Holidays>
```

Step 2: Now add, a new Web Form in the project naming it as "CalendarDemo2.aspx" and write the below code in its " " tag:

```
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
  <ContentTemplate>
    Enter Date: <asp:TextBox ID="txtDate" runat="server" />
    <asp:ImageButton ID="imgDate" runat="server" ImageUrl="~/Icons/Calendar.ico" ImageAlign="AbsMiddle"
                                    Width="20" Height="20" />
    <asp:Calendar ID="cldDate" runat="server" Visible="false" />
  </ContentTemplate>
</asp:UpdatePanel>
```

Now go to design view of the Web Form and generate a "Click" Event Handler for ImageButton, and "SelectionChanged", "DayRender" and "VisibleMonthChanged" Event Handlers for Calendar, and write the below code under "CalendarDemo2.aspx.cs" file:

```
using System.Data;
using System.Drawing;
```

Code under Page Load Event:

```
if (DateTime.Now.Month == 12) {
  cldDate.NextMonthText = "";
}
```

Code under ImageButton Click Event:

```
if (cldDate.Visible == false) {
  cldDate.Visible = true;
}
else {
  cldDate.Visible = false;
}
```

Code under Calendar SelectionChanged Event:

```
txtDate.Text = cldDate.SelectedDate.ToShortDateString();
cldDate.Visible = false;
```

Code under Calendar DayRender Event:

```
//Code for checking dates of Holiday List
DataSet ds = new DataSet();
```

```
ds.ReadXml(Server.MapPath("HolidayList.xml"));
for (int i = 0; i < ds.Tables[0].Rows.Count; i++) {
  if (ds.Tables[0].Rows[i]["Date"].ToString() == e.Day.Date.ToShortDateString()) {
    e.Day.IsSelectable = false;
    e.Cell.BackColor = Color.LightGreen;
    e.Cell.ToolTip = ds.Tables[0].Rows[i]["Reason"].ToString();
  }
}
//Code for checking Second Saturdays
if (e.Day.Date.Day >= 8 && e.Day.Date.Day <= 14) {
  if (e.Day.Date.DayOfWeek == DayOfWeek.Saturday) {
    e.Day.IsSelectable = false;
    e.Cell.BackColor = Color.LightCoral;
    e.Cell.ToolTip = "Second Saturday";
  }
}
//Code for checking Sundays
if (e.Day.Date.DayOfWeek == DayOfWeek.Sunday) {
  e.Day.IsSelectable = false;
  e.Cell.BackColor = Color.LightSeaGreen;
  e.Cell.ToolTip = "Sunday";
}
//Code for checking expired dates
if (e.Day.Date <= DateTime.Now.Date) {
  e.Day.IsSelectable = false;
  e.Cell.BackColor = Color.LightGray;
  e.Cell.ToolTip = "Date Expired";
}
//Code for checking current year
if (e.Day.Date.Year != DateTime.Now.Year) {
  e.Day.IsSelectable = false;
  e.Cell.BackColor = Color.LightCyan;
  e.Cell.ToolTip = "Out of range";
}
```

Code under Calendar VisibleMonthChanged Event:

```
//Code for enabling and disabling of "Next Month" and "Previous Month" buttons:
if (e.NewDate.Month == 12) {
  cldDate.NextMonthText = "";
}
else {
  cldDate.NextMonthText = "&gt;";
}
if (e.NewDate.Month == DateTime.Now.Month) {
  cldDate.PrevMonthText = "";
}
else {
```

```
  cldDate.PrevMonthText = "&lt;";
}
```

AdRotator Control

AdRotator control is used to display a randomly selected advertisement as a banner on the Web Page. The displayed advertisement changes whenever the page refreshes. Advertisement information is stored in a separate XML file. The XML file allows you to maintain a list of advertisements and their associated attributes. Attributes include the path to an image to display, the URL to link to when the control is clicked, the alternate text to display when the image is not available, a keyword, and the frequency of the advertisement.

XML Advertisement File Format: The AdRotator control uses an XML advertisement file to store the advertisement information. When creating the advertisement file, opening, and closing <Advertisements> tags mark the beginning and the end of the file, respectively. All advertisements are nested between the opening and closing <Advertisements> tags. If the file contains multiple <Advertisements> tags, only the first set of <Advertisements> tags in the file will be parsed by the AdRotator control. All other <Advertisements> tags will be ignored.

```
<Advertisements>
  <Ad>
    <ImageUrl><!-- Image URL --></ImageUrl>
    <NavigateUrl><!-- Navigate URL --></NavigateUrl>
    <AlternateText><!-- Alternate Text --></AlternateText>
    <Impressions><!-- Frequency --></Impressions>
    <Keyword><!-- Keyword --></Keyword>
  </Ad>
</Advertisements>
```

- **<Advertisements>** tag marks the beginning and ending of the advertisement file.
- **<Ad>** tag marks the beginning and ending of each advertisement.
- **ImageUrl** tag is used to specify the absolute or relative URL to an image file.
- **NavigateUrl** tag is used to specify the URL of a page to link to when the user clicks the Ad.
- **AlternateText** tag is used to specify the text to display in place of the image when the image specified by the Image Url property is not available and, in some browsers, this text also appears as a ToolTip for the advertisement.

- **Impressions** tag is used to specify a number that indicates the importance of the ad in the schedule of rotation relative to the other ads in the file. Larger the number, the more often the ad is displayed. The total of all impressions values in the file cannot exceed 2,047,999,999. If it does then the control throws a run-time error.
- **Keyword** tag is used to specify a category for advertisements (for example, "Mobiles") that you can filter by.

To work with AdRotator do the following:

Step 1: Download images corresponding to Google, Facebook, Microsoft, Instagram, LinkedIn & Twitter (prefer horizontal images), add a new folder in the project naming it as "AdImages" and add the downloaded images into the folder.

Step 2: Add an XML file in our project naming it as "Ads.xml" and write the below code in it:

```
<Advertisements>
  <Ad>
    <ImageUrl>~/AdImages/Facebook.png</ImageUrl>
    <NavigateUrl>https://facebook.com</NavigateUrl>
    <AlternateText>Visit us at www.facebook.com</AlternateText>
    <Impressions>10</Impressions>
    <Keyword>SNS</Keyword>
  </Ad>
  <Ad>
    <ImageUrl>~/AdImages/Google.png</ImageUrl>
    <NavigateUrl>https://google.com</NavigateUrl>
    <AlternateText>Visit us at www.google.com</AlternateText>
    <Impressions>8</Impressions>
    <Keyword>ADC</Keyword>
  </Ad>
  <Ad>
    <ImageUrl>~/AdImages/Instagram.png</ImageUrl>
    <NavigateUrl>https://instagram.com</NavigateUrl>
    <AlternateText>Visit us at www.instagram.com</AlternateText>
    <Impressions>8</Impressions>
    <Keyword>SNS</Keyword>
  </Ad>
  <Ad>
    <ImageUrl>~/AdImages/LinkedIn.png</ImageUrl>
    <NavigateUrl>https://linkedin.com</NavigateUrl>
    <AlternateText>Visit us at www.linkedin.com</AlternateText>
    <Impressions>10</Impressions>
    <Keyword>SNS</Keyword>
  </Ad>
  <Ad>
    <ImageUrl>~/AdImages/Microsoft.png</ImageUrl>
    <NavigateUrl>https://microsoft.com</NavigateUrl>
    <AlternateText>Visit us at www.microsoft.com</AlternateText>
```

```
    <Impressions>12</Impressions>
    <Keyword>ADC</Keyword>
  </Ad>
  <Ad>
    <ImageUrl>~/AdImages/Twitter.png</ImageUrl>
    <NavigateUrl>https://twitter.com</NavigateUrl>
    <AlternateText>Visit us at www.twitter.com</AlternateText>
    <Impressions>6</Impressions>
    <Keyword>SNS</Keyword>
  </Ad>
</Advertisements>
```

Step 3: Add a new Web Form in the project, name it as "Ads.aspx" and write the following code under its tag:

```
<asp:AdRotator ID="AdRotator1" runat="server" Width="100%" Height="150" AdvertisementFile="~/Ads.xml"
                    Target="_blank" BorderStyle="Solid" BorderColor="Blue" />
```

Note: The "AdvertisementFile" property of the AdRotator control is used to specify the path of advertisement file.

Now run the Web Page and watch the output, and we notice the AdRotator control displaying an Ad from the list of Ad's listed in the XML File and now whenever we refresh the page, we notice the Ad's getting changed, whereas if we want the Ad's to be auto refreshed for any period use the Ajax Timer Control and to test it rewrite the above code under tag as following:

```
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<asp:Timer ID="Timer1" runat="server" Interval ="2000" />
<asp:UpdatePanel id="UpdatePanel1" runat="server">
  <Triggers><asp:AsyncPostBackTrigger ControlID="Timer1" /></Triggers>
  <ContentTemplate>
    <asp:AdRotator ID="AdRotator1" runat="server" Width="100%" Height="150" AdvertisementFile="~/Ads.xml"
                    Target="_blank" BorderStyle="Solid" BorderColor="Blue" />
  </ContentTemplate>
</asp:UpdatePanel>
```

Now when we run the Web Page, we notice that AdRotator control getting auto refreshed for every 5 seconds without refreshing the whole page.

Ad Filters: in the XML file, for every Ad we have specified a Keyword and we can use that Keyword for filtering the Ad's, and to test this go to "Ads.aspx.cs" file and write the following code under "Page Load" Event Handler:

```
AdRotator1.KeywordFilter = "SNS";
```

Now run the Web Page and watch the output where we notice the control displaying Ad's matching to "SNS" keyword only. The drawback in the above approach is we specified the filter value in "Source Code" and in Realtime "Source Code" is not accessible to anyone for modification, so clients can't edit and modify the Keyword value. So, to overcome that problem use Configuration File i.e., "Web.Config". To test this, store the filter value in this configuration file so that administrator of the Site can easily modify it in runtime and to do that open "Web.config" file and write the following code under "<configuration>" tag:

```
<appSettings>
```

```
  <add key="Filter" value="none" />
</appSettings>
```

Now go "Ads.aspx.cs" and re-write code which is present under "Page Load" Event Handler by importing "System.Configuration" namespace as following:

```
string AdFilter = ConfigurationManager.AppSettings.Get("AdFilter");
if (AdFilter != "None") {
  AdRotator1.KeywordFilter = AdFilter;
}
else {
  AdRotator1.KeywordFilter = "";
}
```

Now whenever we want to change the filter value without stopping the site, open "Web.config" file in its physical location and make the modification that is required, which directly reflects to the site in execution.

Consuming the Ad's page in multiple pages of our Site: to consume the Ad's page in all the pages of our site do the following:

Step 1: Add an Html file in the project, name it as "HomePage.html", delete the whole code in it and write the following:

```
<frameset rows="26%, 74%">
  <frame name="A" src="Ads.aspx" />
  <frameset cols="20%, 80%">
    <frame name="B" src="Pages.aspx" />
    <frame name="C" />
  </frameset>
</frameset>
```

Step 2: Add a new Web Form in the project, name it as "Pages.aspx" and write the below code under its tag:

```
<b>Click to navigate:</b>
<ul>
  <li><asp:HyperLink ID="hl1" runat="server" Text="Page1" NavigateUrl="~/Calculator1.aspx" Target="C" /></li>
  <li><asp:HyperLink ID="hl2" runat="server" Text="Page2" NavigateUrl="~/Calculator2.aspx" Target="C" /></li>
  <li><asp:HyperLink ID="hl3" runat="server" Text="Page3" NavigateUrl="~/ColorDialogs.aspx" Target="C" /></li>
  <li><asp:HyperLink ID="hl4" runat="server" Text="Page4" NavigateUrl="~/Contact.aspx" Target="C" /></li>
  <li><asp:HyperLink ID="hl5" runat="server" Text="Page5" NavigateUrl="~/LoginPage.aspx" Target="C" /></li>
  <li><asp:HyperLink ID="hl6" runat="server" Text="Page6" NavigateUrl="~/ListControls1.aspx" Target="C" /></li>
  <li><asp:HyperLink ID="hl7" runat="server" Text="Page7" NavigateUrl="~/ListControls2.aspx" Target="C" /></li>
  <li><asp:HyperLink ID="hl8" runat="server" Text="Page8" NavigateUrl="~/FileDemo1.aspx" Target="C" /></li>
  <li><asp:HyperLink ID="hl9" runat="server" Text="Page9" NavigateUrl="~/FileDemo2.aspx" Target="C" /></li>
  <li><asp:HyperLink ID="hl10" runat="server" Text="Page10" NavigateUrl="~/FileDemo3.aspx" Target="C" /></li>
</ul>
```

MultiView and View Controls

These controls allow us to divide the content of a page into different groups, displaying one group at a time. Each View control manages one group of content, and all the View controls are held together in a MultiView control. MultiView is a container control which is in turn a collection of Views responsible for displaying one View control at a time. By default, the control doesn't contain any View in it, so we need to add them explicitly. Even if the MultiView control is designed with multiple views, when we run the Web Page, the control will not display any View at all and if we want to display a View then we need to set the "ActiveViewIndex" property with the Index value of View which becomes the active view.

To work with these controls, add a new Web Form in the project, name it as "MultiViewDemo.aspx" and write the below code under its tag:

```
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
  <ContentTemplate>
    <asp:MultiView ID="MultiView1" runat="server">
      <asp:View ID="PDView" runat="server">
        <table align="center">
          <caption>Personal Details</caption>
          <tr>
            <td>First Name:</td>
            <td><asp:TextBox ID="txtFirstName" runat="server" /></td>
          </tr>
          <tr>
            <td>Last Name:</td>
            <td><asp:TextBox ID="txtLastName" runat="server" /></td>
          </tr>
          <tr>
            <td>Gender:</td>
            <td>
```

```
        <asp:RadioButton ID="rbMale" runat="server" GroupName="Gender" Text="Male" />
        <asp:RadioButton ID="rbFemale" runat="server" GroupName="Gender" Text="Female" />
      </td>
    </tr>
    <tr>
      <td>Date of Birth:</td>
      <td>
        <asp:TextBox ID="txtDOB" runat="server" />
        <asp:ImageButton ID="imgDOB" runat="server" ImageUrl="~/Icons/Calendar.ico" Width="20"
                                              Height="20" ImageAlign="AbsMiddle" />
        <asp:Calendar ID="cldDOB" runat="server" />
      </td>
    </tr>
    <tr>
      <td>Aadhar Id:</td>
      <td><asp:TextBox ID="txtAadhar" runat="server" /></td>
    </tr>
    <tr>
      <td>Mobile No:</td>
      <td><asp:TextBox ID="txtMobile" runat="server" /></td>
    </tr>
    <tr>
      <td>Email Id:</td>
      <td><asp:TextBox ID="txtEmail" runat="server" /></td>
    </tr>
    <tr>
      <td colspan="2" align="center"><asp:Button ID="btnFPNext" runat="server" Text="Next Page" /></td>
    </tr>
  </table>
</asp:View>
<asp:View ID="EDView" runat="server">
  <table align="center">
    <caption>Education Details</caption>
    <tr>
      <td>Graduation:</td>
      <td><asp:TextBox ID="txtGraduation" runat="server" /></td>
    </tr>
    <tr>
      <td>College Name:</td>
      <td><asp:TextBox ID="txtGCN" runat="server" /></td>
    </tr>
    <tr>
      <td>Year of completion:</td>
      <td>
        <asp:TextBox ID="txtGCY" runat="server" />
        <asp:ImageButton ID="imgGCY" runat="server" ImageUrl="~/Icons/Calendar.ico" Width="20"
```

```
                                    Height="20" ImageAlign="AbsMiddle" />
        <asp:Calendar ID="cldGCY" runat="server" />
      </td>
    </tr>
    <tr>
      <td>Post Graduation:</td>
      <td><asp:TextBox ID="txtPG" runat="server" /></td>
    </tr>
    <tr>
      <td>College Name:</td>
      <td><asp:TextBox ID="txtPGCN" runat="server" /></td>
    </tr>
    <tr>
      <td>Year or Completion</td>
      <td>
        <asp:TextBox ID="txtPGCY" runat="server" />
        <asp:ImageButton ID="imgPGCY" runat="server" ImageUrl="~/Icons/Calendar.ico" Width="20"
                                            Height="20" ImageAlign="AbsMiddle" />
        <asp:Calendar ID="cldPGCY" runat="server" />
      </td>
    </tr>
    <tr>
      <td><asp:Button ID="btnSPPrev" runat="server" Text="Prev Page" /></td>
      <td align="right"><asp:Button ID="btnSPNext" runat="server" Text="Next Page" /></td>
    </tr>
  </table>
</asp:View>
<asp:View ID="FDView" runat="server">
  <table align="center">
    <caption>Family Details</caption>
    <tr>
      <td>Spouse Name:</td>
      <td><asp:TextBox ID="txtSpouse" runat="server" /></td>
    </tr>
    <tr>
      <td>Father Name:</td>
      <td><asp:TextBox ID="txtFather" runat="server" /></td>
    </tr>
    <tr>
      <td>Mother Name:</td>
      <td><asp:TextBox ID="txtMother" runat="server" /></td>
    </tr>
    <tr>
      <td>Siblings (if any):</td>
      <td><asp:TextBox ID="txtSiblings" runat="server" /></td>
    </tr>
```

```
            <tr>
              <td>Children (if any):</td>
              <td><asp:TextBox ID="txtChildren" runat="server" /></td>
            </tr>
            <tr>
              <td><asp:Button ID="btnTPPrev" runat="server" Text="Prev Page" /></td>
              <td align="right"><asp:Button ID="btnTPNext" runat="server" Text="Next Page" /></td>
            </tr>
          </table>
        </asp:View>
        <asp:View ID="CDView" runat="server">
          <table align="center">
            <caption>Confirm Details</caption>
            <tr>
              <td colspan="2">
                <asp:CheckBox ID="CheckBox1" runat="server" Text="All information provided here is correct and I
                              agree to face all the consequences if proved the information is wrong." />
                <br />
                <asp:CheckBox ID="CheckBox2" runat="server" Text="Accept terms and condition." />
              </td>
            </tr>
            <tr>
              <td><asp:Button ID="btnLPPrev" runat="server" Text="Prev Page" /></td>
              <td align="right"><asp:Button ID="btnLPConfirm" runat="server" Text="Confirm Details" /></td>
            </tr>
          </table>
        </asp:View>
      </asp:MultiView>
    </ContentTemplate>
</asp:UpdatePanel>
```

Now run the Web Form and watch the output which will display blank screen because by default the "ActiveViewIndex" property of MultiView control is "-1" which will not display any View, so to resolve the problem go to "MultiViewDemo.aspx.cs" file and write the following code:

Code under Page Load Event:

```
if (!IsPostBack) {
  MultiView1.ActiveViewIndex = 0;
}
```

Code under First Page Next Button Click Event:

```
MultiView1.ActiveViewIndex = 1;
```

Code under Second Page Prev Button Click Event:

```
MultiView1.ActiveViewIndex = 0;
```

Code under Second Page Next Button Click Event:

```
MultiView1.ActiveViewIndex = 2;
```

Code under Third Page Prev Button Click Event:

```
MultiView1.ActiveViewIndex = 1;
```

Code under Third Page Next Button Click Event:

```
MultiView1.ActiveViewIndex = 3;
```

Code under Last Page Prev Button Click Event:

```
MultiView1.ActiveViewIndex = 2;
```

Panel Control

This control works as a container for other controls on a Web Page and it takes care of the appearance or visibility of the controls it contains on it, i.e., we use this control to display or hide a group of controls as per a condition. To work with this control, add a new Web Form in the project, naming it as "PanelDemo.aspx" and write the below code under its " " tag:

```
<h1 style="text-align: center; background-color: yellow; color: red; border-style: double; border-color: aqua">
  Life Insurance Corporation of India
</h1>
<div style="text-align: center; background-color: coral">
  Login As:
  <asp:Button ID="btnAgent" runat="server" Text="Agent" Width="100" />
  <asp:Button ID="btnCustomer" runat="server" Text="Customer" Width="100" />
</div>
<br />
<asp:Panel ID="pnlAgent" runat="server">
  <table align="center">
    <caption>Agent Login</caption>
    <tr>
      <td>Agent Id:</td>
      <td><asp:TextBox ID="txtAgentId" runat="server" /></td>
    </tr>
    <tr>
      <td>Password:</td>
      <td><asp:TextBox ID="txtAgentPwd" runat="server" TextMode="Password" /></td>
    </tr>
    <tr>
      <td colspan="2" align="center">
        <asp:Button ID="btnAgentLogin" runat="server" Text="Login" />
        <asp:Button ID="btnAgentReset" runat="server" Text="Reset" />
```

```
      </td>
    </tr>
  </table>
</asp:Panel>
<asp:Panel ID="pnlCustomer" runat="server">
  <table align="center">
    <caption>Customer Login</caption>
    <tr>
      <td>Customer Id:</td>
      <td><asp:TextBox ID="txtCustomerId" runat="server" /></td>
    </tr>
    <tr>
      <td>Password:</td>
      <td><asp:TextBox ID="txtCustomerPwd" runat="server" TextMode="Password" /></td>
    </tr>
    <tr>
      <td colspan="2" align="center">
        <asp:Button ID="btnCustomerLogin" runat="server" Text="Login" Visible="false" />
        <asp:Button ID="btnCustomerReset" runat="server" Text="Reset" Visible="false" />
      </td>
    </tr>
  </table>
</asp:Panel>
```

Now when we run the Web Form it will display both the 2 panels i.e., Agent Login Panel as well as the Customer Login Panel whereas if we want the panels to be visible based of the user selecting the 2 buttons "Agent" or "Customer" write the following code in "PanelDemo.aspx.cs" file:

Code under Agent Button Click:

```
pnlAgent.Visible = true;
pnlCustomer.Visible = false;
```

Code under Customer Button Click:

```
pnlAgent.Visible = false;
pnlCustomer.Visible = true;
```

Validation Controls

Validation is an important part of any web application. Whenever the user gives input in a Web Page, it must always be validated before sending it across various layers of an application. If we get the user input without any validation, then chances are that data might be wrong. So, validation is a good idea to do whenever we are taking input from the user. In Web Application validations are performed in 2 different levels:

1. Client-Side Validation
2. Server-Side Validation

Client-Side Validation: When a validation is done on the client's browser, then it is known as Client-Side validation. We use Java Script to do Client-Side validations. Since many years, developers are using Java Script for Client-Side validations. Client-side validation is very good, but we must be dependent on browser and scripting language support. Client-side validation is considered convenient for users as they get instant feedback. The main advantage is that it prevents a page from being submitted to the server until the client validation is executed successfully.

Server-Side Validation: When a validation occurs on the server, then it is known as Server-Side validation. Server-Side validation is a secure form of validation. The main advantage of Server-Side validation is that if the user somehow bypasses the Client-Side validation by disabling "Java Script" on his browser, we can still catch the problem on Server. Server-Side provides more security and ensures that no invalid data is processed by the application. Server-Side validation is done by writing the custom logic for validating the input, and developer point of view Server-Side is preferable because it will not fail, it is not dependent on browser and scripting languages.

Validation Controls in ASP.NET: To simplify validations in ASP.NET we are provided with validation controls and these controls will perform data validations on the "Client Side" as well as on the "Server Side" i.e., first they will perform client-side validations by using Java Script, and if at all validations fail it will stop the page being submitted to the server, and in-case if the client disables Java Script on his browser then page will be submitted to server even when the validations fail and in such scenarios Validation Controls will re-validate data on the server-side again. So, at any cost server-side validation will work, whether client-side validation is executed or not, so we have safety of validation check.

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 controls known as Validation Controls that provide an easy-to-use and powerful way to check for errors and, if necessary, display messages to the user. The following is the list of Validation Controls we have in ASP.NET:

- RequiredFieldValidator Control
- CompareValidator Control
- RangeValidator Control
- RegularExpressionValidator Control
- CustomValidator Control
- ValidationSummary Control

Validation Controls are classes which are inherited from the class "BaseValidator" hence they inherit its properties, events, and methods. Therefore, it would help us to look at the properties and the methods of that base class, which are common for all the validation controls:

1. **ControlToValidate:** by using this property we need to specify the "Id" of Input Control which must be validated by the Validation Control.
2. **Display:** this is an Enumerated Property using which we can set how the "Error Message" is shown on the browser and it can be set with any of the following values like: Static[d], Dynamic and None.
3. **EnableClientScript:** this is a boolean property which indicates whether client-side validation should take place or not. The default value is true and if set as false then client-side validations will not take place.
4. **Enabled:** this is a boolean property with default value true and if set as false then validator gets disabled i.e., it will not validate the input control.
5. **ErrorMessage:** this property is used to specify an Error Message that must be displayed when the validation fails, and this message is also copied by the "Validation Summary" control.
6. **Text:** this is also like "Error Message" property only, but when both the properties are set with a value, first preference is given to "Text" only which is displayed at the place where validation control is placed whereas Error Message values will be used by the "Validation Summary" control.
7. **SetFocusOnError:** this is a boolean property with default value false and when set as true will set the focus back in to the Input Control which has been validated by the Validation Control if the validation fails.
8. **ValidationGroup:** this property is used to group a set of controls on the page, so that Validations in 1 group will not affect validations of another group while submitting the Page.
9. **IsValid:** this is a boolean property which indicates whether the value of the input control is valid or not.
10. **ForeColor:** this property is to specify the color in which Error Messages should be displayed.

ASP.NET Web Forms have provided Validation Controls since the initial releases. Prior to ASP.NET 4.5 in a normal validation scenario, when we use a validator to validate any control and use client-side validation, some Java Script is generated and rendered as HTML. Also, some supportive Java Script files are also get loaded by the browser for the validation. Unobtrusive Java Script is the way of writing Java Script language in which we properly separate Document Content and Script Content thus allowing us to make a clear distinction between them. This approach is useful in so many ways as it makes our code less error prone, easy to update and debug.

From ASP.NET 4.5 we need to explicitly specify how ASP.NET globally enables the built-in validator controls to use Unobtrusive Java Script for client-side validation logic and we can set it with any of the 2 values like "None" or "Web Forms". If this key's value is set to "None", the ASP.NET application will use the pre - 4.5 behavior (Java Script inline in the pages) for client-side validation logic. If this key's value is set to "Web Forms", ASP.NET

uses Html 5 data-attributes and late bound Java Script from an added script reference for client-side validation logic. We can set the unobtrusive validation mode in Global.asax file or in the individual Web Form or Web.Config.

Option 1: To set Unobtrusive Validation Mode in Global.asax file we need to set the UnobtrusiveValidationMode property of ValidationSettings class in the Application_Start Event Handler by importing the namespace "System.Web.UI" as following:

```
ValidationSettings.UnobtrusiveValidationMode = UnobtrusiveValidationMode.None;
```

Option 2: To set Unobtrusive Validation Mode in a Web Form we need to set the UnobtrusiveValidaionMode property of ValidationSettings class in Page_Load Event Hander of each Web Form as following:

```
this.UnobtrusiveValidationMode = UnobtrusiveValidationMode.None;
```

Option 3: To set unobtrusive validation mode in Web.config file we need to add a new key under <appSettings> tag with the name "ValidationSettings:UnobtrusiveValidationMode" and value as "none" as below. If <appSettings> tag is present, then directly add the following:

```
<add key="ValidationSettings:UnobtrusiveValidationMode" value="none" />
```

If <appSettings> tag is not present, then write it as following:

```
<appSettings>
  <add key="ValidationSettings:UnobtrusiveValidationMode" value="none" />
</appSettings>
```

Working with RequiredFieldValidator: this validation control is used to check whether the input control is left empty or not i.e., it should either be entered with a value or selected with a value. This control has a property which is specific to itself:

1. **InitialValue:** this property is to specify a value, which the Validation Control should not take into consideration while validating the Input Control.

Add a WebForm in the project naming it as "Validations1.aspx" and write the below code under its tag:

```
Enter Name:
<asp:TextBox ID="txtName" runat="server" />
<asp:ReqiredFieldValidator ID="rfvName" runat="server" ControlToValidate="txtName" ForeColor="Red"
        ErrorMessage="Can't leave the field empty." Display="Dynamic" SetFocusOnError="true" /><br />
Select Country:
<asp:DropDownList ID="ddlCountries" runat="server">
  <asp:ListItem Text="-Select Country-" Value="NoCountry" />
  <asp:ListItem Text="India" Value="Country1" />
  <asp:ListItem Text="Japan" Value="Country2" />
  <asp:ListItem Text="China" Value="Country3" />
  <asp:ListItem Text="England" Value="Country4" />
  <asp:ListItem Text="America" Value="Country5" />
  <asp:ListItem Text="Australia" Value="Country6" />
</asp:DropDownList>
<asp:RequiredFieldValidator ID="rfvCountry" runat="server" ControlToValidate="ddlCountries" ForeColor="Red"
ErrorMessage="Please select a country." InitialValue="NoCountry" Display="Dynamic" SetFocusOnError="true" />
<br /><asp:Button ID="btnSubmit" runat="server" Text="Submit Data" />
<br /><asp:Label ID="lblMsg" runat="server" ForeColor="Red" />
```

Now goto "aspx.cs" file and write the below code in it:

Code under Page_Load Event Handler:

```
if (!IsPostBack) {
  txtName.Focus();
}
```

Code under Submit Page Button Click Event Handler:

```
lblMsg.Text = "Hello user the name you entered is: " + txtName.Text + " and selected country is: " +
ddlCountries.SelectedItem.Text;
```

Now run the WebForm and watch the output, where we notice when the 2 input controls are not filled with a value and Submit Button is clicked, page will not be submitted to server whereas when we fill values and click on Submit Button then the page will be submitted to server and displays message under the Label control.

When we implement data validations logic by using JavaScript we will face a problem i.e. if JavaScript is disabled by client on his browser, then page will be submitted to server even when the validations fail and executes the logic on server even if the logic is dependent on the validations. To test this disable "JavaScript" on your browser and test the page again so that now even if we did not fill the values in input controls and click on the Submit Button will submit page to server and displays the previous message in Label Control.

Whereas the advantage with our Validation Controls is even if JavaScript is disabled and page is submitted to server, still validation controls will re-validate input controls on the server and sets the Validation Controls "IsValid" property with a Boolean value which is true when the validation is success and false if the validation fails. So we can implement logic under the server by checking the condition as following:

```
if (rfvName.IsValid && rfvCountry.IsValid) {
  lblMsg.ForeColor = System.Drawing.Color.Green;
  lblMsg.Text = "Hello user the name you have entered is: " + txtName.Text + " and your selected country is: " +
                                                ddlCountries.SelectedItem.Text;
}
else {
  lblMsg.ForeColor = System.Drawing.Color.Red;
  lblMsg.Text = "Data validations failed.";
}
```

Note: the problem we face with above code is when we have multiple validations controls, checking each Validation Controls "IsValid" value will become complicated and we can avoid this by using Page Classes "IsValid" property because internally each validation control will first perform data validations and sets their "IsValid" property value as true or false based on success or failure of the validations and finally sets the Page Class "IsValid" property as true if all the Validation Controls "IsValid" value is true, whereas if any Validation Controls "IsValid" property is false then Page Class "IsValid" property value also will be false. We can simplify the "if condition" in our previous code as following: if (rfvName.IsValid && rfvCountry.IsValid) => if (IsValid)

Currently validations are getting performed when we click on the Submit Button, whereas if we want the validations to be performed when the focus is leaving the control add the following code in "Page_Load" Event Handler of the if condition:

```
txtName.Attributes.Add("onblur", "ValidatorValidate(rfvName)");
ddlCountries.Attributes.Add("onblur", "ValidatorValidate(rfvCountry)");
```

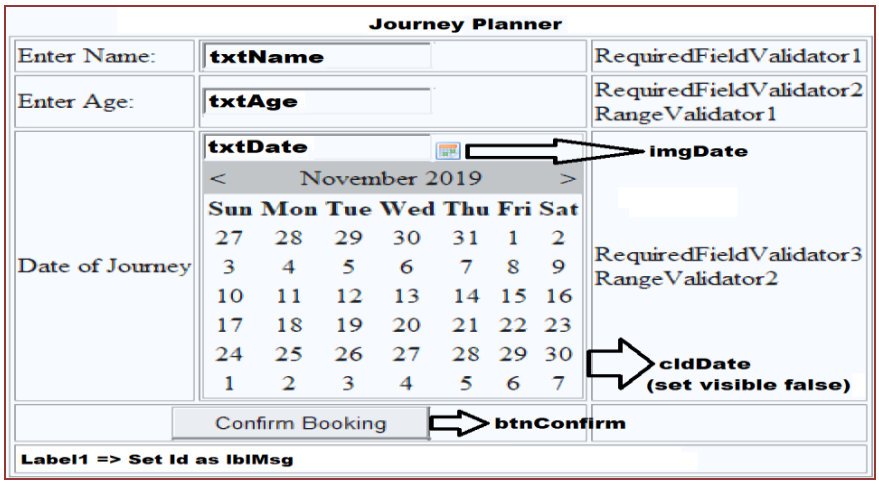
The above statements will bind the “Validation Controls”, “JavaScript” function to the controls “onblur” event so that Event get fired when the focus is leaving the controls.

RangeValidator: this validation control is used to check whether the value entered into an input control is with-in a specifed range or not. This control is associated with these properties in specific:

1. **MinimumValue:** this is to specify the least value that can be entered into the input control being validated.
2. **MaximumValue:** this is to specify the highest value that can be entered into the input control being validated.
3. **Type:** this is to specify the DataType for the values which are being compared and can be set with any of the following values: String [d], Interger, Double, Date and Currency.

Note: none of the ValidationControls except RequiredFieldValidator can check for empty values, so while using other validation controls if we want to perform an empty value check we need to use a RequiredFieldValidator control also along with other validation controls. 1 validation control can be associated with 1 and only 1 input control whereas 1 input control can be associated with any no. of validation controls.

Add a new WebForm in our project naming it as “Validations2.aspx” and design it as following:



To design the above write the below code under tag of the “Validations2.aspx” file:

```
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<table align="center">
  <caption>Journey Planner</caption>
  <tr>
    <td>Enter Name:</td>
    <td><asp:TextBox ID="txtName" runat="server" /></td>
    <td><asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server" /></td>
  </tr>
  <tr>
    <td>Enter Age:</td>
```

```
    <td><asp:TextBox ID="txtAge" runat="server" /></td>
    <td><asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server" /><br />
        <asp:RangeValidator ID="RangeValidator1" runat="server" /></td>
  </tr>
  <tr>
    <td>Date of Journey:</td>
    <td><asp:UpdatePanel ID="UpdatePanel1" runat="server">
          <ContentTemplate>
            <asp:TextBox ID="txtDate" runat="server" />
            <asp:ImageButton ID="imgDate" runat="server" ImageUrl="~/Icons/Calendar.ico" Width="20"
                                    Height="20" ImageAlign="AbsMiddle" />
            <asp:Calendar ID="cldDate" runat="server" Visible="false" />
          </ContentTemplate>
        </asp:UpdatePanel></td>
    <td><asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server" /><br />
        <asp:RangeValidator ID="RangeValidator2" runat="server" /></td>
  </tr>
  <tr>
    <td colspan="2" align="center"><asp:Button ID="btnConfirm" runat="server" Text="Confirm Booking" /></td>
    <td></td>
  </tr>
  <tr>
    <td colspan="3"><asp:Label ID="lblMsg" runat="server" /></td>
  </tr>
</table>
```

Now in the design view select all the 5 Validation Controls at a time, hit "F4" and set the following 3 properties in property window:

Display: Dynamic **ForeColor:** Red **SetFocusOnError:** True

Now go to properties of each Validation Control and set the following properties:

RequiredFieldValidator1:

Id: rfvName **ControlToValidate:** txtName **ErrorMessage:** Name field can't be left empty.

RequiredFieldValidator2:

Id: rfvAge **ControlToValidate:** txtAge **ErrorMessage:** Age field can't be left empty.

RequiredFieldValidator3:

Id: rfvDate **ControlToValidate:** txtDate **ErrorMessage:** Journey date can't be left empty.

RangeValidator1:

Id: rvAge **ControlToValidate:** txtAge **MinimumValue:** 18 **MaximumValue:** 65
ErrorMessage: Traveller's age should be between 18 - 65 years. **Type:** Integer

RangeValidator2:

Id: rvDate **ControlToValidate:** txtDate **Type:** Date

ErrorMessage: Travel date should be with-in 90 days from current date.

Now go to "Validations2.aspx.cs" file and write the following code:

Code under Page_Load:

```
rvDate.MinimumValue = DateTime.Now.ToShortDateString();
rvDate.MaximumValue = DateTime.Now.AddDays(90).ToShortDateString();
if(!IsPostBack) {
  txtName.Focus();
}
```

Code under Image Button Click:

```
if(cldDate.Visible) {
  cldDate.Visible = false;
}
else {
  cldDate.Visible = true;
}
```

Code under Calendar Selection Changed:

```
txtDate.Text = cldDate.SelectedDate.ToShortDateString();
cldDate.Visible = false;
```

Code under Confirm Button Click:

```
if(IsValid) {
  lblMsg.ForeColor = System.Drawing.Color.Green;
  lblMsg.Text = "Your booking is confirmed.";
}
else {
  lblMsg.ForeColor = System.Drawing.Color.Red;
  lblMsg.Text = "Validations failed please re-check your data.";
}
```

Note: When we run the WebForm we face a problem i.e. when we click on the ImageButton to launch the Calender the ImageButton requires to perform a CallBack, but the Validation Control will not allow that to happen until all the Validations are successful so to avoid that problem use ValidationGroup property of ImageButton and set it with some value, for example: "DateGroup", so that when we click on ImageButton it will validate only the controls which comes under this group and currently we don't have any other control coming under this group.

CompareValidator: this control is used for performing 3 types of validations, those are:

1. Compare the value of an Input Control with another Input Control.
2. Compare the value of an Input Control with a fixed given value.
3. Data Type Check.

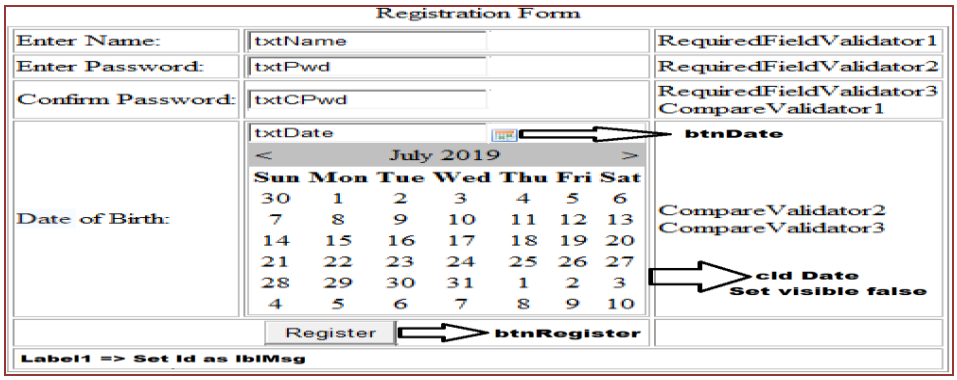
Properties specific to CompareValidator:

1. **ControlToCompare:** this property is to specify the Id of an Input Control with whom ControlToValidate has to be compared.
2. **ValueToCompare:** this property is to specify a value with what ControlToValidate must be compared.

3. **Operator:** this property is to specify the type of comparison that must be performed, and it can be set with any of the following values: Equal [d], NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual and DataTypeCheck.

4. **Type:** this is to specify the DataType for the values which are being compared and can be set with any of the following values: String [d], Integer, Double, Date and Currency.

Add a new WebForm in our project naming it as "Validations3.aspx" and design it as following:



To design the above write the below code under tag of the "Validations2.aspx" file:

```
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<table align="center">
  <caption>Registration Form</caption>
  <tr>
    <td>Enter Name:</td>
    <td><asp:TextBox ID="txtName" runat="server" /></td>
    <td><asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server" /></td>
  </tr>
  <tr>
    <td>Enter Password</td>
    <td><asp:TextBox ID="txtPwd" runat="server" TextMode="Password" /></td>
    <td><asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server" /></td>
  </tr>
  <tr>
    <td>Confirm Password:</td>
    <td><asp:TextBox ID="txtCPwd" runat="server" TextMode="Password" /></td>
    <td><asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server" /><br />
        <asp:CompareValidator ID="CompareValidator1" runat="server" /></td>
  </tr>
  <tr>
    <td>Date of Birth:</td>
    <td><asp:UpdatePanel ID="UpdatePanel1" runat="server">
          <ContentTemplate>
            <asp:TextBox ID="txtDate" runat="server" />
            <asp:ImageButton ID="imgDate" runat="server" ImageUrl="~/Icons/Calendar.ico" Width="20"
                Height="20" ImageAlign="AbsMiddle" ValidationGroup="DateGroup" />
            <asp:Calendar ID="cldDate" runat="server" Visible="false" />
          </ContentTemplate>
```

```
        </asp:UpdatePanel></td>
    <td>
      <asp:CompareValidator ID="CompareValidator2" runat="server" />
      <br />
      <asp:CompareValidator ID="CompareValidator3" runat="server" />
    </td>
  </tr>
  <tr>
    <td colspan="2" align="center">
      <asp:Button ID="btnRegister" runat="server" Text="Register" />
    </td>
    <td> </td>
  </tr>
  <tr>
    <td colspan="3"><asp:Label ID="lblMsg" runat="server" /></td>
  </tr>
</table>
```

Now select all the 6 Validation Controls at a time, hit "F4" and set the following properties in property window:

Display: Dynamic **ForeColor:** Red **SetFocusOnError:** True

Now go to properties of each Validation Control and set the following properties:

RequiredFieldValidator1:

Id: rfvName **ControlToValidate:** txtName **ErrorMessage:** Name field can't be left empty.

RequiredFieldValidator2:

Id: rfvPwd **ControlToValidate:** txtPwd **ErrorMessage:** Password field can't be left empty.

RequiredFieldValidator3:

Id: rfvCPwd **ControlToValidate:** txtCPwd **ErrorMessage:** Confirm password field can't be left empty.

CompareValidator1:

Id: cvCPwd **ControlToValidate:** txtCPwd **ControlToCompare:** txtPwd **Type:** String
ErrorMessage: Confirm password should match with password. **Operator:** Equal

CompareValidator2:

Id: cvDate1 **ControlToValidate:** txtDate **Operator:** DataTypeCheck
ErrorMessage: Entered value is not in a valid date format. **Type:** Date

CompareValidator3:

Id: cvDate2 **ControlToValidate:** txtDate **Operator:** LessThanEqual
ErrorMessage: You need to attain 18 years of age for registration. **Type:** Date

Now goto "Validations3.aspx.cs" file and write the following code:

Code under Page_Load:

```
if(!IsPostBack) {
  txtName.Focus();
```

```
}
cvMajor.ValueToCompare = DateTime.Now.AddYears(-18).ToShortDateString();
```

Code under Image Button Click:

```
if (cldDate.Visible) {
  cldDate.Visible = false;
}
else {
  cldDate.Visible = true;
}
```

Code under Calendar Selection Changed:

```
txtDate.Text = cldDate.SelectedDate.ToShortDateString();
cldDate.Visible = false;
```

Code under Register Button Click:

```
if (IsValid) {
  lblMsg.ForeColor = System.Drawing.Color.Green;
  lblMsg.Text = "Your registration is successful.";
}
else {
  lblMsg.ForeColor = System.Drawing.Color.Red;
  lblMsg.Text = "Validations failed please re-check your data.";
}
```

RegularExpressionValidator: This control will validate input by using some special characters known as Regular Expressions or Regex and these are some special characters using which we can perform data validations without writing complex logic.

B => Braces => [] {} () C => Carrot => ^ D => Dollar => $

Braces:

[] => these are used to specify the characters which are allowed.
E.g: [a-m] or [A-K] or [0-6] or [A-Za-z0-9]

{} => these are used to specify the no. of characters that are allowed.
E.g: {1} or {4, 7} or {1,}

() => these are used to specify a list of options which are accepted.
E.g: (com|net|in|edu)

[A-K] => accepts 1 alphabet between A to K.
[a-m]{5} => accepts 5 alphabets between a to m.
[a-z]{3}[0-9]{5} => accepts 3 alphabets in the first followed by 5 numeric.
Note: in all the above 3 cases after the validating expression, it will accept anything we enter over there and to overcome this problem we need to make the expressions rigid as following:
^[a-z]{3}[0-9]{5}$ => same as the last expression above but this is rigid expression i.e., accepts only 8 chars.

Special characters in regular expressions: there are some special characters with pre-defined logic.

\s => accepts whitespace
\S => doesn't accept whitespace
\d => accepts only numeric values.
\D => accepts only non-numeric values.
\w => accepts only alpha-numeric values.
\W => accepts only non-alphanumeric values.

Validation Expressions using Regular Expressions:

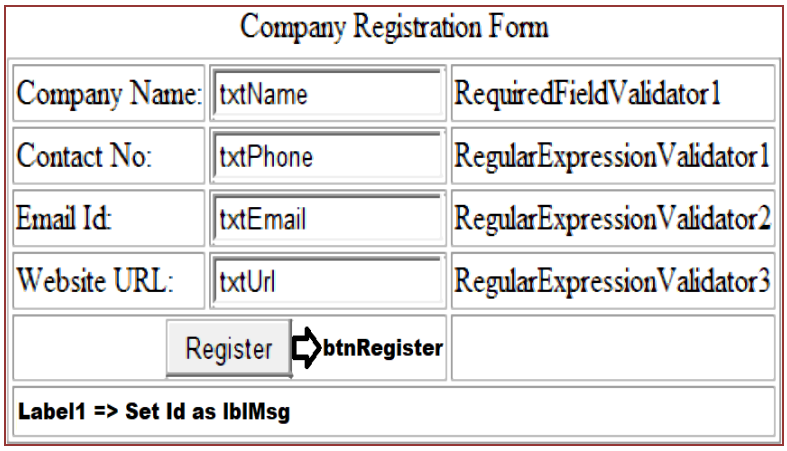
\d{6}	=> Indian Postal Code
\d{3}-\d{7}	=> Indian Railway PNR
\d{4} \d{4} \d{4}	=> Aadhar Number
\d{4} \d{4} \d{4} \d{4}	=> Credit Card Number
http(s)?://([\w-]+\.)+[\w-]+(/[\w- ./?%&=]*)?	=> Website URL Validation.
\w+[\w-\.]*\@\w+((-\w+)|(\w*))\.[a-z]{2,3}	=> Email Validation
[6-9]\d{9}	=> Mobile No. validation that checks No. starts with 6 or 7 or 8 or 9 and will be maximum 10 and minimum 10 digits.
[0][6-9]\d{9}	=> Mobile No. validation that checks No. starts with 0 and after that 6 or 7 or 8 or 9 and will be maximum 11 and minimum 11 digits.
^[0][6-9]\d{9}$\|^[6-9]\d{9}$	=> Mobile No. validation which checks if the No. starts with 0 then it is 11 digit or else it is 10 digit.
^\d{6,8}$\|^[6-9]\d{9}$	=> Validation for either 6-8 digits landline no or 10 digit mobile no that starts with 6 or 7 or 8 or 9.

Note: We can use Regular Expression in ASP.NET by using the RegularExpressionValidator control and specify the Regular Expression under its "ValidationExpression" property.

Properties specific to RegularExpressionValidator:

1. **ValidationExpression:** by using this property we need to specify the "ValidationExpression" for the validation, either by choosing from a list of pre-defined expressions and this can be done by clicking on the button beside the property which will launch "Regular Expression Editor" and it lists a set of pre-defined expressions for Email Id Validation, Website Url Validation, etc or else default value will be Custom using which we can enter or specify our own Validation Expression.

Add a new WebForm in our project naming it as "Validations4.aspx" and design it as following:




```
<table align="center">
  <caption>Company Registration Form</caption>
  <tr>
    <td>Company Name:</td>
    <td><asp:TextBox ID="txtName" runat="server" /></td>
    <td><asp:RequiredFieldValidator ID="rfvName" runat="server" /></td>
  </tr>
  <tr>
    <td>Contact No:</td>
    <td><asp:TextBox ID="txtPhone" runat="server" /></td>
    <td><asp:RegularExpressionValidator ID="revPhone" runat="server" /></td>
  </tr>
  <tr>
    <td>Email Id:</td>
    <td><asp:TextBox ID="txtEmail" runat="server" /></td>
    <td><asp:RegularExpressionValidator ID="revEmail" runat="server" /></td>
  </tr>
  <tr>
    <td>Website Url:</td>
    <td><asp:TextBox ID="txtUrl" runat="server" /></td>
    <td><asp:RegularExpressionValidator ID="revUrl" runat="server" /></td>
  </tr>
  <tr>
    <td colspan="2" align="center"><asp:Button ID="btnRegister" runat="server" Text="Register" /></td>
    <td> </td>
  </tr>
  <tr>
    <td colspan="3"><asp:Label ID="lblMsg" runat="server" /></td>
  </tr>
</table>
```

Now select all the 4 Validation Controls at a time, hit "F4" and set the following properties in property window:

Display: Dynamic ForeColor: Red SetFocusOnError: True

Now go to properties of each Validation Control and set the following properties:

RequiredFieldValidator1:

Id: rfvName ControlToValidate: txtName ErrorMessage: Comapany name field can't be left empty.

RegularExpressionValidator1:

Id: revPhone ControlToValidate: txtPhone ValidationExpression: ^\d{6,8}$|^[6-9]\d{9}$
ErrorMessage: Contact no. can either be Land Phone (6-8 digits) or Mobile (10 digits).

RegularExpressionValidator2:

Id: revEmail ControlToValidate: txtEmail ErrorMessage: Given input is an invalid Email Id format.
ValidationExpression: Click on the button beside and select "Internet e-mail Address" in Regular Expression Editor.

RegularExpressionValidator3:

Id: revUrl ControlToValidate: txtUrl ErrorMessage: Given input is an invalid Website Url format.
ValidationExpression: Click on the button beside and select "Internet URL" in Regular Expression Editor.

Now go to aspx.cs file and write the following code:

Code under Page_Load:

```
if(!IsPostBack) {
  txtName.Focus();
}
```

Code under Register Button Click:

```
if (IsValid) {
  lblMsg.ForeColor = System.Drawing.Color.Green;
  lblMsg.Text = "Your company registration is successful.";
}
else {
  lblMsg.ForeColor = System.Drawing.Color.Red;
  lblMsg.Text = "Your company registration failed because validations failed.";
}
```

CustomValidator: this control doesn't have any logic to perform Data Validations i.e. we need to implement all the logic on our own jusk like implementing manual JavaScript code but the advantage with this is we can implement logic to perform Client Side Validation by using Java Script code as well as we can also implement logic to re-validate the data by using C# code on Server Side, so that in case JavaScript is disabled at Client's Browser still data gets validated on the Server.

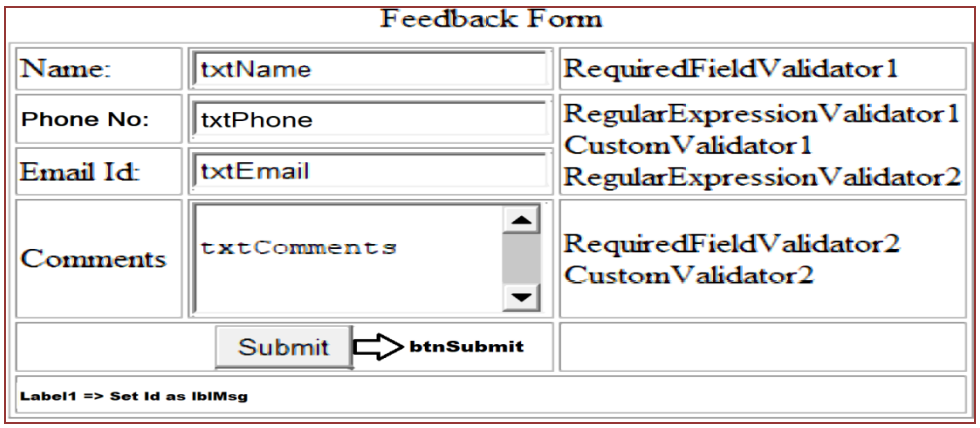
Note: We don't require (optional) setting "ControlToValidate" property for CustomValidator so this Validator is capable of validating more than 1 Input Control also.

Member's specifc to CustomValidator:

1. **ClientValidationFunction:** by using this property we need to specify the name of JavaScript function in which we have implemented logic for Client-Side Validation.

2. **ServerValidate:** this is an Event under which we need implement the logic to perform Server Side Validation.

Add a new WebForm in our project naming it as "Validations5.aspx" and design it as following:




```
<table align="center">
  <caption>Feedback Form</caption>
  <tr>
    <td>Name:</td>
    <td><asp:TextBox ID="txtName" runat="server" /></td>
    <td><asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server" /></td>
  </tr>
  <tr>
    <td>Phone No:</td>
    <td><asp:TextBox ID="txtPhone" runat="server" /></td>
    <td rowspan="2"><asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server" /><br />
                    <asp:CustomValidator ID="CustomValidator1" runat="server"  /><br />
                    <asp:RegularExpressionValidator ID="RegularExpressionValidator2" runat="server" /></td>
  </tr>
  <tr>
    <td>Email Id:</td>
    <td><asp:TextBox ID="txtEmail" runat="server" /></td>
  </tr>
  <tr>
    <td>Comments:</td>
    <td><asp:TextBox ID="txtCommants" runat="server" Rows="3" TextMode="MultiLine" /></td>
    <td><asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server"  /><br />
        <asp:CustomValidator ID="CustomValidator2" runat="server" /></td>
  </tr>
  <tr>
    <td colspan="2" align="center"><asp:Button ID="btnSubmit" runat="server" Text="Submit" /></td>
    <td> </td>
```

```
  </tr>
  <tr>
    <td colspan="3"><asp:Label ID="lblMsg" runat="server" /></td>
  </tr>
</table>
```

Now select all the 6 Validation Controls at a time, hit "F4" and set the following properties in property window:

Display: Dynamic ForeColor: Red SetFocusOnError: True

Now go to properties of each Validation Control and set the following properties:

RequiredFieldValidator1:

Id: rfvName ControlToValidate: txtName ErrorMessage: Name field can't be left empty.

RequiredFieldValidator2:

Id: rfvComments ControlToValidate: txtComments ErrorMessage: Comments field can't be left empty.

RegularExpressionValidator1:

Id: revPhone ControlToValidate: txtPhone ValidationExpression: ^\d{8}$|^[6-9]\d{9}$

ErrorMessage: Phone no. can either be landline (8 digits) or mobile (10 digits).

RegularExpressionValidator2:

Id: revEmail ControlToValidate: txtEmail ErrorMessage: Given input is an invalid Email Id format.

ValidationExpression: Click on the button beside and select "Internet e-mail Address" in Regular Expression Editor.

CustomValidator1:

Id: cvPhoneOrEmail ErrorMessage: Either Phone No. or Email Id must be provided.

ClientValidationFunction: PhoneOrEmail

Now go to events of CustomValidator1 and double click on "ServerValidate" which will generate an Event Handler (cvPhoneOrEmail_ServerValidate) for implementing server-side validation code.

CustomValidator2:

Id: cvComments ControlToValidate: txtComments ClientValidationFunction: Check50Chars

ErrorMessage: Comments should be minimum 50 chars of length.

Now go to events of CustomValidator2 and double click on "ServerValidate" which will generate an Event Handler (cvComments_ServerValidate) for implementing server-side validation code.

Now go to "Source View" and write the following code inside of <head> section:

```
<script>
  function PhoneOrEmail(source, args) {
    if (txtPhone.value.trim().length == 0 && txtEmail.value.trim().length == 0) {
      args.IsValid = false;
    }
    else {
      args.IsValid = true;
    }
```

```
  }
  function Check50Chars(source, args) {
    if (args.Value.trim().length < 50) {
      args.IsValid = false;
    }
    else {
      args.IsValid = true;
    }
  }
</script>
```

Now go to "aspx.cs" file and write the following code:

Code under Page_Load:

```
if(!IsPostBack) {
  txtName.Focus();
}
```

Code under cvPhoneOrEmail_ServerValidate:

```
if (txtPhone.Text.Trim().Length == 0 && txtEmail.Text.Trim().Length == 0) {
  args.IsValid = false;
}
else {
  args.IsValid = true;
}
```

Code under cvComments_ServerValidate:

```
if (args.Value.Trim().Length < 50) {
  args.IsValid = false;
}
else {
  args.IsValid = true;
}
```

Code under Submit Button Click:

```
if (IsValid) {
  lblMsg.ForeColor = System.Drawing.Color.Green;
  lblMsg.Text = "Your feebback is recieved, we will contact you asap.";
}
else {
  lblMsg.ForeColor = System.Drawing.Color.Red;
  lblMsg.Text = "Your company registration failed because validations failed.";
}
```

ValidationSummary: this control doesn't perform any Data Validations but used only for displaying Error Messages i.e., without displaying Error Messages beside the Input Controls we can display all the Error Messages at one location either on top of the Page or bottom of the Page. When we place this Control on a Page it will

automatically inherit the Error Messages that are associated with all Validation Controls that are present on that Page. To test workings with ValidationSummary Control in our current page i.e., "Validations5.aspx", go to desing view and, drag and drop the ValidationSummary Control just below the table and set the ForeColor property value as "Red".

Now run the Web Page and watch the output i.e., if any Validations fail all those Error Messages will be displayed by the ValidationSummary Control in the place where we placed it because this Control will inherit all the Error Messages of the 6 Validation Controls present on the page. But in this case, we also see the Error Messages beside the Input Controls and to avoid them set the Text Property of Validation Controls so that Text value will be displayed beside Input Controls and Error Messages will be displayed by the ValidationSummary control and to test this go to the properties of all 6 Validation Controls on the Page and under the Text enter the value as "*". Now run and watch the difference in output which will display "*" beside the Input Control where the validation failed, and all the Error Messages will be displayed by ValidationSummary control.

Properties specific to ValidationSummary Control:

1. **ShowSummary:** this is a boolean property with the default value true and if set as false will not display the ErrorMessage on Screen.

2. **ShowMessageBox:** this is a boolean property with the default value false and if set as true will display the Error Messages in MessageBox.

Note: set either of the above 2 properties as true so that Error Messages will be displayed either on the Screen or in a MessageBox.

3. **ShowValidationErrors:** this is a boolean property with the default value as true and if set as false Error Messages will not be displayed either on the Screen or in a Message Box.

4. **DisplayMode:** this property is to specify the format how Error Messages must be displayed, and we can choose from any of the below 3 values: BulletedList [d], List and SingleParagraph.

Controls Development

Till now we have been consuming the Controls that are provided by Microsoft and sometimes the available controls may not suit our requirements, so in that situation ASP.NET provides an option of developing our own controls as per our requirements and we call them as User Controls and Custom Controls.

People working with Controls are categorized as Component Developers and Application Developers i.e., the one who develops controls are known as Component Developers and the one who develops applications by making use of the controls are known as Application Developers, and we are application developers because we are developing applications with the help of controls provided by Microsoft.

Note: Sometimes an Application Developer can also become a Component Developer and develop controls, either to customize the existing controls or when the existing controls are not as per the requirements.

To develop Controls in Asp.Net Web Forms, it provides us 2 options:

1. Web Forms User Controls (User Controls)
2. Web Forms Server Controls (Custom Controls)

User Controls	Custom Controls
Designed for single-application scenarios. Deployed in the source form (.ascx) along with the source code of the application. If the same control needs to be used in more than one application, it introduces redundancy and maintenance problems.	Designed so that it can be used by more than one application. Deployed either in the application's Bin directory or in the GAC. Distributed easily and without problems associated with redundancy and maintenance.
Development is like the way Web Forms are developed; well-suited for rapid application development (RAD).	Development involves lots of code because there is no designer support.
A much better choice when you need static content within a fixed layout, for example, when you make headers and footers.	More suitable when an application requires dynamic content to be displayed; can be reused across an application, for example, for a data bound table control with dynamic rows.
Development doesn't require much application designing because they are authored at design time and mostly contain static data.	Development is done from the scratch, requires a good understanding of the control's life cycle and the order in which events execute, which is normally not taken care in user controls.

Developing a User Control:

To develop a User Control we are provided with an item template in the "Add New Item" window with the name "Web Forms User Control", which adds a new item in the project with ".ascx" extension and here also we find "Source View", "Design View" and "Code View" in ".ascx.cs" file, but this class will inherit from UserControl class which is present in System.Web.UI namespace only.

Developing a Customized Calendar Control as User Control:

Step 1: Open the "Add New Item", select "Web Forms" in LHS and now on the RHS we find "Web Forms User Control", select it, name it as "UserCalendar.ascx" and click on "Add" button. By default, the ".ascx" file does not contain any html code in it and also in the place of "Page Directive" we find "Control Directive" but all the attributes will be same as attributes of "Page Directive".

Step 2: now write the following code in ".ascx" file below the "Control Directive":

```
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
  <ContentTemplate>
    <asp:TextBox ID="txtDate" runat="server" />
    <asp:ImageButton ID="imgDate" runat="server" ImageUrl="~/Icons/Calendar.ico" Width="20" Height="20"
                                                ImageAlign="AbsMiddle" />
    <asp:Calendar ID="cldDate" runat="server" Visible="false" NextMonthText="" FirstDayOfWeek="Monday" />
  </ContentTemplate>
</asp:UpdatePanel>
```

Step 3: now go to "ascx.cs" file and there we find a class with the name "UserCalendar" inheriting from the class "UserControl", write the following code in it:

Code under Image Button Click Event:

```
if (cldDate.Visible) { cldDate.Visible = false; }
```

```
else { cldDate.Visible = true; }
```

Code under Calendar SelectionChanged Event:

```
txtDate.Text = cldDate.SelectedDate.ToShortDateString(); cldDate.Visible = false;
```

Code under Calendar DayRender Event:

```
if (e.Day.Date > DateTime.Now) {
  e.Cell.ForeColor = Color.Gray; e.Day.IsSelectable = false; e.Cell.ToolTip = "Out of Range";
}
```

Code under Calendar VisibleMonthChanged Event:

```
if (e.NewDate.Month == DateTime.Now.Month && e.NewDate.Year == DateTime.Now.Year) {
  cldDate.NextMonthText = "";
}
else {
  cldDate.NextMonthText = "&gt;";
}
```

```
//Defining a property for setting a Date or getting the SelectedDate from UserCalendar
public DateTime CalendarSelectedDate {
  get { return cldDate.SelectedDate; }
  set { cldDate.SelectedDate = value; txtDate.Text = value.ToShortDateString(); }
}
```

```
//Defining a method to clear the Text of TextBox and reset the Date to current date
public void Reset() {
  txtDate.Text = ""; cldDate.SelectedDates.Clear(); cldDate.VisibleDate = DateTime.Now;
}
```

Step 4: Now we can consume the UserCalendar in any WebForm of our project and to do that we need to register the control in our project which can be done in 2 different levels i.e., Page Level and Application Level.

Registering the User Control Page Level:

In this approach we register the User Control in the WebForm where we want to consume it, but the drawback is if we want to consume the control in multiple WebForms, we need to register the control in every WebForm we want to consume it.

<%@ Register Src="<name of the .ascx file>" TagName="<a name with what we want to refer the control>" TagPrefix="<some name which we want to use as a prefix>" %>

- Src attribute is to specify the path of our ".ascx" file under which we developed the control.
- TagName is to specify the name with what we want to refer to the control and general we use our original class name only as TagName but can be changed to any value.
- TagPrefix is to specify a prefix value we want to use for our User Control just like we use "asp" as a TagPrefix for all our pre-defined Asp.Net controls.

To test this, add a new WebForm under the project naming it as "TestUserCalendar.aspx" and write the following code below Page Directive:

<%@ Register Src="~/UserCalendar.ascx" TagName="UserCalendar" TagPrefix="NIT1" %>

Now write the following code under tag:

```
<h2 style="background-color:yellow;color:red;text-align:center">Application for Date of Birth Certificate</h2>
<table align="center">
 <caption>Applicant Details</caption>
 <tr> <td>Name:</td> <td><asp:TextBox ID="txtName" runat="server" /></td> </tr>
 <tr> <td>Gender:</td>
  <td>
   <asp:RadioButton ID="rbMale" runat="server" Text="Male" GroupName="Gender" />
   <asp:RadioButton ID="rbFemale" runat="server" Text="Female" GroupName="Gender" />
  </td>
 </tr>
 <tr> <td>Date of Birth:</td> <td><NIT1:UserCalendar ID="ucDate" runat="server" /></td> </tr>
 <tr> <td>Mobile No:</td> <td><asp:TextBox ID="txtMobile" runat="server" /></td> </tr>
 <tr> <td>Address:</td> <td> <asp:TextBox ID="txtAddress" runat="server" TextMode="MultiLine" /> </td> </tr>
 <tr> <td colspan="2" align="center">
   <asp:Button ID="btnSubmit" runat="server" Text="Submit Data" Width="100" />
   <asp:Button ID="btnClear" runat="server" Text="Reset Form" Width="100" />
  </td>
 </tr>
</table>
```

Now goto TestUserCalendar.aspx.cs file and write the following code:

Code under Page_Load:

```
if (!IsPostBack) { txtName.Focus(); }
```

Code under Submit Data Button:

```
Response.Write("<script>alert('" + txtName.Text + "')</script>");
if (rbMale.Checked) { Response.Write("<script>alert('Applicant is male')</script>"); }
else if (rbFemale.Checked) { Response.Write("<script>alert('Applicant is female')</script>"); }
Response.Write("<script>alert('" + ucDate.CalendarSelectedDate.ToShortDateString() + "')</script>");
Response.Write("<script>alert('" + txtMobile.Text + "')</script>");
Response.Write("<script>alert('" + txtAddress.Text + "')</script>");
```

Code under Reset Form Button:

```
txtName.Text = txtAddress.Text = txtMobile.Text = ""; rbMale.Checked = rbFemale.Checked = false;
ucDate.Reset(); txtName.Focus();
```

Registering the User Control Application Level:

In this approach we register the User Control under the project in Web.config file and the advantage with this is we can consume the Control in all the Web Forms of the project with out registering it again and again.

Note: To register a User Control in application level we have a restriction i.e. the User Control and Web Form can't be present in the same folder, so add a new Folder under the project naming it as "UserControls" and drag the "UserCalendar.ascx" into the "UserControls" folder so that the WebForm is in the root folder of our project and UserControl is under "UserControls" folder which is present under the root folder.

To test the above first delete or comment the "Register Directive" we have added in "TestUserCalendar.aspx", then open Web.config file and write the following code inside <system.web> tag:

```
<pages>
  <controls>
    <add src="~/UserControls/UserCalendar.ascx" tagPrefix="NIT1" tagName="UserCalendar"/>
  </controls>
</pages>
```

Developing a Custom Control:

Open a new empty Asp.Net Web Application project naming it as "AspControls", open the "Add New Item" windows, select "Web Forms" in LHS and now on the RHS we find "Web Forms Server Control" select it, name it as "CustomCalendar.cs" and click on "Add" button. This will define a class with the name "CustomCalendar" inheriting for "WebControl" class of "System.Web.UI.WebControls" namespace and we need to write all the code for designing and executing in this class only. Now if we look into the code we will find 2 attributes on top of the class "DefaultProperty" and "ToolBoxData".

- DefaultProperty specifies the default property for this control and it's "Text"
- ToolBoxData property is to specify the information about the code that should be generated when we drag and drop the control from "ToolBox" on to a WebForm and right now the data will be as following:

[ToolboxData("<{0}:CustomCalendar runat=server></{0}:CustomCalendar>")]

In place of "{0}" the TagPrefix what we provide while registering the control will come and "CustomCalendar" is the name of Control which is nothing but our class name, which we can either leave the same or change it as per our requirements. Now delete all the existing code in the class and write our code inside the class which should finally look as following:

```
public class CustomCalendar : WebControl {
  TextBox txtDate; ImageButton imgDate; Calendar cldDate;
  protected override void CreateChildControls() {
    txtDate = new TextBox(); txtDate.ID = "txtDate"; txtDate.Width = Unit.Pixel(200);
    imgDate = new ImageButton(); imgDate.ID = "imgDate"; imgDate.Click += ImgDate_Click;
    cldDate = new Calendar(); cldDate.ID = "cldDate"; cldDate.Visible = false; cldDate.NextMonthText = "";
    cldDate.FirstDayOfWeek = FirstDayOfWeek.Monday; cldDate.SelectionChanged += CldDate_SelectionChanged;
    cldDate.DayRender += CldDate_DayRender; cldDate.VisibleMonthChanged += CldDate_VisibleMonthChanged;
    this.Controls.Add(txtDate); this.Controls.Add(imgDate); this.Controls.Add(cldDate);
  }
  private void ImgDate_Click(object sender, ImageClickEventArgs e) {
    if (cldDate.Visible) { cldDate.Visible = false; }
    else { cldDate.Visible = true; }
  }
  private void CldDate_SelectionChanged(object sender, EventArgs e) {
    txtDate.Text = cldDate.SelectedDate.ToShortDateString(); cldDate.Visible = false;
  }
  private void CldDate_DayRender(object sender, DayRenderEventArgs e) {
    if ((e.Day.Date.DayOfYear > DateTime.Now.DayOfYear && e.Day.Date.Year == DateTime.Now.Year)
          || e.Day.Date.Year > DateTime.Now.Year) {
      e.Cell.ForeColor = System.Drawing.Color.Gray;
      e.Day.IsSelectable = false; e.Cell.ToolTip = "Out Of Range";
    }
  }
```

```
 private void CldDate_VisibleMonthChanged(object sender, MonthChangedEventArgs e) {
  if (e.NewDate.Month == DateTime.Now.Month && e.NewDate.Year == DateTime.Now.Year) {
   cldDate.NextMonthText = "";
  }
  else { cldDate.NextMonthText = "&gt;"; }
 }
 public DateTime CalendarSelectedDate {
  get { EnsureChildControls(); return cldDate.SelectedDate; }
  set { EnsureChildControls(); cldDate.SelectedDate = value; txtDate.Text = value.ToShortDateString(); }
 }
 public string ButtonImageUrl {
  get { EnsureChildControls(); return imgDate.ImageUrl; }
  set { EnsureChildControls(); imgDate.ImageUrl = value; }
 }
 public void Reset() {
  txtDate.Text = ""; cldDate.SelectedDates.Clear(); cldDate.VisibleDate = DateTime.Now;
 }
 public override void RenderControl(HtmlTextWriter writer) {
  txtDate.RenderControl(writer); imgDate.RenderControl(writer); cldDate.RenderControl(writer);
 }
}
```

Note: now build the project to compile and generate an assembly which will have the name “AspControls.dll”.

In CustomControls, CreateChildControls method should be used for writing the logic for creating the child controls, setting the properties and binding event handlers to events as we performed in the above code. CreateChildControls method is defined as “virtual” in “Control” class which is a parent class for all Controls, so we have overriden that method and implemented the logic for creating TextBox, ImageButton and Calendar.

EnsureChildControls method is typically used in Custom Controls which use child controls for functionality. The EnsureChildControls method is called in order to make sure that child controls have been created and are ready to process input, to perform data binding, or to perform other tasks. This method first checks whether the child controls are created or not and if not created then CreateChildControls method is called.

Every Asp.Net Server Control contains RenderControl method which is automatically called by the page during rendering, which outputs server control content to the provided HtmlTextWriter object. This method is defined as “virtual” in “Control” class which is a parent class for all Controls and Custom Control developers should override this method and implement the logic for rendering their control. We have overriden that method in our Control and called the RenderControl method of TextBox, ImageButton and Calendar because those controls already contain that method.

Consuming the Custom Control: the advantage with Custom Controls is they can be added to ToolBox and then we can “Drag and Drop” them on any WebForm and to do that go back to our regular project i.e. “ControlsDemo”, open the ToolBox window, right click on it and choose the option “Add Tab” which will add a new tab, specify a name to it for example: “NIT Controls”. Now right click on the new tab and select the option “Choose Items” which will open “Choose ToolBox Items” window, click on the “Browse” button on it and select “AspControls.dll” from its physical location and click “Ok” which will add “CustomCalendar” under the new tab.

Registering the Custom Control Page level: to register the Custom Control page level use the following statement on a WebForm below page directive:

<%@ Register Assembly="AspControls" Namespace="AspControls" TagPrefix="NIT2" %>

Registering the Custom Control Application level: to register the Custom Control application level open the Web.config file and write code under <system.web> tag where we registered "UserCalendar" which should now look as following:

```
<pages>
  <controls>
    <add assembly="AspControls" namespace="AspControls" tagPrefix="NIT2"/>
  </controls>
</pages>
```

Now add a new WebForm naming it as "TestCustomCalendar.aspx" and write the following code under tag:

```
<h2 style="background-color:yellow;color:red;text-align:center">Application for Date of Birth Certificate</h2>
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<table align="center">
  <caption>Applicant Details</caption>
  <tr> <td>Name:</td> <td><asp:TextBox ID="txtName" runat="server" /></td> </tr>
  <tr>  <td>Gender:</td>
    <td>
      <asp:RadioButton ID="rbMale" runat="server" Text="Male" GroupName="Gender" />
      <asp:RadioButton ID="rbFemale" runat="server" Text="Female" GroupName="Gender" />
    </td>
  </tr>
  <tr> <td>Date of Birth:</td>
    <td>
      <asp:UpdatePanel ID="UpdatePanel1" runat="server"> <ContentTemplate>
        <NIT2:CustomCalendar ID="ccDate" runat="server" ButtonImageUrl="~/Icons/Calendar.ico" />
      </ContentTemplate> </asp:UpdatePanel>
    </td>
  </tr>
  <tr> <td>Mobile No:</td> <td><asp:TextBox ID="txtMobile" runat="server" /></td> </tr>
  <tr> <td>Address:</td> <td> <asp:TextBox ID="txtAddress" runat="server" TextMode="MultiLine" /> </td> </tr>
  <tr> <td colspan="2" align="center">
      <asp:Button ID="btnSubmit" runat="server" Text="Submit Data" Width="100" />
      <asp:Button ID="btnClear" runat="server" Text="Reset Form" Width="100" />
    </td>
  </tr>
</table>
```

Now goto TestCustomCalendar.aspx.cs file and write the following code

Code under Page_Load:

```
if (!IsPostBack) { txtName.Focus(); }
```

Code under Submit Data Button:

```
Response.Write("<script>alert('" + txtName.Text + "')</script>");
if (rbMale.Checked) { Response.Write("<script>alert('Applicant is male')</script>"); }
else if (rbFemale.Checked) { Response.Write("<script>alert('Applicant is female')</script>"); }
Response.Write("<script>alert('" + ccDate.CalendarSelectedDate.ToShortDateString() + "')</script>");
Response.Write("<script>alert('" + txtMobile.Text + "')</script>");
Response.Write("<script>alert('" + txtAddress.Text + "')</script>");
```

Code under Reset Form Button:

```
txtName.Text = txtAddress.Text = txtMobile.Text = ""; rbMale.Checked = rbFemale.Checked = false;
ccDate.Reset(); txtName.Focus();
```

Developing a VideoPlayer control as a Custom Control: Most of the existing Html controls are developed by Microsoft as "Asp.Net Server Controls" and given for us to consume and all those controls will finally render Html, whereas controls introduced in "Html 5" like <audio> and <video> are not provided to us in Asp.Net as Server Controls, so let us create a VideoPlayer control which will render into "Html 5 Video".

To perform this task go to "AspControls" project under which we have developed "CustomCalendar", add a new "Web Form Server Control" in it naming it as "VideoPlayer.cs", delete the whole content present inside the class and write our code in it which should look as following:

```
public class VideoPlayer : WebControl {
  public bool AutoPlay { get; set; } = false;
  public new bool Controls { get; set; } = false;
  public bool Loop { get; set; } = false;
  public bool Muted { get; set; } = false;
  public string Poster { get; set; } = null;
  public string Mp4Url { get; set; } = null;
  public string WebMUrl { get; set; } = null;
  public string OggUrl { get; set; } = null;
  public override void RenderControl(HtmlTextWriter writer) {
    writer.AddAttribute(HtmlTextWriterAttribute.Id, this.ID);
    writer.AddAttribute(HtmlTextWriterAttribute.Name, this.ID);
    writer.AddAttribute(HtmlTextWriterAttribute.Width, this.Width.ToString());
    writer.AddAttribute(HtmlTextWriterAttribute.Height, this.Height.ToString());
    if (AutoPlay) { writer.AddAttribute("autoplay", "autoplay"); }
    if (Controls) { writer.AddAttribute("controls", "controls"); }
    if (Loop) { writer.AddAttribute("loop", "loop"); }
    if (Muted) { writer.AddAttribute("muted", "muted"); }
    if (Poster != null) { writer.AddAttribute("poster", this.Poster); }

    writer.RenderBeginTag("video");                                    //Generates <video> tag
    if (this.Mp4Url != null) {
      writer.AddAttribute(HtmlTextWriterAttribute.Src, this.Mp4Url);   //Generates src attribute for <source> tag
      writer.AddAttribute(HtmlTextWriterAttribute.Type, "video/mp4");  //Generates type attribute for <source> tag
      writer.RenderBeginTag("source");                                 //Generate <source> tag
      writer.RenderEndTag();                                           //Generate end tag for <source>
    }
    if (this.WebMUrl != null) {
```

```
    writer.AddAttribute(HtmlTextWriterAttribute.Src, this.WebMUrl);   //Generates src attribute for <source> tag
    writer.AddAttribute(HtmlTextWriterAttribute.Type, "video/webm");//Generates type attribute for <source> tag
    writer.RenderBeginTag("source");                                  //Generate <source> tag
    writer.RenderEndTag();                                            //Generate end tag for <source>
   }
   if (this.OggUrl != null) {
    writer.AddAttribute(HtmlTextWriterAttribute.Src, this.OggUrl);    //Generates src attribute for <source> tag
    writer.AddAttribute(HtmlTextWriterAttribute.Type, "video/ogg");   //Generates type attribute for <source> tag
    writer.RenderBeginTag("source");                                  //Generate <source> tag
    writer.RenderEndTag();                                            //Generate end tag for <source>
   }
   writer.RenderEndTag();                                             //Generates end tag for <video>
  }
}
```

Consuming the VideoPlayer Control: to consume the VideoPlayer we have developed; first compile the project so that the VideoPlayer control is added to "ASPControls.dll", now go back to "ControlDemo" project, open the ToolBox window, right click on "NIT Controls" tab we have added earlier, select "Choose Items", click "Browse" button and select "ASPControls.dll" from its physical location which will add "VideoPlayer" control below the "CustomCalendar". Now Add 2 new folders under the project naming them as "Images" and "Videos" and then add 1 image into "Images" folder and 1 video into "Videos" folder, then add a new WebForm in the project naming it as "TestVideoPlayer.aspx" and write the following code in its tag:

```
<NIT2:VideoPlayer ID="Video1" runat="server" Poster="/Images/Mumbai.jpg" Mp4Url="/Videos/Mumbai.mp4" Controls="true" Muted="true" Loop="true" Width="400" Height="400" />
```

Master Pages

One attribute of a well-designed website is a consistent site-wide page layout. Take the www.nareshit.com website for example; every page has the same content at the top and bottom of the page. At the very top of each page displays a light gray bar with a list of locations where the branches are located. Beneath that are the company logo, and the core sections: Home, All Courses, Software Training, Services, and so forth. Likewise, the bottom of the page includes information about clients, Address, contact details, quick links etc.

Another attribute of a well-designed site is the ease with which the site's appearance can be changed. For instance, the look and feel can be changed in the future, perhaps the menu items along the top will expand to include a new section or may be a radically new design with different colors, fonts, and layout will be unveiled. Applying such changes to the entire site should be a fast and simple process that does not require modifying the thousands of web pages that make up the site.

Creating a site-wide page template in ASP.NET is possible using Master Pages. In a nutshell, a Master Page is a special type of ASP.NET page that defines the markup that is common among all Content Pages (a Content Page is an ASP.NET page that is bound to the Master Page). Whenever a Master Page's layout or formatting is

changed, all its Content Pages output is immediately updated, which makes applying site-wide appearance and changes as easy by updating and deploying a single file i.e., Master Page.

Understanding How Master Pages Work:

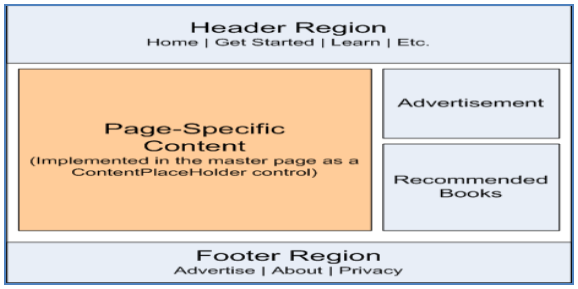
Building a website with a consistent site-wide page layout requires that each web page emit common formatting markup for example, while each page on www.nareshit.com have their own unique content, each of these pages also render a series of common elements that display the top-level section links: Home, All Courses, Software Training, Services, and so on.

There are a variety of techniques for creating web pages with a consistent look and feel. A naive approach is to simply copy and paste the common layout markup into all web pages, but this approach has several downsides. For starters, every time a new page is created, you must remember to copy and paste the shared content into the page. Such copying and pasting operations are ripe for error as you may accidentally copy only a subset of the shared markup into a new page. And to top it off, this approach makes replacing the existing site-wide appearance with a new one a real pain because every single page in the site must be edited to use the new look and feel.

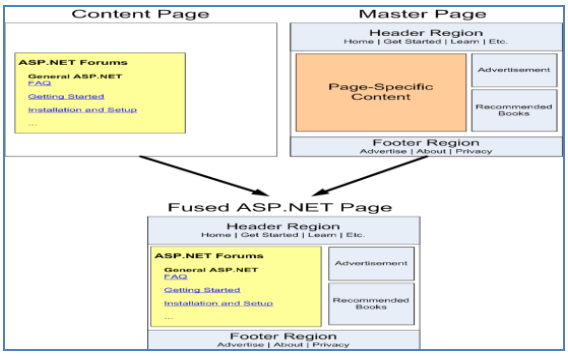
Prior to ASP.NET version 2.0, page developers often placed common markup in User Controls and then added these User Controls to each page. This approach requires the page developer to remember that they need to manually add the User Controls to every new page, and this allowed for easier site-wide modifications because when updating the common markup only the User Controls needed to be modified. Unfortunately, Visual Studio.NET 2002 and 2003 which used to create ASP.NET 1.x applications - rendered User Controls in the Design view as gray boxes. Consequently, page developers using this approach did not enjoy a WYSIWYG design-time environment.

The shortcomings of using User Controls were Cited in ASP.NET version 2.0 and Visual Studio 2005 with the introduction of master pages. A master page is a special type of ASP.NET page that defines the site-wide markup and the regions where associated content pages can define their custom markup and those regions are defined by "ContentPlaceHolder" controls. The "ContentPlaceHolder" control simply denotes a position in the master page's control hierarchy where custom content can be injected by a content page.

Below figure shows how a master page might look like. Note that the master page defines the common site-wide layout - the markup at the top, bottom, and right of every page - as well as a "ContentPlaceHolder" in the middle-left where the unique content of each individual web page must come and sit.



Once a master page has been defined it can be bound to a new ASP.NET page and those ASP.NET pages (content pages), include a Content control for each of the master page's ContentPlaceHolder controls. When the content page is visited through a browser the ASP.NET engine creates the master page's control hierarchy and injects the content page's control hierarchy into the appropriate places and the combined control hierarchy is rendered and the resulting HTML is returned to the end user's browser. Below figure illustrates this concept:



Creating a Master Page: To add a Master Page in our site, open the "Add New Item" window and select the option "WebForms Master Page" and name it as "Site.master" (.master is the extension of a Master Page). The first line in the declarative markup is the @Master directive and this is like the @Page directive that appears in our ASP.NET pages and all attributes will be same as @Page directive attributes only.

Now if we look into the code of the file we find a "ContentPlaceHolder" control named head and this control appears within the <head> section and can be used to declaratively add content in to the <head> element like style and script code. We also find another "ContentPlaceHolder" control named "ContentPlaceHolder1" and this control appears within the Web Form and serves as the region for the content page's user interface.

Now delete the code present inside the <form> tag along with the "ContentPlaceHeader" and write the below code in it:

```
<div id="divHeader" style="background-color: lightpink; color: aqua; text-align: center; font-size: xx-large">
  Naresh I Technologies
</div>
<div id="divMenu" style="background-color: beige; text-align: center">
  <asp:HyperLink ID="hlHome" runat="server" NavigateUrl="~/Home.aspx" Text="Home" /> 
  <asp:HyperLink ID="hlMission" runat="server" NavigateUrl="~/Mission.aspx" Text="Mission" /> 
  <asp:HyperLink ID="hlAbout" runat="server" NavigateUrl="~/About.aspx" Text="About" />
</div>
<div id="divContent"> <asp:ContentPlaceHolder ID="body" runat="server" /> </div>
<div id="divFooter" style="background-color: azure">
  <fieldset style="border:dotted blue;color:brown">
    <legend>Contact Us</legend>
    Address: Opp. Satyam Theatre, Ameerpet, Hyderabad – 16 <br />
    Phone: 2374 6666 <br /> Email: info@nareshit.com <br /> Website: www.nareshitc.om
  </fieldset>
</div>
```

Creating Associated Content Pages: with the above master page created, we are now ready to start creating ASP.NET pages that are bound to the master page. Such pages are referred to as content pages and we need to create 3 content pages Home.aspx, Mission.aspx and About.aspx.

Add a new ASP.NET page to the project and bind it to the "Site.master" master page and to do that right-click on the project in Solution Explorer and choose "Add New Item" option, select the "Web Form with Master Page" template, enter the name as "Home.aspx" and click "Add", which opens "Select a Master Page" dialog box

from where you can choose the master page to use. This action will add a content page which contains a @Page directive that points back to its master page and a “Content” control for each of the Master Page's “ContentPlaceHolder” controls. Currently we have 2 “ContentPlaceHolder” controls on Master Page so we find 2 “Content” controls on this page and the code will be as following in “Home.aspx”:

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true"
                    CodeBehind="Home.aspx.cs" Inherits="ControlsDemo.Home" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server"></asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="body" runat="server"></asp:Content>
```

Title attribute is to set a title for our page, give a value to title as “Home Page” and then write the following code under the 2 “Content” controls:

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
 <style>
  p { text-align: justify; color: palevioletred; font-family: Arial; text-indent: 50px; }
  ol { color: cadetblue; background-color: burlywood }
 </style>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="body" runat="server">
 <p> We have set the pace with online learning. Learn what you want, when you want, and practice with the
      instructor-led training sessions, on-demand video tutorials which you can watch and listen. </p>
 <ol>
  <li>150+ Online Courses</li>
  <li>UNLIMITED ACCESS</li>
  <li>EXPERT TRAINERS</li>
  <li>VARIETY OF INSTRUCTION</li>
  <li>ON-THE-GO-LEARNING</li>
  <li>PASSIONATE TEAM</li>
  <li>TRAINING INSTITUTE OF CHOICE</li>
 </ol>
</asp:Content>
```

Same as the above add 2 more content pages naming them as “Mission.aspx” and “About.aspx”, set the title as “Mission Page” for “Mission.aspx” and “About Page” for “About.aspx”, and write below code under them:

Mission.aspx:

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
 <style>
  p { text-align: justify; color: blue; font-family: Calibri; text-indent: 50px; }
 </style>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="body" runat="server">
<p> We appreciate you taking the time today to visit our web site. Our goal is to give you an interactive tour of our new and used inventory, as well as allow you to conveniently get a quote, schedule a service appointment, or apply for financing. The search for a luxury car is filled with high expectations. Undoubtedly, that has a lot to do
```

with the vehicles you are considering, but at Motors, we think you should also have pretty high expectations for your dealership. </p>
</asp:Content>

About.aspx:

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
  <style>
    p { text-align: justify; color: green; font-family: 'Agency FB'; text-transform: capitalize; text-indent: 50px; }
  </style>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="body" runat="server">
<p> NareshIT (Naresh i Technologies) is a leading Software Training Institute and provides Job Guarantee Program through Nacre in Hyderabad, Chennai, Bangalore, Vijayawada and across the world with Online Training services. Managed and Lead by IT Professionals with more than a decade experience in leading MNC companies. We are most popular for our training approach that enables students to gain real-time exposure on cutting-edge technologies. </p>
</asp:Content>
```

Now run any of the above 3 content pages and we notice that they merge with master page we have created above and launches, click on any menu in the top will launch other pages in the site.

Creating another Master Page:

Add a new Master Page in our site naming it as “NITSite.master”. Add a new folder under the project naming it as “Images” and copy 2 images into it which we can use them as header and footer of our new Master Page. Delete the code present under the <form> tag of “MasterPage” and write the below code there:

```
<table style="width:100%">
  <tr><td colspan="2"><asp:Image ID="imgHeader" runat="server" Width="100%" Height="100"
        ImageUrl="~/Images/Header.png" /></td></tr>
  <tr>
    <td style="background-color: bisque; width: 25%; vertical-align: top">
      <b>Courses Offered:</b> <br />
      <center>
        <asp:HyperLink ID="hlCS" runat="server" Text="CSharp" NavigateUrl="~/CSharp.aspx" /> <br />
        <asp:HyperLink ID="hlAsp" runat="server" Text="Asp.Net Web Forms" NavigateUrl="~/Asp.aspx" /> <br />
        <asp:HyperLink ID="hlMvc" runat="server" Text="Asp.Net MVC" NavigateUrl="~/Mvc.aspx" /> <br />
        <asp:HyperLink ID="hlSql" runat="server" Text="Sql Server" NavigateUrl="~/Sql.aspx" />
      </center> <br /><br />
      <b>Start Date: </b> <asp:ContentPlaceHolder ID="cphDate" runat="server" /> <br />
      <b>Batch Time: </b> <asp:ContentPlaceHolder ID="cphTime" runat="server" /> <br />
      <b>Course Fees: </b> <asp:ContentPlaceHolder ID="cphFees" runat="server" /> <br />
      <b>Faculty: </b> <asp:Label ID="lblFaculty" runat="server" />
    </td>
    <td style="vertical-align: top; width: 75%; background-color: azure">
      <asp:ContentPlaceHolder ID="cphBody" runat="server" />
    </td>
  </tr>
```

```
 <tr>
  <td colspan="2">
   <asp:Image ID="imgFooter" runat="server" Width="100%" Height="50" ImageUrl="~/Images/Footer.png" />
  </td>
 </tr>
</table>
```

Add a new “Web Form with Master Page” under the project naming it as “CSharp.aspx” and write the below code in it under the 5 content controls and set the title of the page as “CSharp”.

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
 <style> p { text-align: justify; font-family: Arial; text-indent: 50px; color: dodgerblue; } </style>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="cphDate" runat="server">06/01/2020</asp:Content>
<asp:Content ID="Content3" ContentPlaceHolderID="cphTime" runat="server">11:00 A.M.</asp:Content>
<asp:Content ID="Content4" ContentPlaceHolderID="cphFees" runat="server">&#8377; 1500/-</asp:Content>
<asp:Content ID="Content5" ContentPlaceHolderID="cphBody" runat="server">
 <p>It is an Object-Oriented and Platform Independent programming language developed by Microsoft as part of the .NET initiative and approved as a standard by ECMA and ISO.</p>
 <p>Anders Hejlsberg leads development of the language, which has procedural, object-oriented syntax based on C++ and includes influences from several programming languages most importantly Delphi and Java with a particular emphasis on simplification.</p>
 <p>CSharp’s principal designer and lead architect Anders Hejlsberg, has previously involved with the design of Visual J++, Borland Delphi, and Turbo Pascal languages also. In interviews and technical papers he has stated that flaws in most major programming languages like C++, Java, Delphi, and Smalltalk drove the design of CSharp programming language.</p>
</asp:Content>
```

Add another “Web Form with Master Page” under the project naming it as “Asp.aspx” and write the below code in it under the 5 content controls and set the Title as “Asp.Net Web Forms”.

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
 <style> p { text-align: justify; font-family: Arial; text-indent: 50px; color: crimson; } </style>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="cphDate" runat="server">06/01/2020</asp:Content>
<asp:Content ID="Content3" ContentPlaceHolderID="cphTime" runat="server">7:00 A.M.</asp:Content>
<asp:Content ID="Content4" ContentPlaceHolderID="cphFees" runat="server">&#8377; 2000/-</asp:Content>
<asp:Content ID="Content5" ContentPlaceHolderID="cphBody" runat="server">
 <p>It is an open-source server-side web application framework designed by Microsoft for web development to produce dynamic web pages. It is designed to allow programmers to build Web Sites, Web Applications and Web Services.</p>
 <p>It was first released in January 5, 2002 with 1.0 version of .NET Framework, and is a successor to Microsoft's Active Server Pages (ASP) technology. The current version of ASP.Net is 4.7 released on April 5, 2017 which will be the last version and it is succeeded by ASP.Net Core. ASP.NET is built on the Common Language Runtime (CLR), allowing programmers to write ASP.NET code by using any supported .NET language and while coding we have access to all the BCL in .NET Framework, which enable us to benefit from the CLR, type safety, inheritance, polymorphism and so on.</p>
```

```
</asp:Content>
```

Add another “Web Form with Master Page” under the project naming it as “Mvc.aspx” and write the below code in it under the 5 content controls and set the Title as “Asp.Net MVC”.

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
  <style>
    p { text-align: justify; font-family: Arial; text-indent: 50px; color:coral }
  </style>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="cphDate" runat="server">06/01/2020</asp:Content>
<asp:Content ID="Content3" ContentPlaceHolderID="cphTime" runat="server">11:00 A.M.</asp:Content>
<asp:Content ID="Content4" ContentPlaceHolderID="cphFees" runat="server">&#8377; 1000/-</asp:Content>
<asp:Content ID="Content5" ContentPlaceHolderID="cphBody" runat="server">
  <p>The Model-View-Controller (MVC) architectural pattern separates an application into three main groups of components: Models, Views, and Controllers. This pattern helps to achieve separation of concerns. Using this pattern, user requests are routed to a Controller which is responsible for working with the Model to perform user actions and/or retrieve results of queries. The Controller chooses the View to display to the user, and provides it with any Model data if requires.</p>
  <p>MVC traditionally used for desktop graphical user interfaces (GUIs), this architecture has become popular for designing web applications and even mobile, desktop and other clients. Popular programming languages like Java, C#, Ruby, PHP and others have their own MVC frameworks that are currently being used in web application development.</p>
  <p>Trygve Reenskaug introduced MVC into Smalltalk-76 while visiting the Xerox Palo Alto Research Center in the 1970s. In the 1980s, Jim Althoff and others implemented a version of MVC for the Smalltalk-80 class library.</p>
</asp:Content>
```

Add another “Web Form with Master Page” under the project naming it as “Sql.aspx” and write the below code in it under the 5 content controls and set the Title as “Sql Server”.

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
  <style> p { text-align: justify; font-family: Arial; text-indent: 50px; color:darkorchid } </style>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="cphDate" runat="server">06/01/2020</asp:Content>
<asp:Content ID="Content3" ContentPlaceHolderID="cphTime" runat="server">2:00 P.M.</asp:Content>
<asp:Content ID="Content4" ContentPlaceHolderID="cphFees" runat="server">&#8377; 1000/-</asp:Content>
<asp:Content ID="Content5" ContentPlaceHolderID="cphBody" runat="server">
  <p>Microsoft SQL Server is a relational database management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data as requested by other software applications--which may run either on the same computer or on another computer across a network.</p>
  <p>Microsoft markets at least a dozen different editions of Microsoft SQL Server, aimed at different audiences ranging from small single-machine applications to large Internet applications with many concurrent users.</p>
  <p> Microsoft and Sybase released version 1.0 in 1989. However, the partnership between these two ended in the early 1990s. Microsoft maintained ownership rights to the name SQL Server. Since the 1990s, subsequent versions of SQL Server have been released including SQL Server 2000, 2005, 2008, 2012, 2014, 2016 and 2017.</p>
</asp:Content>
```

Now run any of the 4 content pages and watch the output, where we notice the Content Page launched by merging with Master Page and also, we find the links to other pages on the LHS of the page and by clicking on the appropriate link the corresponding page gets loaded.

Passing values to Controls present on a Master Page, from Content Pages: If we want to pass any value to a control that is present on a Master Page, from Content Page; 1st we need to give access to that Control to Content Page which can be performed in 4 different ways.

Option 1: By using the "Master" property in a "Content Page" we can get a reference to the "Master Page" of that "Content Page" and with that reference we can get access to the Controls of "Master Page" by calling "FindControl" method on that reference and to test that, go to "CSharp.aspx.cs" file and write the below code under "Page_Load" method:

```
MasterPage mp = Master;
Control ctrl = mp.FindControl("lblFaculty");
Label lblFaculty = (Label)ctrl;
lblFaculty.Text = "Bangarraju";
        Or
Control ctrl = Master.FindControl("lblFaculty");
Label lblFaculty = (Label)ctrl;
lblFaculty.Text = "Bangarraju";
        Or
Label lblFaculty = (Label)Master.FindControl("lblFaculty");
lblFaculty.Text = "Bangarraju";
        Or
((Label)Master.FindControl("lblFaculty")).Text = "Bangarraju";
```

Note: NITSite is a "Master Page" and the parent class for every "Master Page" is "MasterPage" class which is defined in "System.Web.UI" Namespace and this is very similar to a normal Web Page inheriting from "Page" class.

Option 2: By defining a property in the "Master Page" we can expose the Control to "Content Pages", so that "Content Pages" can access the Control and assign values to them and to test that, go to "NITSite.master.cs" file and write the below code in the class:

```
public Label Faculty {
  get { return lblFaculty; }
}
```

Now in the Content Page we can directly access the Control without calling "FindControl" method and to test that process go to "Asp.aspx.cs" file and write the below code in "Page_Load" Method:

```
MasterPage mp = Master;
NITSite obj = (NITSite)mp;
obj.Faculty.Text = "Bangarraju";
        Or
NITSite obj = (NITSite)Master;
obj.Faculty.Text = "Bangarraju";
        Or
((NITSite)Master).Faculty.Text = "Bangarraju";
```

Note: In the above case "Master" property will provide a reference to the "Master Page" of current "Content Page" but in the form of Parent i.e. "MasterPage" class but not as "NITSite" class, so with that reference we can't call "Faculty" property because "Parent can hold the reference of Child but can't access members of Child (Rule No. 3 of Inheritance)" and that is the reason why we have converted that reference from "Parent Type" (MasterPage) back into "Child Type" (NITSite) for accessing the "Faculty" property.

Option 3: As discussed above the return type of "Master" property in any "Content Page" is "MasterPage", and that property is defined in "Page" class which is the parent class of "Content Page", so we can re-implement that property in "Content Page" and change the return type of that property to our actual "Master Page" i.e. "NITSite" so that we can directly access the members of that class in "Content Page" and to test that process go to "Mvc.aspx.cs" file and write the below property in the class:

```
public new NITSite Master {
 get {
   return (NITSite)base.Master;
 }
}
```

Note: in the above case to re-implement the property we have used "Hiding or Shadowing" because that property is not declared as "Virtual" in Page class.

Now we can directly get access to "NITSite" class in "Content Page" by using the "Master" property we have defined above and to test that, write the following code in Page_Load method:

Master.Faculty.Text = "Sudhakar Sharma";

Option 4: Without again defining a "Master" property explicitly in the "Content Pages" we are provided with "MasterType" directive which if used in "Content Pages", will automatically define the "Master" property same as we defined above and to test that, go to "Sql.aspx" file and write the following statement below "Page Directive":

<%@ MasterType VirtualPath="~/NITSite.Master" %>

This action will immediately define a Master property under the current class and to check that, open Solution Explorer, expand the WebForm "Sql.aspx", and below that we find "Sql.aspx.designer.cs" file, open it and there we find "Master" property same as we defined in "Mvc.aspx.cs" file above, so now we can directly use that "Master" property and access the "Faculty" property, and to test that go to "Sql.aspx.cs" file and write the below code in "Page_Load" method:

Master.Faculty.Text = "Sudhakar Ladda";

Theme

ASP.NET themes are a collection of properties that define the appearance of pages and controls in your Web site. A theme can include skin files, which define property settings for ASP.NET Web Server Controls, and can also include cascading style sheet files (.css files) and graphics. By applying a theme, you can give the pages in your Web Site a consistent appearance.

A theme is a collection of property settings that allow you to define the look of pages and controls, and then apply the look consistently across all the pages in a Web application. Themes are made up of a set of elements like skins, cascading style sheets (CSS), images, and other resources also. At a minimum, a theme will contain skins. Themes are defined in special directories in your Web Site or on your Web Server with the name "App_Themes".

Skin: 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="Red" />
```

We create ".skin" files in the Theme folder. A ".skin" file can contain one or more control skins for one or more control types. You can define skins in a separate file for each control or define all the skins for a theme in a single file. 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.)

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: 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 and these are used for styling Html Elements.

Theme Graphics and Other Resources: Themes can also include graphics and other resources, such as script files or sound files. For example, part of your page theme might include a skin for a TreeView control. As part of the theme, you can include the graphics used to represent the expand button and the collapse button. Typically, the resource files for the theme are in the same folder as the skin files for that theme, but they can be elsewhere in the Web application, in a subfolder of the theme folder for example. To refer to a resource file in a subfolder of the theme folder, use a path like the one shown in this Image control skin:

```
<asp:Image runat="server" ImageUrl="ThemeSubfolder/filename.ext" />
```

You can also store your resource files outside the theme folder. If you use the tilde (~) syntax to refer to the resource files, the Web application will automatically find the images. For example, if you place the resources for a theme in a subfolder of your application, you can use paths of the form ~/SubFolder/filename.ext to refer to resource files, as in the following example.

```
<asp:Image runat="server" ImageUrl="~/AppSubfolder/filename.ext" />
```

Theme Settings Precedence: You can specify the precedence that theme settings take over local control settings by specifying how the theme is applied.

If you set a page's Theme property, control settings in the theme and the page are merged to form the final settings for the control. If a control setting is defined in both the control and the theme, the control settings from the theme override any page settings on the control. This strategy enables the theme to create a consistent look across pages, even if controls on the pages already have individual property settings. For example, it allows you to apply a theme to a page you created in an earlier version of ASP.NET.

Alternatively, you can apply a theme as a style sheet theme by setting the page's StylesheetTheme property. In this case, local page settings take precedence over those defined in the theme when the setting is defined in both places. This is the model used by cascading style sheets. You might apply a theme as a style sheet theme if you want to be able to set the properties of individual controls on the page while still applying a theme for an overall look.

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. The following example shows a typical page theme, defining two themes named BlueTheme and PinkTheme as following:

MyWebSite

- App_Themes
  - BlueTheme
    - Controls.skin
    - BlueTheme.css
  - PinkTheme
    - Controls.skin
    - PinkTheme.css

Create a page theme: Go to Solution Explorer, right-click the project under which we want to create a page theme, Click Add, Click Add ASP.NET Folder and then Click Theme. If the App_Themes folder does not already exist, Visual Studio creates it first and then creates a new folder for the theme as a child folder of the App_Themes folder, rename it as "1stTheme". The name of this folder is also the name of the page theme. For example, if you create a folder named \App_Themes\1stTheme then the name of your theme is "FirstTheme". Now you can add files to your Theme for control skins, style sheets, and images that make up the theme.

Adding a skin file to our page theme: In Solution Explorer, right-click the name of your theme and then click Add New Item and in the Add New Item dialog box, click Skin File, in the Name box, type a name for the .skin file as "1stSkin", and then click Add which add "1stSkin.skin" under "1stTheme".

Note: The typical convention is to create one .skin file per control, such as Button.skin or Calendar.skin. However, you can even define skin for multiple controls under a single ".skin" file.

In the .skin file, add a normal control definition by using declarative syntax, but include only the properties that you want to set for the theme. The control definition must include the runat="server" attribute, and it must not include the ID="" attribute. Now write the following code under skin file:

```
<asp:TextBox runat="server" BackColor="#CCFF99" BorderColor="Red" BorderStyle="Solid" BorderWidth="5px"
        Font-Names="Arial Black" Font-Size="X-Large" Width="300px" ForeColor="#0033CC" />
<asp:Button runat="server" BackColor="#33CCFF" BorderColor="Yellow" BorderStyle="Solid" BorderWidth="5px"
        Font-Names="Arial Black" Font-Size="X-Large" ForeColor="Red" Width="150px" Height="60px" />
<asp:Label runat="server" BackColor="YellowGreen" BorderColor="Blue" BorderStyle="Double"
BorderWidth="5px" Font-Names="Arial Black" Font-Size="XX-Large" ForeColor="Red" />
```

This Button, Label and TextBox control skin's does not contain a skinID attribute so they will be applied to all of the Buttons and TextBoxs controls in the themed application that do not specify the skinID attribute.

Note: An easy way to create a control skin is to add the control to a page and configure it so that it has the look you want. For example, you might add a Calendar control to a page and set its day header, selected date, and other properties. Then, you can copy the control definition from the page to a skin file, but you must remove the ID attribute.

Adding a cascading style sheet file to our page theme: In Solution Explorer, right-click the name of your theme and then click Add New Item and in the Add New Item dialog box, click Style Sheet, in the Name box, type a name for the .css file as "1stCSS", and then click Add which add "1stCSS.css" under "1stTheme". Now write the following code under css file: **body { color: brown; font-size: x-large; background-color: bisque; }**

Now add a new Web Form in our project naming it as "ThemeLoginForm.aspx" and write the following code under its tag:

```
<table align="center">
 <caption><asp:Label ID="lblTitle" runat="server" Text="Login Form" /></caption>
 <tr>
  <td>User Name:</td>
  <td><asp:TextBox ID="txtName" runat="server" /></td>
 </tr>
 <tr>
  <td>Password:</td>
  <td><asp:TextBox ID="txtPwd" runat="server" TextMode="Password" /></td>
 </tr>
 <tr>
  <td>Email Id:</td>
  <td><asp:TextBox ID="txtEmail" runat="server" TextMode="Email" /></td>
 </tr>
 <tr>
  <td colspan="2" align="center">
   <asp:Button ID="btnLogin" runat="server" Text="Login" />
   <asp:Button ID="btnReset" runat="server" Text="Reset" />
  </td>
 </tr>
</table>
```

Applying an ASP.NET Theme: we can apply themes at "Page Level" or "Application Level". Setting a theme at Application Level applies styles and skins to all the pages and controls in the Site unless you override a theme for an individual page. Setting a theme at the page level applies styles and skins to that page and all its controls.

Applying a Theme Page Level: set the "Theme" or "StyleSheetTheme" attribute of the @Page directive with the name of the theme we want to use, as shown in the following example:

```
<%@ Page Theme="1stTheme" …… %> or
<%@ Page StyleSheetTheme="1stTheme" …… %>
```

Note: By default, themes override local control settings. Alternatively, you can set a theme as a style sheet theme, so that the theme applies only to control settings that are not explicitly set on the control.

Now run the “ThemeLoginForm.aspx” page and watch the output, the theme and its corresponding styles and skins now apply the Button’s and TextBox’s present on the page.

Applying a theme Application Level: in this case we need to apply the theme for Application in Web.config file and to test this first delete the “Theme or StyleSheetTheme” attribute to Page directive of our “ThemeLoginForm.aspx”, open Web.config file and under <system.web> tag if <pages> tag is available add "theme" attribute to it which should look as following:

<pages theme="1stTheme"> **or**
<pages styleSheetTheme="1stTheme">

Named Skins: we can define only one default skin per control whereas by using the “SkinID” attribute in the control’s skin declaration we can create named skins for the same type of control. To test this open the “1stSkin.skin” file again and add the following code in it:

```
<asp:TextBox SkinId="EmailSkin" runat="server" BackColor="#CCEE88" BorderColor="Green" BorderStyle="Ridge"
BorderWidth="5px" Font-Names="Arial Black" Font-Size="X-Large" Width="300px" ForeColor="#0033DD" />
<asp:TextBox SkinId="PwdSkin" runat="server" BackColor="Yellow" BorderColor="#0066FF" BorderStyle="Ridge"
BorderWidth="5px" Font-Names="Arial Black" Font-Size="X-Large" Width="300px" ForeColor="#FF6600" />
```

In the above case we defined 2 new skins for TextBox with name “PwdSkin” and “EmailSkin” and our intention is to apply “PwdSkin” to all Password TextBox’s and “EmailSkin” to all Email TextBox’s in our site, but when we run “ThemeLoginForm.aspx” and watch the output we notice the default skin only applied to all the 3 TextBox's whereas if we want to apply “PwdSkin” to Password TextBox and “EmailSkin” to Email TextBox we need to explicitly specify that and to do that go to Design View of the “.aspx” file, go to properties of Password TextBox and select “PwdSkin” under “SkinId” property of that control and then , go to properties of Email TextBox and select “EmailSkin” under “SkinId” property of that control and run the Web Form to watch difference in output.

Disable ASP.NET Themes: we can configure a page or control to ignore themes. Themes override local settings for page and control appearance by default. Disabling this behavior is useful when a control or page already has a predefined look that you do not want the theme to override.

To disable themes for a page, set the EnableTheming attribute of the @Page directive to false, as following:

```
<%@ Page EnableTheming="false" %>
```

To disable themes for a control set the EnableTheming property of the control to false, as following:

```
<asp:Calendar id="Calendar1" runat="server" EnableTheming="false" />
```

Applying ASP.NET Themes Programmatically: in addition to specifying theme and skin preferences in page declarations and configuration files, you can apply themes programmatically and to do that we need to apply theme programmatically in the Page’s PreInit Event Handler method as following:

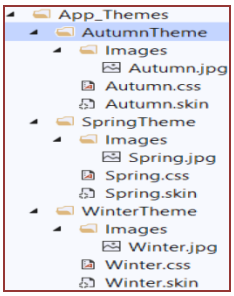
```
Page.Theme = "BlueTheme";
```

Add a new Web Form in the project naming it as “ThemeRegistrationForm.aspx” and write the following code under its tag:

```
<table align="center">
 <caption><asp:Label ID="lblTitle" runat="server" Text="Registration Form" /></caption>
 <tr><td colspan="2" align="center">
   <asp:DropDownList ID="ddlTheme" runat="server" AutoPostBack="true">
    <asp:ListItem Text="-Choose Theme-" Value="ChooseTheme" />
    <asp:ListItem Text="Autumn" Value="AutumnTheme" />
```

```
      <asp:ListItem Text="Spring" Value="SpringTheme" />
      <asp:ListItem Text="Winter" Value="WinterTheme" />
    </asp:DropDownList></td>
  </tr>
  <tr><td><asp:Label ID="lblName" runat="server" Text="User Name:" /></td>
    <td><asp:TextBox ID="txtName" runat="server" /></td></tr>
  <tr><td><asp:Label ID="lblPwd" runat="server" Text="Password:" /></td>
    <td><asp:TextBox ID="txtPwd" runat="server" TextMode="Password" /></td></tr>
  <tr><td><asp:Label ID="lblCPwd" runat="server" Text="Confirm Pwd:" /></td>
    <td><asp:TextBox ID="txtCPwd" runat="server" TextMode="Password" /></td></tr>
  <tr><td><asp:Label ID="lblAadhar" runat="server" Text="Aadhar Id:" /></td>
    <td><asp:TextBox ID="txtAadhar" runat="server" /></td></tr>
  <tr><td><asp:Label ID="lblMobile" runat="server" Text="Mobile No:" /></td>
    <td><asp:TextBox ID="txtMobile" runat="server" TextMode="Phone" /></td></tr>
  <tr><td><asp:Label ID="lblEmail" runat="server" Text="Email Id:" /></td>
    <td><asp:TextBox ID="txtEmail" runat="server" TextMode="Email" /></td></tr>
  <tr><td colspan="2" align="center">
      <asp:Button ID="btnRegister" runat="server" Text="Register" />
      <asp:Button ID="btnReset" runat="server" Text="Reset" /></td></tr>
</table>
```

Now add 3 new themes under “App_Themes” folder naming them as “AutumnTheme”, “SpringTheme” and “WinterTheme”. Under each theme add 1 folder and name it as “Images” and add 1 image into that folder representing the seasonal theme. Under each theme add 1 skin file and name them as “Autumn.skin”, “Spring.skin” and “Winter.skin” respectively. Under each theme add 1 css file and name them as “Autumn.css”, “Spring.css” and “Winter.css” respectively. This is now finally looking at following:



Now write the following code under each .css and .skin files:

Autumn.css:

```
body {background-image: url(../AutumnTheme/Images/Autumn.jpg); background-size: cover }
```

Spring.css:

```
body { background-image:url(../SpringTheme/Images/Spring.jpg);background-size:cover }
```

Winter.css:

```
body { background-image: url(../WinterTheme/Images/Winter.jpg); background-size: cover }
```

Autumn.skin:

```
<asp:TextBox runat="server" BorderColor="Blue" BorderStyle="Solid" BorderWidth="5px" Font-Names="Arial Black" Font-Size="X-Large" Width="300px" />
```

```
<asp:Button runat="server" BorderColor="Blue" BorderStyle="Solid" BorderWidth="5px" Font-Names="Arial Black" Font-Size="X-Large" />
<asp:Label runat="server" BorderColor="Blue" BorderStyle="Solid" BorderWidth="5px" Font-Names="Arial Black" Font-Size="X-Large" BackColor="Azure" />
<asp:DropDownList runat="server" BorderColor="Blue" BorderStyle="Solid" BorderWidth="5px" Font-Names="Arial Black" Font-Size="X-Large" />
```

Spring.skin:

```
<asp:TextBox runat="server" BorderColor="Green" BorderStyle="Groove" BorderWidth="5px" Font-Names="Arial Black" Font-Size="X-Large" Width="300px" />
<asp:Button runat="server" BorderColor="Green" BorderStyle="Groove" BorderWidth="5px" Font-Names="Arial Black" Font-Size="X-Large" />
<asp:Label runat="server" BorderColor="Green" BorderStyle="Groove" BorderWidth="5px" Font-Names="Arial Black" Font-Size="X-Large" BackColor="Azure" />
<asp:DropDownList runat="server" BorderColor="Green" BorderStyle="Groove" BorderWidth="5px" Font-Names="Arial Black" Font-Size="X-Large" />
```

Winter.skin:

```
<asp:TextBox runat="server" BorderColor="Red" BorderStyle="Ridge" BorderWidth="5px" Font-Names="Arial Black" Font-Size="X-Large" Width="300px" />
<asp:Button runat="server" BorderColor="Red" BorderStyle="Ridge" BorderWidth="5px" Font-Names="Arial Black" Font-Size="X-Large" />
<asp:Label runat="server" BorderColor="Red" BorderStyle="Ridge" BorderWidth="5px" Font-Names="Arial Black" Font-Size="X-Large" BackColor="Azure" />
<asp:DropDownList runat="server" BorderColor="Red" BorderStyle="Ridge" BorderWidth="5px" Font-Names="Arial Black" Font-Size="X-Large" />
```

Now goto "ThemeRegistrationForm.aspx.cs" file and write the below code in ThemeRegistrationForm class:

```
protected void Page_PreInit(object sender, EventArgs e) {
  string SelectedTheme = Request.Form["ddlTheme"];
  if (SelectedTheme != "ChooseTheme") {
    this.Theme = SelectedTheme;
  }
}
```

ViewState

Web applications are stateless i.e., these applications do not save data of 1 request, for using it in the next request. When an application is stateless, the server does not store any state about the client session. To test this, add a new WebForm naming it as "HitCount.aspx" and write the following code in its tag:

```
<asp:Button ID="btnCount" runat="server" Text="Hit Count" /> <br />
<asp:Label ID="lblCount" runat="server" ForeColor="Red" />
```

Now goto "HitCount.aspx.cs" file and write the following code in it:

Declarations:

```
int Count = 0;
```

Code under Hit Count Button Click:

```
Count += 1;
lblCount.Text = "Hit Count: " + Count;
```

Now run the Web Form and click on the button for any no. of times still it will display the output as 1 only because every time we click on the button, server will create the instance of our page class, initializes "Count" with "0", increments the value of Count to "1", sends the output to client machine and destroys the page instance, and this process repeats every time we click on the button so every time the value of Count is "1" only and this is called as "Stateless" behavior. To overcome the above problem, we use ViewState which is a technique used to store user data on page at the time of post back of a web page.

Syntax of storing a value into ViewState: ViewState[string name] = value (Object);

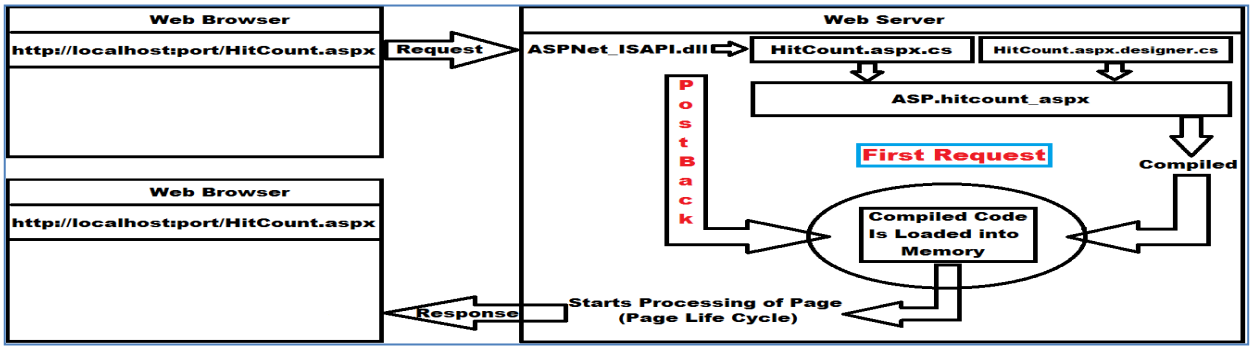
Syntax of accessing a value from ViewState: Object value = ViewState[string name];

To resolve the problem in our previous page rewrite the code under click of the button as following:

```
if (ViewState["Counter"] == null) { Count = 1; }
else { Count = (int)ViewState["Counter"] + 1; ViewState["Counter"] = Count; lblCount.Text = "Hit Count: " + Count; }
```

Note: Same as the above every Control will also maintain the state of its values and we call it as ControlState which will hold the values of controls and restores the value to control after page post back.

Processing a Page on the Server



When the request comes from a client for the first time to a page, ASPNet_ISAPI.dll on the Web Server will take the request, creates a new class "ASP.hitcount_aspx" inheriting from the class "HitCount" which is defined on "HitCount.aspx.cs" and "HitCount.aspx.designer.cs" files, compiles the new class and loads the compiled code into memory, and if at all the request comes for the second time (Post Back) for the page which is already compiled and loaded into memory, without recompiling the page again, the compiled code in the memory will be used for starting the life cycle of page.

Page Life Cycle

Life cycle of a page describes how a Web Page gets processed on the server i.e., when browser sends a request for that page.

Page Request: The page request occurs before the page life cycle begins, i.e., when the page is requested by a user, server determines whether the page needs to be parsed and compiled (therefore beginning the life cycle of a page) or whether a cached version of the page can be sent in response without running the page.

Various stages in the life cycle of a page will be as following:

Stage1: Start In this stage the page properties such as request and response are created and set, and at this stage the page also determines whether the request is a post back or a new request and sets the IsPostBack property, which will be true if it's a Post Back request or else false if it's a first request.

Stage2: Initialization During this stage all the controls on the page will be created and each controls unique Id property will be set.

Stage3: Load In this stage if the current request is a post back, control properties are loaded with information retrieved from View State (Control State).

Stage4: Validation In this stage any server-side validations that are required will be performed and sets the IsValid property of individual validator controls and then of the page.

Stage5: Event Handling If the request is a post back, then all the controls corresponding event handlers are called (which is required only) along with any cached events also.

Stage6: Rendering In this stage the page is rendered i.e., converts into Html and before rendering view state and control state is saved for the page and all controls.

Note: During the rendering stage, page calls the RenderControl method on each control and writes its output to the Output Stream object of Page's Response property.

Stage7: UnLoad: After the page has been rendered and the output is sent to client, it's ready to discard and Unload event is raised, and at this stage the Request and Response properties are destroyed, and cleanup is performed.

Page Events: Within each stage of page life cycle, page raises certain events which can be used for performing certain actions and these are the List of page Life Cycle events that we will use most frequently.

1. PreInit
2. Init
3. InitComplete
4. PreLoad
5. Load

-Control Events (All Postback and Cached Events)

6. LoadComplete
7. PreRender
8. PreRenderComplete
9. SaveStateComplete

-RenderControl Method Calling

10. UnLoad

1. **PreInit**: Raises after the start stage is completed and before the initialization stage begins. We use this event for performing the following actions:

I. Check the IsPostBack property to determine whether this is the first-time page is being processed.
II. The IsCallback and IsCrossPagePostBack properties are also set at this time.
III. Create or re-create Custom Controls.
IV. Set a Master Page dynamically.
V. Set the Theme property dynamically.

2. **Init**: Raises after all controls have been initialized and any skin settings have been applied. The Init event of individual controls occurs before the Init event of page. Use this event to read or initialize control properties.

3. **InitComplete**: Raised at the end of page's initialization stage. Only one operation takes place between Init and InitComplete events i.e., tracking of view state is turned on. View state tracking enables controls to persist any values that are programmatically added to the ViewState collection. Until view state tracking is turned on, any values added to view state are lost across post backs. Controls typically turn on view state tracking immediately after they raise their Init event. Use this event to make any changes to view state that we want to make sure are persisted after the next post back.

4. **PreLoad**: Raised after the page, loads view state for itself and all controls, and after it starts processing post back data that is included with the Request object.

5. **Load**: Page object calls the OnLoad method on Page, and then recursively does the same for each child control until all controls are loaded. The Load event of individual controls occurs after the Load event of page. Use this event to get or set properties of controls and to establish database connections.

Control events: Use these events to handle specific control events such as a Button control's Click event or a TextBox control's TextChanged event. In a post back request, if the page contains validation controls, check the IsValid property of Page before performing any processing.

6. **LoadComplete**: Raised at the end of event-handling stage and use this event for performing any tasks that require for all the controls on page that have been loaded.

7. **PreRender**: Raises after Page object has created all controls that are required to render the page, including child controls of Custom Controls. Page object raises this PreRender event on itself and then recursively does the same for each child control. Use this event to make any final changes to contents of page or its controls because hereafter rendering stage begins.

8. **PreRenderComplete**: Raises after all controls on page are ready for render.

9. **SaveStateComplete**: Raises after view state and control state have been saved for page and for all controls. Any changes to the page or controls at this point that affects rendering will not be retrieved on the next post back.

Render: This is not an event but a method and at this stage of processing, Page calls this method on each control. All ASP.NET Web Server Controls has a RenderControl method which writes the control's markup to "Output Stream" that must be sent to the browser as output.

10. **Unload**: Raised for each control and then for page. Use this event to do any final cleanup such as closing control-specific database connections.

Note: During unload stage - page and its controls rendering is completed, so you cannot make any further changes to the response stream and if you attempt to call a method such as "Response.Write" will throw an exception.

To test and understand about PageEvents add a new WebForm under the project naming it as "PageEvents.aspx", place a button on it and set the Text as "Post Back". Now go to "aspx.cs" file, copy the method "Page_Load" present in the class and paste it for 9 more times in the same class and rename the 9 methods as following: "Page_PreInit", "Page_Init", "Page_InitComplete", "Page_PreLoad", "Page_LoadComplete", "Page_PreRender", "Page_PreRenderComplete", "Page_SaveStateComplete" and "Page_Unload", and write a statement in each method describing about the event as following:

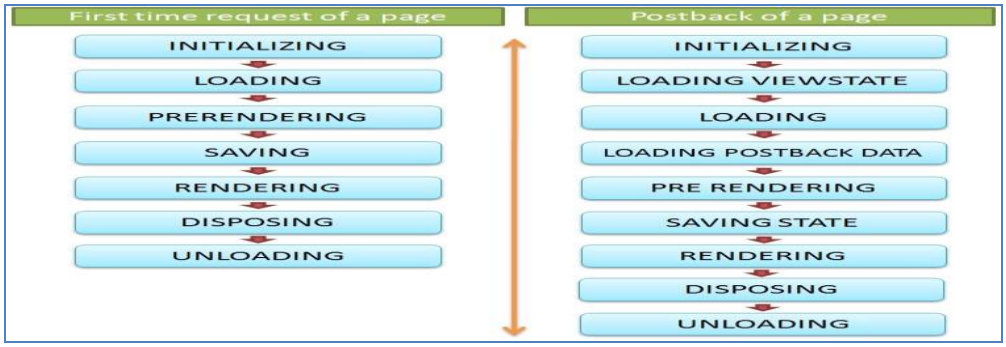
Code under Page_PreInit: Response.Write("Pre-Init Event Fired.
");
Code under Page_Init: Response.Write("Init Event Fired.
");
Code under Page_InitComplete: Response.Write("InitComplete Event Fired.
");
Code under Page_PreLoad: Response.Write("PreLoad Event Fired.
");
Code under Page_Load: Response.Write("Load Event Fired.
");
Code under Page_LoadComplete: Response.Write("LoadComplete Event Fired.
");
Code under Page_PreRender: Response.Write("PreRender Event Fired.
");
Code under Page_PreRenderComplete: Response.Write("PreRenderComplete Event Fired.
");
Code under Page_SaveStateComplete: Response.Write("SaveStateComplete Event Fired.
");
Code under Page_Unload: //Response.Write("Unload Event Fired.
");

Now generate a Click Event Handler for Button and write the following code in it:

```
Response.Write("<font color='red'>");
Response.Write("Button Click Event Fired. <br />");
Response.Write("</font>");
```

Now run the WebForm and watch the output, we notice all the events getting fired in a sequence, now click on the button to perform a Postback which will also fire Click Event along with all the other Events whereas click event will fire after the Load event and before the LoadComplete event.

Series of actions that are performed between the first request and postback request of a page:



AutoEventWireup attribute of Page Directive: ASP.NET supports an automatic way to bind Page Events to its Event Handlers if the AutoEventWireup attribute of the Page directive is set to true (default is also true only), provided Event Handler names follow a naming convention i.e "Page_<Event Name>" for example "Page_Init", "Page_Load", whereas if we set the attribute value as false or we don't follow the naming convention then Event Handlers should be explicitly bound to their corresponding events under Constructor of the page as following:

```
public PageEvents()
{
  this.PreInit += Page_PreInit;
  this.Init += Page_Init;
  this.InitComplete += Page_InitComplete;
  this.PreLoad += Page_PreLoad;
  this.Load += Page_Load;
  this.LoadComplete += Page_LoadComplete;
  this.PreRender += Page_PreRender;
  this.PreRenderComplete += Page_PreRenderComplete;
  this.SaveStateComplete += Page_SaveStateComplete;
  this.Unload += Page_Unload;
}
```

Tracing

ASP.NET tracing enables you to view diagnostic information about a single request for an ASP.NET page. ASP.NET tracing enables you to follow a page's execution path, display diagnostic information at run time, and debug your application. We can enable tracing in an ASP.Net application in 2 different levels:

1. Page Level Tracing
2. Application-Level Tracing

Page Level Tracing: If tracing is enabled page level, whenever a page is requested, ASP.NET appends to the page a series of tables containing execution details about the page request. Tracing is disabled by default in an ASP.NET application and to enable it add Trace attribute to Page directive with the value true as following:

```
<%@ Page Trace="true" Language="C#" ... %>
```

Note: To test this, add the Trace attribute to "PageEvents.aspx" and run the page, and we will see the trace data at bottom of the page.

Application Level Tracing: Instead of enabling tracing for individual pages, you can enable it for your entire application and in that case every page in our application will perform tracing. Application tracing is useful in realtime environments because you can easily enable it and disable it without editing individual pages. When tracing is enabled application level it will not display trace information on the page, but we need to view trace information with the help of a trace viewer. To enable Application-Level Tracing first delete your Page Level Tracing, now open the Web.config file and add the following information to it under <system.web> tag:

```
<trace enabled="true" />
```

Now run the application and notice in the page we will not see any trace data at bottom of the page and to view trace data we need to use the trace viewer i.e. "Trace.axd" and to do that copy the URL of your current executing page, open a new tab in the browser, paste the URL in the new tab and there delete the "PageEvents.aspx" file name and write trace.axd, for example:

http://localhost:port/PageEvents.aspx => http://localhost:port/trace.axd

Attributes of <trace> element in Web.config:

1. **Enabled:** set it true to enable tracing for the application becase default is false.

```
<trace enabled="true" />
```

2. **PageOutput:** set it true to display trace data both in the page and in the trace viewer also, default is false.

```
<trace enabled="true" pageOutput="true" />
```

3. **RequestLimit:** when you enable tracing for an application, ASP.NET collects trace information for each request to that application up to the maximum number of requests you specify and the default number of requests is 10.

```
<trace enabled="true" pageOutput="true" requestLimit="20" />
```

4. **MostRecent:** by default, when the trace viewer reaches its request limit, the application stops recording trace requests. However, you can configure the MostRecent attribute to true (default is false) which will discard the old data when the maximum number of requests are reached and adds new entries.

```
<trace enabled="true" pageOutput="true" requestLimit="20" mostRecent="true" />
```

5. **LocalOnly:** set it true to make the trace viewer (trace.axd) available only on local Web server and default is also true, where as if set as false we can access trace data from remote or client machines also.

```
<trace enabled="true" pageOutput="true" requestLimit="20" mostRecent="true" localOnly="false" />
```

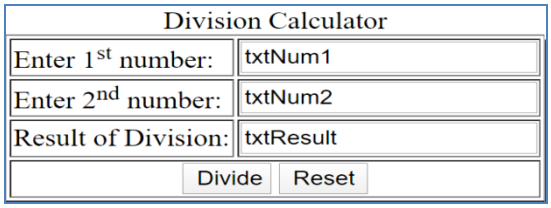
Writing Custom Message into Trace: You can append custom trace information to trace in an ASP.NET page or to the trace log. You can write trace information by using the TraceContext class's Warn or Write methods and the difference between the two methods is that a message written with the Warn method appears in red color and Write method in black color. To test this write the following statements in the button click Event Handler of "PageEvents.aspx.cs" file:

```
Trace.Write("Writing data into trace using Write method.");
Trace.Warn("Writing data into trace using Warn method.");
```

Exception Handling in Web Applications

Whenever a runtime error occurs in a program, CLR will internally create an instance of Exception class which is matching with the Exception that got occurred and throws that instance, which will now cause abnormal termination of the program and displays an error message describing the reason for termination.

When any runtime error occurs in a WebPage immediately details of that error will be show to end users on the browser screen and to check that add a new WebForm under project naming it as "ExceptionHandling.aspx" and design it as following:




```
<table align="center">
  <caption>Division Calculator</caption>
  <tr>
    <td>Enter 1<sup>st</sup> numbger:</td>
    <td><asp:TextBox ID="txtNum1" runat="server" /></td>
  </tr>
  <tr>
    <td>Enter 2<sup>nd</sup> number:</td>
    <td><asp:TextBox ID="txtNum2" runat="server" /></td>
  </tr>
  <tr>
    <td>Result Obtained:</td>
    <td><asp:TextBox ID="txtResult" runat="server" /></td>
  </tr>
  <tr>
    <td colspan="2" align="center">
      <asp:Button ID="btnDivide" runat="server" Text="Divide" />
      <asp:Button ID="btnReset" runat="server" Text="Reset" />
    </td>
  </tr>
</table>
```

Now go to "aspx.cs" file and write the following code:

Code under Page_Load:

```
if (!IsPostBack) {
  txtNum1.Focus();
}
```

Code under Divide Button Click:

```
int num1 = int.Parse(txtNum1.Text);
int num2 = int.Parse(txtNum2.Text);
```

```
int result = num1 / num2;
txtResult.Text = result.ToString();
```

Code under Reset Button Click:

```
txtNum1.Text = txtNum2.Text = txtResult.Text = "";
txtNum1.Focus();
```

Now run the Web Page with out debugging i.e. Ctrl + F5 and check the output and here we have a chance of getting 3 types of Exceptions like FormatException, DivideByZeroException and OverflowException when we enter wrong values in the TextBox's and when ever we get the Exception it will abnormally terminate the program and shows an "Yellow Screen" displaying the details of Exception. The "Yellow Screen" which is shown has a problem i.e. it will display the source code of the line where we got error and along with that it also displays 2 lines of code before and 2 lines of code after the error line and the problem here is if those lines contain any complex logic, end users can view that logic and to avoid displaying that "Yellow Screen" to users we are provided with 3 Error Handling options in ASP.Net, those are:

1. Block Level Error Handling
2. Page Level Error Handling
3. Application Level Error Handling

Block Level Error Handling: this is performed by using the very traditional structured error handling technique i.e. try and catch blocks and when code is enclosed under these blocks exceptions get handled, stopping abnormal termination. To handle error we need to use try and catch blocks as following:

```
try {
  -Statements which will cause any runtime errors.
  -Statements which should not execute when the error got occurred.
}
catch(<exception class name><var>) {
  -Statements which should execute only when the error got occurred.
}
```

To test "block level error handling" re-write the code under the divide button of our previous page as following:

```
try {
  int num1 = int.Parse(txtNum1.Text);
  int num2 = int.Parse(txtNum2.Text);
  int result = num1 / num2;
  txtResult.Text = result.ToString();
}
catch (Exception ex) {
  Response.Redirect("ErrorPage.aspx?ErrorMessage=" + ex.Message);
}
```

Now add a new WebForm under the project naming it as "ErrorPage.aspx" and write the following code under its tag:

```
<h1 style="background-color: yellowgreen; color: orangered; text-align: center">Error Page</h1>
<font size="4">There is an error in the application and the details of the error are:
  <asp:Label ID="lblMsg" runat="server" ForeColor="Red" />
</font>
```

Now go to "ErrorPage.aspx.cs" file and write the following code under Page_Load method:

```
lblMsg.Text = Request.QueryString["ErrorMessage"];
```

After doing all the above, run "ExceptionHandling.aspx" page again and now when ever any Exception occurs in the page with out displaying the "Yellow Screen" it will redirect to "ErrorPage.aspx" and displays the error details on that page.

Note: The drawback of "Block Level Error Handling" is we need to add these try and catch blocks under each and every method where there is a chance of getting any exception and it is a tedious and lengthy process, so to overcome this problem use "Page Level Error Handling".

Page Level Error Handling: this is a process of handling errors that occur in the whole page by using an Event Handler method i.e. "Page_Error" and by implementing this method under the class any error that occur in the page can be handled. When this method is implemented under our page class, where ever the error occurs in the page, control will directly transfer to this method, so we need to implement logic for handling errors under this method.

To test using the above process, first delete try and catch blocks we have added to the logic under Divide button click event handler and add the following event handler method to the class:

```
protected void Page_Error(object sender, EventArgs e) {
  Exception ex = Server.GetLastError();
  Server.ClearError();
  Response.Redirect("ErrorPage.aspx?ErrorMessage=" + ex.Message);
}
```

Now run the Web Page again and watch the output which will be same as previous but the only difference is here we can handle all errors of a page with single method with out writing multiple try and catch blocks.

Application Level Error Handling: in page level error handling we need to implement a separate Page_Error method for each Web Page, so when we have multiple Web Pages the process becomes tedious and lengthy. To avoid this problem we are provided with Application Level Error Handling which can be performed by implementing an Event Handler method known as Application_Error.

Application_Error Event Handler is capable of handling exceptions that occur under any page of the site and this method should be implemented under a special class known as "Global". We find this class under our project in a file "Global.asax" and this file will be present under every Web Application project by default. To view the Global class open the Global.asax file present under our Web Application project and there we find the Global class inheriting from a pre-defined class "HttpApplication" of System.Web namespace and by default the class contains an Event Handler method with the name Application_Start.

To test the process of application level error handling, first comment the "Page_Error" Event Handler method we defined earlier, now open Global.asax file, rename the method "Application_Start" present in the class as "Application_Error" and write the following code in it:

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

```
  Exception ex = Server.GetLastError();
  Server.ClearError();
  if(ex.InnerException != null) {
    Response.Redirect("ErrorPage.aspx?Message=" + ex.InnerException.Message);
  }
  else {
    Response.Redirect("ErrorPage.aspx?Message=" + ex.Message);
  }
}
```

In the above case when an error occurs in a Web Page and was not handled over there, server will raise another Exception there i.e. "HttpUnhandledException" and then control is transferred to Application_Error event handler, so here "ex.Message" returns the error message associated with HttpUnhandledException class but not of the actual fired Exception, so to get the error message of the actual fired Exception we need to use the statement "ex.InnerException.Message".

Sending mails to "Customer Support" team when an Exception occurs: in the above Web Page when any Exception occured we are redirecting to "ErrorPage" and displaying the error message over there, but some errors can't be fixed by the users directly and may require the "Technical Team" to fix those errors, and in such cases our application should report those errors to the "Customer Support" of the site or directly to "Technical Team" and to do that an mail should be generated by the Web Page with all the error details and send to them.

Displaying data present under Products.xml of our project in a GridView control on a Web Page:

Add a new WebForm naming it as "DisplayProducts.aspx" and the write the following code under tag:

```
<asp:GridView ID="GridView1" runat="server" />
```

Now go to "DisplayProducts.aspx.cs" file and write the following code under Page_Load Method by importing System.Data namespace.

```
string Physicalpath = Server.MapPath("~/Products.xml");
DataSet ds = new DataSet(); ds.ReadXml(Physicalpath);
GridView1.DataSource = ds; GridView1.DataBind();
```

Now run the Web Page and watch the output, which displays the Products data in a table format. To do that DataSet will first load data from the specified Xml File but in this case if it could not locate the Xml File in the specified location then "FileNotFoundException" will occur and jumps to our "Application_Error" event handler method we have implemented earlier and from there it jumps to "ErrorPage.aspx" to display the "Error Message" to end users. To test this process go to the physical location where "Products.xml" is present, move the file to another location, run the Web Form again and watch the output. The drawback in this approach is "Error Page" will display all the details of error to end users along with the path of "Xml File" which is missing and this information is no way required to end users as he can't fix the problem. To overcome this problem the information has to be sent to "Customer Support" or "Techinal Team" so that they can resolve the problem and we can do that by generating a mail and sending them.

To send a mail lets write the code under a class and use it where ever it is required and to do that add a new class under the project naming it as MailSystem.cs and write the following code in it:

```
using System.Net; using System.Text; using System.Net.Mail;
```

```
public class MailSystem {
  public static void SendMail(Exception ex, string PageName) {
    StringBuilder mailBody = new StringBuilder();
    mailBody.Append("Hello Team,");
    mailBody.Append("<br /><br />");
    mailBody.Append("Client is facing an error in output of the page: " + PageName);
    mailBody.Append("<br /><br />");
    mailBody.Append("<font color='red' size='4'>");
    mailBody.Append(ex.GetType() + ": " + ex.Message );
    mailBody.Append("</font>");
    mailBody.Append("<br /><br />");
    mailBody.Append("<font color='green' size='4'>");
    mailBody.Append("Note: Please look into the problem and try to resolve it ASAP.");
    mailBody.Append("</font>");
    mailBody.Append("<br /><br />");
    mailBody.Append("Regards");
    mailBody.Append("<br /><br />");
    mailBody.Append("Customer Support");

    MailMessage mailMsg = new MailMessage("Nithelpcenter1234@gmail.com", "m.bangarraju@gmail.com");
    mailMsg.Subject = "Error Mail";
    mailMsg.Body = mailBody.ToString();
    mailMsg.IsBodyHtml = true;

    SmtpClient client = new SmtpClient("smtp.gmail.com", 587);
    client.Credentials = new NetworkCredential("Nithelpcenter1234@gmail.com", "Hello123$$");
    client.EnableSsl = true;
    client.Send(mailMsg);
  }
}
```

Add a new WebForm naming it as "ErrorPageWithMail.aspx" and write the following code under its tag:

```
<h1 style="background-color: yellowgreen; color: orangered; text-align: center">Error Page</h1>
<p>There is an error in the application and our technical team is looking into the error and we will fix it as early as
                                        possible, so please try after some time.</p>
<h4>Any queries please feel free to contact our customer support.</h4>
```

Now go to DisplayProducts.aspx.cs file and write the following code under the class:

```
protected void Page_Error(object sender, EventArgs e) {
  Exception ex = Server.GetLastError();
  Server.ClearError();
  MailSystem.SendMail(ex, "DisplayProducts.aspx");
  Response.Redirect("ErrorPageWithMail.aspx");
}
```

After doing all the above when ever an exception occurs in the page, control jumps to Page_Error Event Handler, sends an error mail to Customer Support and redirects to "ErrorPageWithMail.aspx".

Storing Mail Settings in Web.config file: in the above case to send a mail we have entered all the details for sending the mail like Sender Id, Password, Smtp Host and Port No. in our class, but if in any case if those details change there will not be any chance of modification after hosting the site because source code will not be available, so to overcome that problem we need to store all those values in Web.config file so that SmtpClient class will read those details and sends the mail. To test this process open Web.config file and write the following code under the tag <configuration></configuration> as following:

```
<system.net>
  <mailSettings>
    <smtp deliveryMethod="Network">
      <network host="smtp.gmail.com" port="587" enableSsl="true" userName="Enter Your User Id here"
                                        password="Enter Your Password Here" />
    </smtp>
  </mailSettings>
</system.net>
```

Now go to MailSystem class and replace the last 4 lines of code in SendMail method with the below 2 lines:

```
SmtpClient client = new SmtpClient();
client.Send(mailMsg);
```

Note: to send mails from your gmail account first you need to go to "Manage your Google Account" => Select Security Tab => scroll down to "Signing in to Google" => under that set the "2-Step Verification" value to "Off", scroll down to "Less secure app access" and set it as "On".

Custom Errors

This is another process of handling errors in an Asp.Net Web Applications apart from the 3 options we have learnt above. In this process we handle errors in the Web Page thru their "Http Status Codes" using "Web.config" file. Every action we perform in a Web Application returns some result, which is informed to the client thru a code known as Http Status Codes.

Status Code	Description
200	Ok => Success
400	Bad Request => The request cannot be fulfilled due to bad syntax
403	Forbidden => The request was a legal request, but the server is refusing to respond to it
404	Not Found => Resource Not Found
405	Method not allowed => Request made to page using a request method not supported by page
408	Timeout => Server time out
500	Internal Server Error => Exception

Note: Custom Errors work only when we don't have block level or page level or application level error handling.

To test Custom Errors add a new Web Page under the project naming it as "CustomErrors.aspx" and write the following code under " " tag:

```
<h1 style="background-color: yellowgreen; color: orangered; text-align: center">Custom Error Page</h1>
<p style="text-align: justify; text-indent: 50px">There is an un-fortunate error in the application and we are aware of that problem. Our technical team is looking into the problem, so please try after some time.</p>
<h4>Any queries please feel free to contact our customer support.</h4>
```

Now open Web.config file and write the following code under <system.web> tag:

```
<customErrors mode="On" defaultRedirect="CustomErrors.aspx" />
```

Now if we get any error in the application of any "Http Status Code" then it will redirect to "CustomErrors.aspx" page whereas if we want to redirect to a different page based on the status codes i.e. if any Internal Server Error with Status Code 500 then redirecting to 1 page, Page Not Found with Status Code 404 then redirecting to 1 page can also be performed by creating different Error Pages and specifying that information in Web.config file.

To test the process, add 2 new WebForms under the Project naming them as "PageNotFound.aspx" and "InternalServerError.aspx" and write the following code under their tag:

PageNotFound.aspx:

```
<h1 style="background-color: yellowgreen; color: orangered; text-align: center">Page Not Found Error</h1>
<p style="text-align: justify">Either the requested page is not available or moved to a different location.</p>
```

InternalServerError.aspx:

```
<h1 style="background-color: yellowgreen; color: orangered; text-align: center">Internal Server Error</h1>
<p style="text-align: justify">There is an internal server error (Exception) occured in the requested page, so please contact customer support team for any additional help.</p>
```

Now re-open Web.config file and re-write the <customErrors> elements under <system.web> tag as following:

```
<customErrors mode="On" defaultRedirect="CustomErrors.aspx">
  <error statusCode="404" redirect="PageNotFound.aspx" />
  <error statusCode="500" redirect="InternalServerError.aspx" />
</customErrors>
```

Now for all Internal Server Errors (Exception) it will redirect to "InternalServerError.aspx", in case requested Page is not found it will redirect to "PageNotFound.aspx" and for rest of all other errors it will redirect to "CustomErrors.aspx".

Mode Attribute of CustomErrors element: mode attribute of CustomErrors element can be set with 3 different values: On, Off and RemoteOnly (default value).

On => in this case end users and developers will see Custom Error Pages without being displayed with detailed error messages i.e., "Yellow screen of death".
Off => in this case detailed error messages (Yellow screen of death) will be shown to the end users and developers also, but not Custom Error Pages.
RemoteOnly => in this case detailed error messages (Yellow screen of death) is shown to local users (developers) whereas end users will receive Custom Error Pages.

Globalization and Localization

Globalization is an approach of developing an application neutral of a culture and Localization is an approach of translating a globalized application to a particular culture.

Culture: It's a set of standards followed by a country regarding Date and Time Format, Calendar System, Currency, Number System, Measuring System, Writing System, Sorting Order and Language. Every country follows or adopts a different culture and representing the culture of a country there was a culture code or culture name which will be as following: **en-US, en-GB, fr-FR, ja-JP, hi-IN, te-IN, ta-IN, mr-IN, or-IN, etc.**

Note: in culture name the first 2 characters in lower case represents the language and the next 2 characters in upper case represents the country.

By default, every applications runs in American culture i.e., "en-US" and to test this add a new WebForm naming it as "TestCulture.aspx" and write the following code under tag:

```
<asp:Label ID="lblDate" runat="server" Text="Select Date: " />
<asp:TextBox ID="txtDate" runat="server" />
<asp:ImageButton ID="imgDate" runat="server" ImageUrl="~/Icons/Calendar.ico" Width="20" Height="20"
                                                ImageAlign="AbsMiddle" />
<asp:Calendar ID="cldDate" runat="server" Visible="false" />
```

Now go to aspx.cs file and write the following code in it:

Code under Image Button Click:

```
if (cldDate.Visible) {
  cldDate.Visible = false;
}
else {
  cldDate.Visible = true;
}
```

Code Under Calender Selection Changed:

```
txtDate.Text = cldDate.SelectedDate.ToShortDateString();
cldDate.Visible = false;
```

Now when we run the application and select a date from Calendar then selected date is displayed in TextBox, but if we notice the format of date is "mm/dd/yyyy" and also calendar control displays month names and week names in English which proves that our application is running in "American-English" culture.

We can change the culture of our Web Application by setting Culture attribute to the Page Directive with the Culture Name we want as following:

```
<%@ Page Language="C#" Culture="hi-IN" … %>
```

Now when we run the application selected date will be in "dd-mm-yyyy" format and also calendar control displays month names and week names in hindi because we set the culture as "India-Hindi", but the draw back in this approach is, the culture we set right now applies to all users accessing the application from any where,

whereas if we want to set the culture differently for each user i.e., based on his browser preferred language settings, then set the culture in page directive as following:

<%@ Page Language="C#" Culture="Auto" … %>

Note: In the above case based on browser preferred language settings culture will vary for each user and to test this change the preferred language settings of your browser to any specific language.

How the Culture of an application changes based on Culture attribute of Page Directive?

Ans: In our Liraries under the System.Globalization namespace we are provided with a class known as "CultureInfo" and this class contains the information of all the available cultures in the industry and their culture standards. When the instance of CultureInfo class is created by passing CultureName as "Constructor Parameter" will take the responsibility of changing the culture.

System.Globalization.CultureInfo(string CultureName)

When we specify the culture attribute for page directive with any static value like "hi-IN" or "fr-FR" or "de-DE" then internally instance of CultureInfo class is created by taking the culture name as a parameter to its constructor, so that culture of our Thread changes, whereas when the culture attribute value is set as "Auto" then server will first read default language settings of browser and creates the instance of CultureInfo class with the Culture Name it has read and changes the Thread culture.

Thread: this is a unit of execution which is responsible to serve a client request i.e., whenever a client sends his request to server, server will create an instance of Thread class and gives that Thread to client for serving his requests. So, if we have "n" users connected to the server there will also be "n" Threads created.

Note: CultureInfo class is responsible in changing all the attributes of culture like Date and Time Format, Calendar System, Number System, Currency, Measuring System, Writing System & Sorting Order but not **Language**, and this is the reason why in our above Page Calendar is changing it's culture, but the text "Select Date" we have used for "Label" is not changing i.e. it always shows Text in English only, because language translations are never performed implicitly as we are not provided with any proper language translation tools in the industry, so as a developer it is purely our responsibility to perform language translations for each and every language we wanted to support.

Developing a Multi-Lang Registration Form: Add a new WebForm naming it as "MultiLangRegistrationForm.aspx" and write the following code in tag:

```
<table align="center">
 <tr>
  <td colspan="2" align="center"><asp:Label ID="lblTitle" runat="server" Text="Registration Form" /></td>
 </tr>
 <tr>
  <td colspan="2" align="center">
   <asp:DropDownList ID="ddlLang" runat="server" AutoPostBack="True">
    <asp:ListItem Text="English" Value="en" />
    <asp:ListItem Text="français" Value="fr" />
    <asp:ListItem Text="हिंदी" Value="hi" />
    <asp:ListItem Text="తెలుగు" Value="te" />
   </asp:DropDownList>
  </td>
```

```
 </tr>
 <tr>
  <td><asp:Label ID="lblUser" runat="server" Text="User Id:" /></td>
  <td><asp:TextBox ID="txtUser" runat="server" /></td>
 </tr>
 <tr>
  <td><asp:Label ID="lblPwd" runat="server" Text="Password:" /></td>
  <td><asp:TextBox ID="txtPwd" runat="server" TextMode="Password" /></td>
 </tr>
 <tr>
  <td><asp:Label ID="lblName" runat="server" Text="Name:" /></td>
  <td><asp:TextBox ID="txtName" runat="server" /></td>
 </tr>
 <tr>
  <td><asp:Label ID="lblPhone" runat="server" Text="Phone No:" /></td>
  <td><asp:TextBox ID="txtPhone" runat="server" /></td>
 </tr>
 <tr>
  <td><asp:Label ID="lblEmail" runat="server" Text="Email Id:" /></td>
  <td><asp:TextBox ID="txtEmail" runat="server" TextMode="Email" /></td>
 </tr>
 <tr>
  <td><asp:Label ID="lblAddress" runat="server" Text="Address:" /></td>
  <td><asp:TextBox ID="txtAddress" runat="server" TextMode="MultiLine"/></td>
 </tr>
 <tr>
  <td align="center" colspan="2"><asp:Button ID="btnRegister" runat="server" Text="Register"  />
                    <asp:Button ID="btnReset" runat="server" Text="Reset" /></td>
 </tr>
</table>
```

In the above case we wanted to support 4 different languages for our registration form and to do that we need to enter “Text” for labels and buttons in 4 languages explicitly and that should be done under a special file known as “Resource File”.

Resource files are used for storing information specific to each language we wanted to support in a “Name/Value” pair. We need to create these resource files separate for each language that we wanted to support and store information specific to that language. Resource files are saved with “.resx” extension and while naming resource files we need to follow a pattern i.e., “<Base Name>.<Culture>.resx” where “<Base Name>” should be same for all languages and “<Culture>” varies for each language.

By default, Visual Studio provides support for generating a resource file for English language with the existing “Text” of controls and based on that we can generate resource files for remaining other languages also. To generate resource file for English go to design view of the “.aspx” page, go to Tools Menu and select the option “Generate Local Resource” and this action will perform the following:

1. Adds a special Asp.Net Folder with the name “App_LocalResources” under the project and under this folder we find a resource file that is created for english language with the name “MultLangRegistrationForm.aspx.resx” and here “.resx” is the file extension, “MultiLangRegistrationForm.aspx” is the "<Base Name>" and for english language “<Culture>” is optional.
2. For every aspx controls and page directive it adds a key for referring to that element in the resource file which will be in the following pattern:

   meta:resourcekey="PageResource1" => For Page Directive
   meta:resourcekey="lblTitleResource1" => For Title Label
   meta:resourcekey="ddlLangResource1" => For DropDownList

Note: in the above case PageResource1, lblTitleResource1 and ddlLangResource1 are the keys that are used in the resource file for referring to that control.

3. For Page directive we will also find “Culture & UI Culture” attributes with a value “Auto” for changing the Culture and UI Culture of the page based on user’s preferred language settings.
4. Now if we open the Resource File and observe we notice keys referring to controls and with that key it will refer to Text and ToolTip of that control and for Page it will refer to Title.

Note: Currently resource file is having keys referring to all controls so delete entries for DropDownList, List Items and TextBox controls also (both Text and ToolTip) because we require keys for Labels and Buttons only but those keys refer to Text and ToolTip of each control, but we require only Text entries so delete all ToolTip entries and finally we need only 10 entries to be present i.e. Text of 7 Labels, 2 buttons and Title of Page.

Creating Resource Files for other Languages: open solution explorer, right click on the existing resource file, select copy, right click on “App_LocalResources” folder and select paste; repeat the same process for 2 more times to generate 4 resource files totally under the folder and rename the 3 new resource files as following:

- MultiLangRegistrationForm.aspx.fr.resx
- MultiLangRegistrationForm.aspx.hi.resx
- MultiLangRegistrationForm.aspx.te.resx

Now open each resource file, copy content under “Value” which is present in English, use google translator, translate values from English into French, Hindi, and Telugu languages respectively, and paste those translated values in place of English values under the corresponding resource files so that each resource file should be as following:

MultiLangRegistrationForm.aspx.resx

Name	Value
lblTitleResource1.Text	Registration Form
lblUserResource1.Text	User Id:
lblPwdResource1.Text	Password:
lblNameResource1.Text	Name:
lblPhoneResource1.Text	Phone No:
lblEmailResource1.Text	Email Id:
lblAddressResource1.Text	Address:
btnRegisterResource1.Text	Register
btnResetResource1.Text	Reset
PageResource1.Title	Registration Form - English

MultiLangRegistrationForm.aspx.fr.resx

Name	Value
lblTitleResource1.Text	Formulaire d'inscription
lblUserResource1.Text	Identifiant d'utilisateur:
lblPwdResource1.Text	Mot de passe:
lblNameResource1.Text	Prénom:
lblPhoneResource1.Text	Pas de téléphone:
lblEmailResource1.Text	Identifiant de messagerie:
lblAddressResource1.Text	Adresse:
btnRegisterResource1.Text	S'inscrire
btnResetResource1.Text	Réinitialiser
PageResource1.Title	Formulaire d'inscription – français

MultiLangRegistrationForm.aspx.hi.resx

Name	Value
lblTitleResource1.Text	पंजीकरणफॉर्म
lblUserResource1.Text	यूज़रआईडी:
lblPwdResource1.Text	पारणशब्द:
lblNameResource1.Text	नाम:
lblPhoneResource1.Text	फोननंबर:
lblEmailResource1.Text	ईमेलआईडी:
lblAddressResource1.Text	पता:
btnRegisterResource1.Text	रजिस्टर
btnResetResource1.Text	रीसेट
PageResource1.Title	पंजीकरणफॉर्म – हिंदी

MultiLangRegistrationForm.aspx.te.resx

Name	Value
lblTitleResource1.Text	సభ్యత్వనమోదుపత్రం
lblUserResource1.Text	వినియోగదారునిగుర్తింపు:
lblPwdResource1.Text	పాస్‌వర్డ్:
lblNameResource1.Text	పేరు:
lblPhoneResource1.Text	చరవాణి:
lblEmailResource1.Text	ఇమెయిల్:
lblAddressResource1.Text	చిరునామా:
btnRegisterResource1.Text	నమోదు
btnResetResource1.Text	రీసెట్
PageResource1.Title	సభ్యత్వనమోదుపత్రం – తెలుగు

Note: once we do the above automatically our page language changes based on browser preferred language settings and to test that run the WebPage and watch the output by changing the preferred language under browser.

Even if the text of Labels and Buttons changes based on browser's preferred language settings, the language in DropDownList will be still showing the first item in the list i.e., "English" only and if we want this to be set according to the browser preferred language, then write the following code under "Page_Load" Event Handler by reading "Http_Accept_Language" from Server Variables Collection as following:

```
if (!IsPostBack) {
  ddlLang.SelectedValue = Request.ServerVariables["Http_Accept_Language"].Substring(0, 2);
}
```

Now if we want to change the “Text” based on DropDownList selection then write the following code in “aspx.cs” file:

```
using System.Threading; using System.Globalization;
protected override void InitializeCulture() {
  string CultureName = Request.Form["ddlLang"];
  if (CultureName != null) {
    CultureInfo ci = new CultureInfo(CultureName);
    Thread.CurrentThread.CurrentCulture = ci;
    Thread.CurrentThread.CurrentUICulture = ci;
  }
}
```

The InitializeCulture method of Page class contains no logic in it and developers extending the functionality of the Page class can override the InitializeCulture method to set the Culture and UICulture information for a Thread associated with the client.

CurrentThread is a static property of Thread class which returns the reference of Thread associated with current client because every client request is handled by a seperate Thread. CurrentCulture property of Thread class gets or sets the culture for the current thread and CurrentUICulture property gets or sets the current culture used by the ResourceManager to look up culture-specific resources at run time.

State Management

Web Applications are stateless i.e., we can never access the values of 1 request, in the next request of the same page or other pages also. But sometimes we need the values of 1 request, in the next request to same page or other pages and to overcome this problem and maintain the state of values between multiple requests to the same page or between different pages we are provided with the concept called as State Management.

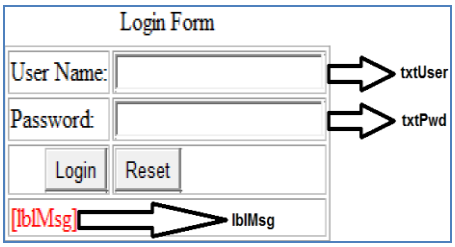
In ASP.Net to maintain the state of values we are provided with various techniques like:

1. View State
2. Query String
3. Hidden Field
4. Cookie
5. Session
6. Application

To understand State Management first open a new ASP.NET Web Application project, name it as "ASPStateMgmt", specify the location to save as our personal folder, click ok and in the window opened choose "Empty Project Template", check "Web Forms CheckBox" and click on Create Button. Now let's host the project under Local IIS and to do that open Solution Explorer and under the project - double click on "Properties" node which opens the Project Property window, click on the "Web" in LHS and now on the right, under "Servers" choose "Local IIS" option in the "DropDownList", click on "Create Virtual Directory" button, and click on "Save" icon in Visual Studio Toolbar.

Option 1: ViewState is a mechanism to preserve the values of the Page and Controls between Post backs. It is a Page-Level State Management technique i.e., it maintains the state of values for a single user between multiple requests of 1 page (**Single User-Local Data**).

Creating a Login Page with 3 failure attempts: Add 3 new Web Forms under the project naming them as "LoginWithFailureCount.aspx", "SuccessWithFailureCount.aspx" and "FailureWithFailureCount.aspx", and design "LoginWithFailureCount.aspx" as below:




```
<table align="center">
  <caption>Login Form</caption>
  <tr>
    <td>User Name:</td>
    <td><asp:TextBox ID="txtUser" runat="server" /></td>
  </tr>
  <tr>
    <td>Password:</td>
    <td><asp:TextBox ID="txtPwd" runat="server" TextMode="Password" /></td>
  </tr>
```

```
  <tr>
    <td colspan="2" align="center">
      <asp:Button ID="btnLogin" runat="server" Text="Login" />
      <asp:Button ID="btnReset" runat="server" Text="Reset" />
    </td>
  </tr>
  <tr>
    <td colspan="2">
      <asp:Label ID="lblMsg" runat="server" ForeColor="Red" />
    </td>
  </tr>
</table>
```

Now go to LoginWithFailureCount.aspx.cs file and write the below code:

Code under Page_Load Event Handler:

```
if (!IsPostBack) {
  txtUser.Focus();
  ViewState["FailureCount"] = 0;
}
```

Code under Login Button Click Event Handler:

```
if (txtUser.Text == "admin" && txtPwd.Text == "admin") {
  Response.Redirect("SuccessWithFailureCount.aspx?Name=" + txtUser.Text);
}
else {
  int Count = (int)ViewState["FailureCount"] + 1;
  if(Count == 3) {
    Response.Redirect("FailureWithFailureCount.aspx?Name=" + txtUser.Text + "&Count=" + Count);
  }
  ViewState["FailureCount"] = Count;
  lblMsg.Text = Count + " attempts failed and the maximum is 3.";
}
```

Code under Reset Button Click Event Handler:

```
txtUser.Text = txtPwd.Text = "";
txtUser.Focus();
```

Now go to SuccessWithFailureCount.aspx.cs file and write the below code:

Code under Page_Load Event Handler:

```
string Name = Request.QueryString["Name"];
Response.Write("Hello " + Name + ", welcome to the site.");
```

Now go to FailureWithFailureCount.aspx.cs file and write the below code:

Code under Page_Load:

```
string Name = Request.QueryString["Name"];
int Count = int.Parse(Request.QueryString["Count"]);
```

Response.Write("Hello " + Name + ", you have failed all the " + Count + " attempts to login.");

Now run the "LoginForm", enter valid credentials, and click on Login button which will redirect to "SuccessPage" and if the credentials are wrong it will give a chance for another 2 times to correct the values and if at all the 3 attempts fail then it will redirect to "FailurePage".

In our above code to maintain the FailureCount we have used ViewState because Web Applications are Stateless and ViewState is used for maintaining the state of values, which holds values of a page between multiple requests of that page for a single user i.e., **"Single User-Local Data"**.

Where does Web Application store ViewState Values?

Ans: Web Applications, store ViewState values on the same page only i.e., while rendering the output of a page it writes ViewState values into that rendered output and sends them to the browser, so we can view those values using "View Page Source" option of the browser and there we find a "Hidden Field" with the name "__ViewState" that contains ViewState values stored in "Base64 Encrypted String" format. Next time when the page is submitted to the Server all the ViewState values are carried to the Server and there Server will read and decrypt those values.

Drawbacks of ViewState:

1. ViewState values are transported between Browser and Server every time we perform a Post Back, so in case if we store huge volumes of data in ViewState then transporting all that data between Browser and Server will increase the network traffic.
2. As discussed above ViewState values are accessible only with the Same Page and that is the reason why we have explicitly transported "Name & Failure Count" values to "FailureWithFailureCount.aspx" as a Query String, because in the second page we can't access first page ViewState values.

Disabling ViewState in a WebPage: It's possible to disable ViewState either for a particular Web Page or for the whole Application also which can be done in 2 different ways:

Page Level Disabling: by setting "EnableViewState" attribute of "Page Directive" value as "false" we can disable ViewState values for a particular page. To test this, go to "LoginWithFailureCount.aspx" and add "EnableViewState" attribute to "Page Directive" as following:

```
<%@ Page EnableViewState="false" ... %>
```

Note: now if we run the WebForm we get "NullReferenceException" because ViewState is disabled.

Application Level Disabling: We can also disable ViewState application level i.e., under the Web.config file so that ViewState will not work in any page and to do that write the following code under <system.web> tag of Web.config file.

```
<pages enableViewState ="false" />
```

Note: Control State is turned "On" by default for every control on the page regardless of whether it is used during a post-back or not and serializes the data, and we can't turn off the Control State of controls.

Option 2: Query Strings this is another option by using which we can maintain the state of required values by concatenating those values to page URL as we have performed in our previous example.

Syntax: PageUrl?Key=Value&Key=Value&.....&Key=Value

For example, in our previous pages we have used query strings to transfer values to other pages as following:

Response.Redirect("SuccessWithFailureCount.aspx?Name=" + txtUser.Text);

```
Response.Redirect("FailureWithFailureCount.aspx?Name=" + txtUser.Text + "&Count=" + Count);
```

Drawbacks of Query Strings:

1. By using this we can pass values only to a particular page and that to when we are transferring control either by using Server.Transfer or Response.Redirect methods.
2. We can't carry more volumes of data by using a Query String because it supports only 2048 bytes of data.
3. Passing values by Query Strings is not secured because those values are visible in the Address bar of browser.

Option 3: Hidden Fields By using Hidden Field control also, we can maintain the state of values within a particular page and more over as discussed earlier ViewState maintains its values thru Hidden Fields only. So, without using ViewState we can use Hidden Fields and maintain the state of values on our page.

Hidden Fields Vs ViewState:

1. In case of ViewState, data is secured because it is stored in a "Base64 Encrypted String" format whereas Hidden Field values are not encrypted, and if encryption is required, we need to explicitly perform it by implementing our own logic.
2. ViewState values are not accessible to other pages whereas we can read Hidden Field values on the page where we are submitting, by using Request object.

Add 2 new Web Forms in the project naming them as "HitCountWithHiddenField.aspx" and "NewForm.aspx" and write the following code under " " tag of 1st form:

```
<asp:Button id="Button1" runat="server" Text="Hit Count with View State" />
<asp:Label ID="Label1" runat="server" ForeColor="red" />
<br />
<asp:Button id="Button2" runat="server" Text="Hit Count with Hidden Field" />
<asp:HiddenField ID="hfCount" runat="server" Value="0" />
<asp:Label ID="Label2" runat="server" ForeColor="red" />
<br />
<asp:Button ID="Button3" runat="server" Text="Launch New Form" PostBackUrl="~/NewForm.aspx" />
```

Write the below code under Page_Load Event Handler of HitCountWithHiddenField.aspx:

```
if(!IsPostBack) {
  ViewState["HitCount"] = 0;
}
```

Write the below code under "Hit Count with View State" button Click Event Handler of 1st form:

```
int Count = (int)ViewState["HitCount"] + 1;
ViewState["HitCount"] = Count;
Label1.Text = "Hit Count with View State: " + Count;
```

Write the below code under "Hit Count with Hidden Field" button Click Event Handler of 1st form:

```
int Count = int.Parse(hfCount.Value) + 1;
hfCount.Value = Count.ToString();
Label2.Text = "Hit Count with Hidden Field: " + Count;
```

Write the below code under Page_Load of NewForm.aspx:

```
int Count = int.Parse(Request.Form["hfCount"]);
```

```
Response.Write($"Value of Hidden Field is: {Count}");
```

Now run the 1st form, click on the 2 Hit Count buttons, and allow the Count value to increment and at any point of time go to "View Page Source" option on browser and you can see the Hidden Field value but not ViewState value i.e., it will be in encrypted format. Now click on "Launch New Form" button which will display the "Count" value of previous page stored in Hidden Field on the new page.

Option 4: Cookies A Cookie is a small piece of text that is used to store user-specific information and that information can be read by the Web Application whenever user visits the site. When a user requests for a Web Page, Web Server sends not just a page, but also a cookie containing the date and time. Cookies are stored in a folder on the user's hard disk and when the user requests for the Web Page again, browser looks on the hard disk for Cookies associated with the Web Page and sends them to the Server. Browser stores cookies separately for each different site visited.

By using Cookies also, we can maintain the state of values between multiple pages for a single user. ASP.NET provides us options for both reading and writing cookies on client machines.

Writing Cookies on client machine: we can write cookies on client machine in 2 different ways, those are:

1. Create the instance of HttpCookie class; store values into it as an array and then write that cookie to client machine by using Response object.

```
HttpCookie cookie = new HttpCookie("LoginCookie");
cookie["User"] = "Raju"; cookie["Pwd"] = "admin";
Response.Cookies.Add(cookie);
```

2. Write each value to browser as an individual cookie using Response object in a name/value combination.

```
Response.Cookies["User"].Value = "Raju";
Response.Cookies["Pwd"].Value = "admin";
```

Reading Cookies back on our WebPages: we can read Cookies present on the client machine in any WebPage of our application by using Request object.

1. If we write the cookie to browser by using the 1st approach, we need to read it as following:

```
HttpCookie cookie = Request.Cookies["LoginCookie"];
string User = cookie["User"]; string Pwd = cookie["Pwd"];
```

2. If we write the cookie to browser by using the 2nd approach, we need to read it as following:

```
string User = Request.Cookies["User"].Value;
string Pwd = Request.Cookies["Pwd"].Value;
```

Cookies are or 2 types:

I. In-Memory Cookies
II. Persistent Cookies

In-Memory cookies are stored in browser's memory so once the browser is closed, immediately all the cookies that are associated with that browser window will be destroyed, and by default every cookie is In-Memory only. Persistent Cookies are stored on Hard Disk of the client machines, so even after closing the browser window

also they will be persisting and can be accessed next time we visit the site. To make a cookie as persistent we need to set "Expires" property of Cookie with a "DateTime" value.

Setting expires property of Cookie:

1. If we write the cookie to browser by using 1st approach, we need to set the expires property as following:

```
cookie.Expires = <DateTime>;
```

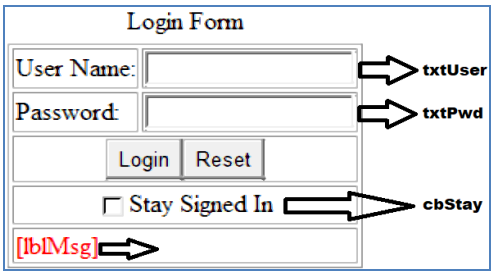
2. If we write the cookie to browser by using 2nd approach, we need to set the expires property as following:

```
Response.Cookies["user"].Expires = <DateTime>;
Response.Cookies["pwd"].Expires = <DateTime>;
```

To work with Cookies first add 3 new Web Forms under the project naming them as:

1. LoginWithStaySignedIn.aspx
2. SuccessWithStaySignedIn.aspx
3. FailureWithStaySignedIn.aspx

Design LoginWithStaySignedIn.aspx as below:




```
<table align="center">
  <caption>Login Form</caption>
  <tr>
    <td>User Name:</td>
    <td><asp:TextBox ID="txtUser" runat="server" /></td>
  </tr>
  <tr>
    <td>Password:</td>
    <td><asp:TextBox ID="txtPwd" runat="server" TextMode="Password" /></td>
  </tr>
  <tr>
    <td colspan="2" align="center">
      <asp:Button ID="btnLogin" runat="server" Text="Login" />
      <asp:Button ID="btnReset" runat="server" Text="Reset" />
    </td>
  </tr>
  <tr>
    <td colspan="2" align="center"><asp:CheckBox ID="cbStay" runat="server" Text="Stay Signed In" /></td>
  </tr>
  <tr>
    <td colspan="2"><asp:Label ID="lblMsg" runat="server" ForeColor="Red" /></td>
```

```
  </tr>
</table>
```

Now go to LoginWithStaySignedIn.aspx.cs file and write the below code:

Code under Page_Load Event Handler:

```
if(Request.Cookies["LoginCookie"] != null) {
  Response.Redirect("SuccessWithStaySignedIn.aspx");
}
if(!IsPostBack) {
  txtUser.Focus();
  ViewState["FailureCount"] = 0;
}
```

Code under Reset Button Click Event Handler:

```
cbStay.Checked = false;
txtUser.Text = txtPwd.Text = "";
txtUser.Focus();
```

Code under Login Button Click Event Handler:

```
if(txtUser.Text == "admin" && txtPwd.Text == "admin") {
  HttpCookie cookie = new HttpCookie("LoginCookie");
  cookie["User"] = txtUser.Text;
  cookie["Pwd"] = txtPwd.Text;
  if(cbStay.Checked) {
    cookie.Expires = DateTime.Now.AddDays(10);
  }
  Response.Cookies.Add(cookie);
  Response.Redirect("SuccessWithStaySignedIn.aspx");
}
else {
  int Count = (int)ViewState["FailureCount"] + 1;
  if(Count == 3) {
    Response.Cookies["User"].Value = txtUser.Text;
    Response.Cookies["Count"].Value = Count.ToString();
    Response.Redirect("FailureWithStaySignedIn.aspx");
  }
  ViewState["FailureCount"] = Count;
  lblMsg.Text = Count + " attempt(s) failed to login and maximum are 3 only.";
}
```

Now go to SuccessWithStaySignedIn.aspx.cs file and write the below code under Page_Load Event Handler:

```
if (Request.Cookies["LoginCookie"] != null) {
  string Name = cookie["User"];
  string Pwd = cookie["Pwd"];
  Response.Write("Hello " + Name + ", welcome to the site.");
}
else {
```

```
  Response.Redirect("LoginWithStaySignedIn.aspx");
}
```

Now go to FailureWithStaySignedIn.aspx.cs file and write the below code under Page_Load Event Handler:

```
if (Request.Cookies["User"] != null && Request.Cookies["Count"] != null) {
  string User = Request.Cookies["User"].Value;
  string Count = Request.Cookies["Count"].Value;
  Response.Write($"Hello {User}, you have failed all the {Count} attempts to login.");
}
else {
  Response.Redirect("LoginWithStaySignedIn.aspx");
 }
```

Now run "Login Page" and if we enter valid credentials, it will take you to "SuccessPage" and if we enter wrong credentials it will check for 3 times and will take you to "Failure Page". When valid credentials are entered with "StaySignedIn" option checked, then it will not ask for credentials when we open the "Login Page" for next time and directly takes you to "Success Page" and in this case we can directly open "Success Page" from next time whereas if the credentials or not supplied, then if we try to open "Success Page" or "Failure Page" it will redirect us back to "Login Page" because unless credentials are provided we can't open any page of the site.

Drawbacks of Cookies:

1. We can create only 50 cookies for each website, so every new cookie from the site will override the old cookie once after reaching the limit.
2. A cookie can store only 4 K.B. of data that too of type string only.
3. Cookies are not secured because they are stored on client machines.
4. Because cookies are stored on client machines there is a problem like clients can either delete the cookies or even disable cookies.

Location of Cookies:

Microsoft Edge: C:\Users\<User>\AppData\Local\Microsoft\Edge\User Data\Default
Google Chrome: C:\Users\<User>\AppData\Local\Google\Chrome\User Data\Default

Disabling cookies on Brower: Click on "…" option beside the Address bar in browser => select settings and search for "Cookies" which shows "Cookies and other site data" option and use the appropriate option to disable cookies.

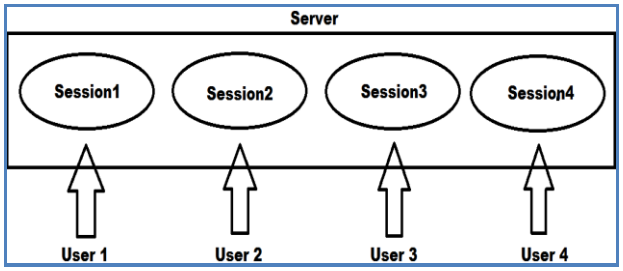
Delete Cookies on Browser: To delete cookies on our browser use the command Ctrl + Shift + Delete which will open "Clear browsing data" window, in that choose "Cookies and other site data" checkbox and click "Clear Data".

Option 5: Session

In ASP.NET session is a state that is used to store and retrieve values of a user across all the pages of site i.e., it is "Single User Global Data". It helps to identify requests from the same browser during a time (session). It is used to store value for the time. By default, ASP.NET session state is enabled for all ASP.NET applications. Each created session is stored in "SessionStateItemCollection" object. We can get current session value by using Page object's Session property which is of type "HttpSessionState".

Whenever a user sends his first request to server, server will provide a session for that user to store data that is associated with that user, giving the option of accessing that data across all the pages of site. Session

doesn't have any size limitation and more over they can store any type of data like Scalar Types and Complex Types also. Sessions are unique i.e., the session associated with one user is never accessible to other users.



Storing values into the Session:

Session[string key] = value (object)

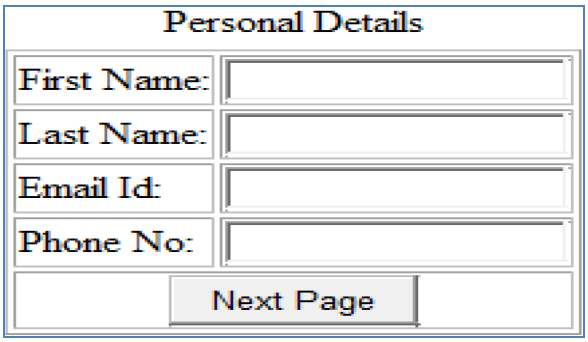
Accessing values from Session:

object value = Session[string key]

To test Sessions, add 3 new Web Forms under the project naming them as:

1. PersonalDetails.aspx
2. FamilyDetails.aspx
3. DisplayDetails.aspx

Step 1: Design PersonalDetails.aspx as following:




```
<table align="center">
  <caption>Personal Details</caption>
  <tr>
    <td>First Name:</td>
    <td><asp:TextBox ID="txtFName" runat="server" /></td>
  </tr>
  <tr>
    <td>Last Name:</td>
    <td><asp:TextBox ID="txtLName" runat="server" /></td>
  </tr>
  <tr>
    <td>Phone No:</td>
    <td><asp:TextBox ID="txtPhone" runat="server" /></td>
  </tr>
  <tr>
```

```
    <td>Email Id:</td>
    <td><asp:TextBox ID="txtEmail" runat="server" /></td>
  </tr>
  <tr>
    <td colspan="2" align="center">
      <asp:Button ID="btnNext" runat="server" Text="Next Page" />
    </td>
  </tr>
</table>
```

Now go to PersonalDetails.aspx.cs and write the below code:

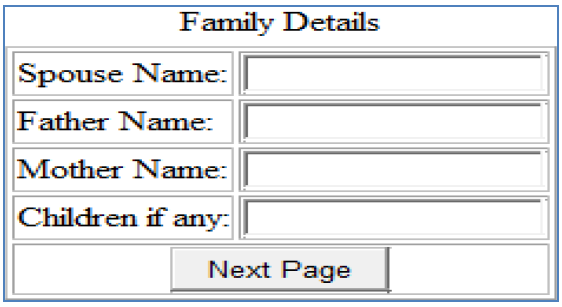
Code under Page_Load Event Handler:

```
if (!IsPostBack) {
  txtFName.Focus();
}
```

Code under Next Page Button Click Event Handler:

```
Dictionary<string, object> userDetails = new Dictionary<string, object>();
userDetails.Add("First Name", txtFName.Text);
userDetails.Add("Last Name", txtLName.Text);
userDetails.Add("Phone No", txtPhone.Text);
userDetails.Add("EMail Id", txtEmail.Text);
Session["UserData"] = userDetails;
Response.Redirect("FamilyDetails.aspx");
```

Step 2: Design FamilyDetails.aspx as following:




```
<table align="center">
  <caption>Family Details</caption>
  <tr>
    <td>Spouse Name:</td>
    <td><asp:TextBox ID="txtSName" runat="server" /></td>
  </tr>
  <tr>
    <td>Father Name:</td>
    <td><asp:TextBox ID="txtFName" runat="server" /></td>
  </tr>
  <tr>
    <td>Mother Name:</td>
```

```
    <td><asp:TextBox ID="txtMName" runat="server" /></td>
  </tr>
  <tr>
    <td>Children (if any):</td>
    <td><asp:TextBox ID="txtChildren" runat="server" /></td>
  </tr>
  <tr>
    <td colspan="2" align="center">
      <asp:Button ID="btnNext" runat="server" Text="Next Page" />
    </td>
  </tr>
</table>
```

Now go to FamilyDetails.aspx.cs file and write the below code in it:

Code under Page_Load Event Handler:

```
if (Session["UserDetails"] == null) {
  Response.Redirect("PersonalDetails.aspx");
}
else {
  txtSName.Focus();
}
```

Code under Next Page Button Click Event Handler:

```
if (Session["UserData"] != null) {
  Dictionary<string, object> userDetails = (Dictionary<string, object>)Session["UserData"];
  userDetails.Add("Spouse Name", txtSName.Text);
  userDetails.Add("Father Name", txtFName.Text);
  userDetails.Add("Mother Name", txtMName.Text);
  userDetails.Add("Children", txtChildren.Text);
  Session["UserData"] = userDetails;
  Response.Redirect("DisplayDetails.aspx");
}
else {
  Response.Redirect("PersonalDetails.aspx");
}
```

Step 3: Go to DisplayDetails.aspx.cs file and write thefollowing code under Page_Load:

```
if (Session["UserData"] != null) {
  Dictionary<string, object> userDetails = (Dictionary<string, object>)Session["UserData"];
  foreach (string Key in userDetails.Keys) {
    Response.Write(Key + ": " + userDetails[Key] + "<br />");
  }
}
else {
  Response.Redirect("PersonalDetails.aspx");
}
```

Note: In the above example 1st we are storing data of "PersonalDetails" page in to a Dictionary and then putting that Dictionary into Session and redirecting to "FamilyDetails" page, and in that page we are picking the Dictionary from Session which is stored by "PersonalDetails" page, storing "FamilyDetails" data also into the Dictionary and again putting that Dictionary into the Session which is finally captured in "DisplayDetails" page to display them in the form of "name/value" pairs.

Run "PersonalDetails.aspx" page, fill in the details, and click on the "Next Page" button which will launch "FamilyDetails.aspx" and here also fill in the details and click on the "Next Page" button to launch "DisplayDetails.aspx", which will display all the values that are captured from the first 2 pages. Now copy the URL of the "DisplayDetails.aspx" page in Address bar, open a new tab in the same browser window, paste the URL in Address bar to execute, and here also we see the values that are captured in 1st and 2nd pages, whereas if we open a new instance of a new browser and paste the URL it will not display the values but takes you to the PersonalDetails.aspx because Session values of 1 user are not shared with another user.

How is a session identified to which user it belongs to?

Ans: Whenever a Session is created for a user it is given with a Unique Id known as "SessionId" and this "SessionId" is written to client's browser in the form of a "In-Memory Cookie", so whenever the client comes back to the server in a next request, server will read the Cookie, picks the "SessionId" and associates user with his exact Session.

Note: because SessionId is stored on client's browser in the form of an "In-Memory Cookie", other tabs under the browsers instance can also access that session, whereas a new instance of a new browser can't access the Session.

How SessionId is maintained if the browser disables cookies?

Ans: if the browser disables Cookies, then Sessions are not maintained for that browser or user, and to test this disable Cookies on your Browser and run the "PersonalDetails.aspx" page, fill in the details and click on "Next Page" button but still we will be on the same page. To resolve this problem, we need to configure <sessionState> element in the "Web.config" file. <sessionState> element has an attribute known as "Cookie Less" and it can be set with different values like false(d), true and AutoDetect.

1. Default value is false which indicates that sessions can't be maintained without a cookie.
2. If set as true, then without storing the "SessionId" in the form of a cookie, server will store "SessionId" in the URL of Page, so in this case even if the browser supports Cookies also, without using them, "SessionId" is stored in URL. The problem in this approach is if we copy the URL of one instance of browser and use it under another instance of a new browser, we can access the Session values that are associated with the "SessionId" present in the URL.
3. If set as "AutoDetect" then server will first verify, whether browser supports cookies or not and if supported, "SessionId" is stored as Cookie and if not supported "SessionId" is stored in Page URL, and this is the most recommended option to use.

To try this, go to Web.config file and write the following code under <system.web> tag:

```
<sessionState cookieless="AutoDetect" />
```

Note: <sessionState> element in Web.config is used for making settings to Session and its behavior.

Now run "PersonalDetails.aspx" page again and notice the URL in Address bar which will contain "Session Id" if cookies are disabled in browser, or else we will not find "Session Id" in Page URL.

What happens to Sessions associated with clients if the client closes the browser?

Ans: Every Session will be having a "time-out" period of "20 Minutes (default)" from the last request (Sliding Expiration), so within 20 Minutes if the Session is not used by the User, Server will destroy that Session.

Note: we can change the default time-out period of "20 Mins" to our required value thru "Web.config" file by setting "timeout" attribute value of "sessionState" element as following:

```
<sessionState cookieless="AutoDetect" timeout="1" />
```

Can we explicitly destroy a Session associated with a User?

Ans: Yes, this can be performed by calling "Abandon()" method on the Session and this is what we generally do under "Sign Out" or "Log Out" options in a Web Site or Web Application.

E.g.: Session.Abandon();

Where will Web Server store Session values?

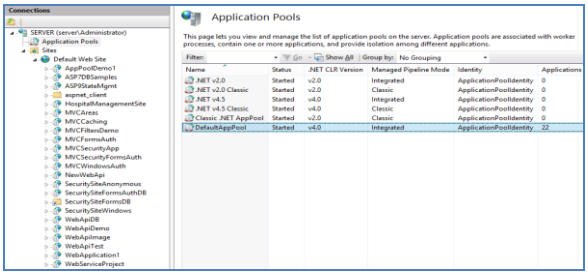
Ans: Web Server can store Session values in 3 different locations:

I. In-Proc [d]
II. State Server
III. SQL Server

In-Proc: this is the default option used for storing Session values and in this case Session values are stored under the memory of "IIS Worker Process".

What is IIS Worker Process?

Ans: Under IIS, Web Application run's inside of a container known as Application Pool and an Application Pool is a container of Web Applications that will execute under a single or multiple worker process. Application Pool is the heart of a Web Application and by default under IIS all Web Applications runs under the same Application Pool which is created when IIS is installed i.e., "Default App Pool". To check that open "IIS Manager" and in the LHS under "Connections Panel" we will find "Applications Pools" and "Sites" options, select "Application Pools" which displays "DefaultAppPool" on the RHS.



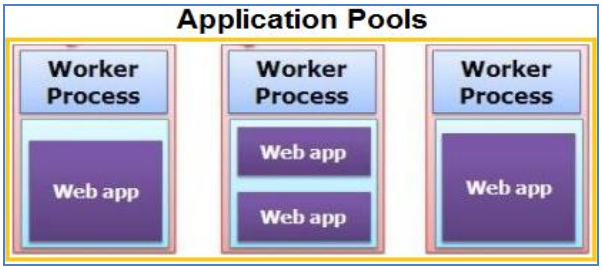
If you notice the above there are 22 applications on the server running under "DefaultAppPool" and it is still possible to run each Web Application under a separate Application Pool which enables us to isolate our Web Application for better Security, Reliability and Availability; therefore, problems in one application pool do not affect Web Sites or Applications in other Application Pools.

Note: whenever a new Site is created under IIS, it will also create an Application Pool under which the Site runs, and name of that Application Pool will be the same name of Site.

We can also create our own Application Pools under IIS and to do that right click on Application Pools node under Connection Panel and select "Add Application Pool" which opens a window asking for a name, enter name as "MyPool" and click "Ok". Currently the new Application Pool i.e., "MyPool" doesn't have any applications running under it and if we want our "ASPStateMgmt" application to run under "MyPool", then right click on

"ASPStateMgmt" under "Default Web Site" and choose Manage Applications => Advanced Settings, which opens a window and in that select "Application Pool" and Click on the button beside it which opens another window listing all the Pools that are available in a DropDownList, select "MyPool" and Click Ok and Ok again.

The IIS Worker Process is a Windows Process (w3wp.exe) which runs Web applications and is responsible for handling requests sent to a Web Server for a specific Application Pool. Each application pool creates at least one instance of w3wp.exe and that is what processes requests in your application, responsible for processing Asp.net application request and sending back response to the client. All ASP.NET functionalities run within the scope of this Worker Process.



We can view the "IIS Worker Process" that is associated with each Application Pool under "Task Manager" which displays a separate "IIS Worker Process" for each Application Pool. To get the exact details go to "Details Section" in the Task Manager and there we find "w3wp.exe" and beside that it will display the Application Pool to which the Worker Process is associated.

Note: in In-Proc session mode Session Values are stored under the Memory that is associated with IIS Worker Process, who runs our Web Application. So, in this case if we recycle the IIS Worker Process all the Session Values that are stored under this will be destroyed and to test this, open "Task Manager" identify the "IIS Worker Process" under which our Web Application is running, select it and click on "End Task" button which will destroy all the session values associated with that application.

To test the above add a new WebForm under Project, naming it as "HitCounter.aspx", place a button on it and set the text as "Hit Count" and write the following code under its "Click Event Handler":

```
int Count = 0;
if (Session["HitCounter"] == null) {
  Count = 1;
}
else {
  Count = (int)Session["HitCounter"] + 1;
}
Session["HitCounter"] = Count;
Response.Write("Hit Count: " + Count);
```

Now run the Web Page, click on the "Hit Count" button for multiple times to increment the "Count" value and at any point of time (without closing the browser) open the "Task Manager" and recycle the IIS Worker Process which is associated with our Application, now go back to the browser, click on the "Hit Count" button and notice, Hit Count value will be 1 again because the old Session value is destroyed.

State Server: this is separate software for storing "Session Values" which is installed when we install .Net Framework on any machine and it can be found in the "Services Window". To see that go to Control Panel =>

Administrative Tools => Services and in that we find "Asp.Net State Server" Service. To use this, first we need to set "mode" attribute of <sessionState> element in the "Web.config" file and "mode" attribute accepts any of the following values like: Off, In-Proc [d], SQL Server and State Server.

Note: default is In-Proc i.e., Sessions are stored under IIS Worker Process Memory, if set as off application will not maintain Sessions at all and if set as State Server then Session values are stored under ASP.NET State Service.

To use State Service for storing Session values we need to do the following:

Step 1: Open Windows Services console, right click on ASP.NET State Service and select Start to start the service.

Step 2: Now open Web.config file and write the sessionState tag as following:

```
<sessionState mode="StateServer" stateConnectionString="tcpip=localhost:42424" />
```

Note: "localhost" refers to the machine name and if ASP.NET State Server software is running on a remote machine then write that machine name in place of localhost and 42424 is the Port No. on which ASP.NET State Server software will be running.

Now run "HitCounter.aspx" page again and click on "Hit Count" button to increment the "Count" value and at any point of time, recycle the "IIS Worker Process" in "Task Manager", go back to the browser and click on the "Hit count" button and notice that "Count" value will not be reset to 1 because now Sessions are not stored under "IIS Worker Process Memory" but stored on "Asp.Net State Server" and if the "Count" value has to be reset we need to "Restart" the "State Server" service under "Services Console".

SQL Server: if "Session Mode" is set as SQL Server then Session values are stored under SQL Server DB and to use that we need to do the following:

Step 1: run the "aspnet_regsql" tool at "VS Developer Command Prompt", so that the required DB, Tables and Stored Procedure for maintaining Sessions will be created under SQL Server.

```
aspnet_regsql -?  => Help
aspnet_regsql -S <Server Name> -U <User Id> -P <Password> -E < In-case of Windows Auth> -ssadd -sstype t|p|c
```

-S: to specify Sql Server name.
-U: to specify User Id in case of Sql Authentication.
-P: to specify password in case of Sql Authentication.
-E: this must be used in case of Windows Authentication and in such case don't use -U and -P option again.
-ssadd: is to enable support for Sql Server Session State and this will create a DB on the server.
Note: If we want to remove the support for session state we need to use -ssremove in place -ssadd.
-sstype: is to specify the type of tables we want to use where "t" indicates temporary tables, "p" indicates persisted tables and "c" indicates custom tables - but in this case we need to create our own DB to store Session data, and to specify that database name we need to use "-d <Database Name>" option in the last.

Testing the processes of using "aspnet_regsql":

For SQL Server Authentication: aspnet_regsql -S Server -U Sa -P 123 -ssadd -sstype t
For Windows Authentication: aspnet_regsql -S Server -E -ssadd -sstype t

Note: in the above case we are using temporary tables for storing Session State values, so a new Database is created on the Server with the name "ASPState" and under the DB it creates a set of Stored Procedures for

managing the data and because we have asked for temporary tables, all the required tables gets created on "TempDB - System Database" and this Database will be re-created every time we re-start SQL Server, whereas if we ask for persisted tables then all the required tables also gets created on "ASPState" database only, so even if we re-start SQL Server, then also tables and their values will be persisting.

Step 2: Now open Web.config file and re-write the <sessionState> tag as following:

Sql Server Authentication:

```
<sessionState mode="SQLServer" sqlConnectionString="Data Source=Server;User Id=Sa;Password=123" />
```

Windows Authentication:

```
<sessionState mode="SQLServer" sqlConnectionString="Data Source=Server;Integrated Security=SSPI" />
```

Now run "HitCounter.aspx" page again, click on "Hit Count" button to increment "Count" value and now either recycle the "IIS Worker Process" or restart "State Server" but still the "Count" value will not be reset and if we want the value to be reset, we need to restart SQL Server under "Services Console".

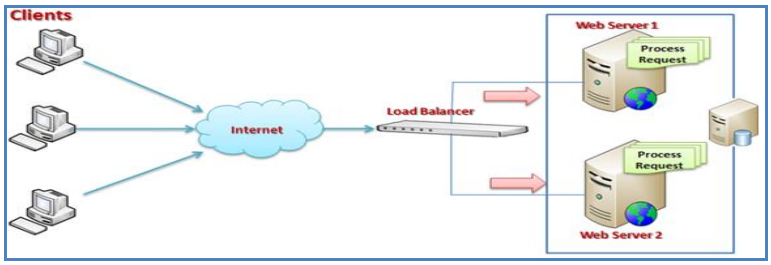
Removing support for Sql Server Session State: aspnet_regsql -S Server -U sa -P 123 -ssremove

---

When to use State Server and SQL Server Session Modes over In-Proc Session Mode?

Ans: State Server and SQL Server session mode options are used in scenarios where a Web Application is running in "Web Farm" or "Web Garden" architectures.

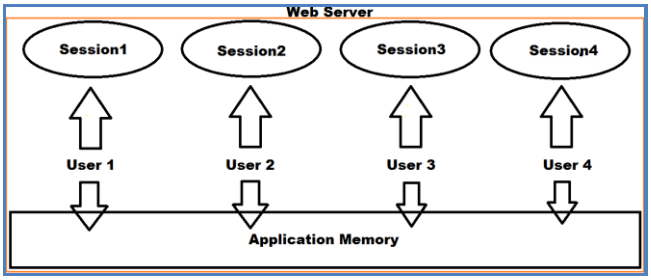
Web Farm: it's an approach of hosting a Web Application on multiple Web Servers for balancing the load, so that client request first comes to Load Balancer and Load Balancer will in-turn redirect the client to an appropriate Web Server which is free currently. In case of Web Farm architecture "In-Proc" Session Mode can't be used because if each request from the same client is re-directed to a different Web Servers, then Session Data of 1 Web Server is not accessible to other Web Servers, so to overcome this problem we install "State Server" or "SQL Server" Software on a remote system and configure all the Web Servers to that machine as following:



Web Garden: as discussed earlier every Application Pool under which our Web Application executes will be having 1 Worker Process to execute that Application, but we can create more than 1 Worker Process also in an Application Pool and if we do that, we call that as Web Garden.

To add Worker Processes to an Application Pool right click on the Application Pool in "IIS Manager", select "Advanced Settings" which opens a window and, in that window, under the "Process Model" settings we find "Maximum Worker Processes" with the value 1, change it to any value other than 1 and click Ok.

Option 6: Application State Application-State is also a state management technique, and it is a global storage mechanism that is used to store data on the server which is shared between all users i.e., data stored in Application-State is accessible to all users and anywhere in the application (Multi-User Global Data). Application-State is stored in the memory of the Web Server and is faster than storing and retrieving information from a Database. Application-State is used in the same way as Session-State, but Session-State is specific for a single user, whereas Application-State is common for all users of the application.



Application-State does not have any default expiration period like Session-State, so when we recycle the worker process only the application object will be lost. Data is stored into Application-State in the form of name/value pairs only like we store in Session-State and ViewState. We store data into Application-State by using "Application" object of our parent class "Page" and that Application object is of type "HTTPApplicationState" class.

Storing values into Application-State:

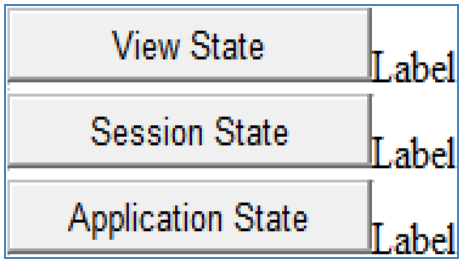
Application[string key] = value (object)

Accessing values from Application-State:

object value = Application[string key]

Note: Application Memory is not Thread-Safe, so to overcome the problem whenever we are dealing with Application-State data we need to call Lock() and UnLock() methods on Application object.

To compare the difference between "ViewState", "Session-State" and "Application-State" memory add a new WebForm under the project naming it as "CompareStates.aspx" and design it as following:




```
<h1 style="background-color:yellow;color:red;text-align:center;text-decoration:underline">Compare States</h1>
<asp:Button ID="Button1" runat="server" Text="View State" Width="200" />
<asp:Label ID="Label1" runat="server" ForeColor="Red" /><br />
```

```
<asp:Button ID="Button2" runat="server" Text="Session State" Width="200" />
<asp:Label ID="Label2" runat="server" ForeColor="Red" /><br />
<asp:Button ID="Button3" runat="server" Text="Application State" Width="200" />
<asp:Label ID="Label3" runat="server" ForeColor="Red" />
```

Now goto CompareStates.aspx.cs file and write the following code:

Code under View State Button:

```
int Count = 0;
if (ViewState["Counter"] == null) {
  Count += 1;
}
else {
  Count = (int)ViewState["Counter"] + 1;
}
ViewState["Counter"] = Count;
Label1.Text = "View State: " + Count;
```

Code under Session State Button:

```
int Count = 0;
if (Session["Counter"] == null) {
  Count += 1;
}
else {
  Count = (int)Session["Counter"] + 1;
}
Session["Counter"] = Count;
Label2.Text = "Session State: " + Count;
```

Code under Application State Button:

```
int Count = 0;
Application.Lock();
if (Application["Counter"] == null) {
  Count += 1;
}
else {
  Count = (int)Application["Counter"] + 1;
}
Application.UnLock();
Application["Counter"] = Count;
Label3.Text = "Application State: " + Count;
```

Now run the above page and click on the 3 buttons to increment their count to 5, then copy the URL of page, open a new tab, and call the page by using copied URL. When we click on "ViewState" button value will be 1 because it is "Single User Local Data". When we click on "Session State" button the value will be 6 because it is "Single User Global Data" and the "Session-Id" is accessible between tabs of the browser, then open another instance of a new browser, open the page using copied URL and in this case ViewState value will 1, Session value

also will be 1 because "Session-Id" of the old browser is not carried to new instance of new browser but application value will be "old + 1" as this is "Multi User Global Data".

Global.asax: This file is also known as the ASP.NET application file, is an optional file that contains code for responding to "Application-Level" and "Session-Level" events raised by ASP.NET. The file contains a class with the name "Global" that extends the HttpApplication class. There can be only one "Global.asax" file per application and it should be in the application's root directory only. Every ASP.Net Web Application project contains this file by default. Global class contains methods for handling Application & Session events like:

1. Application_Start
2. Application_End
3. Application_Error
4. Session_Start
5. Session_End

Application_Start: This method is invoked when the application first starts i.e., whenever the first request comes to the application and executes 1 and only 1 time in the life cycle of an application. This event handler is a useful place to provide application-wide initialization code.

Application_End: This method is invoked just before an application ends. The end of an application can occurs because IIS is being restarted or because the application is transitioning to a new application domain in response to updated files or the worker process recycling and it typically contains application cleanup logic.

Application_Error: This method is invoked whenever an unhandled exception occurs in the application, and we implement all the error handling code in this.

Session_Start: This method is invoked each time a new session begins i.e., when the first request comes from a user. This is often used to initialize user-specific information.

Session_End: This method is invoked whenever the user's session ends. A session ends when your code explicitly releases it or when its time is out after there have been no more requests received within a given timeout period (typically 20 minutes) and this contains logic for cleanup of user specific data.

Note: Global.asax file is never called directly by the user, rather they are called automatically in response to Application and Session events. When Global.asax files changes, the framework reboots the application and the Application_OnStart event is fired once again when the next request comes in. Note that the Global.asax file does not need recompilation if no changes have been made to it.

A program to find out the "Total No. of Users" and "Total No. of Online Users" for a site by using "Global.asax":

Step 1: Open Global.asax file and write the below code under the class Global.

Code under Application_Start method:

```
Application["TNU"] = 0;
Application["TOU"] = 0;
```

Code under Session_Start method:

```
Application.Lock();
Application["TNU"] = (int)Application["TNU"] + 1;
Application["TOU"] = (int)Application["TOU"] + 1;
Application.UnLock();
```

Code under Session_End method:

```
Application.Lock();
Application["TOU"] = (int)Application["TOU"] - 1;
Application.UnLock();
```

Step 2: Add a new WebForm under project, name it as UsersCount.aspx and write the below code under its div tag.

```
Total Users: <asp:Label ID="Label1" runat="server" ForeColor="Red" /><br />
Online Users: <asp:Label ID="Label2" runat="server" ForeColor="Red" /><br />
<asp:Button ID="btnSignOut" runat="server" Text="Sign-Out" />
```

Step 3: Now goto UsersCount.aspx.cs file and write the below code.

Code under Page_Load:

```
Label1.Text = Application["TNU"].ToString();
Label2.Text = Application["TOU"].ToString();
```

Under Sign-Out Button:

```
Session.Abandon();
```

Run the page for multiple times and notice both "TNU" and "TOU" will increment and if at all we click on Sign-Out button then we notice "TOU" value getting decremented.

ADO.Net

Pretty much every application deal with data in some manner, whether that data comes from memory, databases, XML files, text files, or something else. The location where we store the data can be called as a Data Source or Data Store where a Data Source can be a file, database, Address books or indexing server etc.

Programming Languages cannot communicate with Data Sources directly because each Data Source adopts a different Protocol (set of rules) for communication, so to overcome this problem long back Microsoft has introduced intermediate technologies like Odbc and Oledb which works like bridge between the Applications and Data Sources to communicate with each other.

ODBC (Open Database Connectivity) is a standard C programming language middleware API for accessing database management systems (DBMS). ODBC accomplishes DBMS independence by using an ODBC driver as a translation layer between the application and the DBMS. The application uses ODBC functions through an ODBC driver manager with which it is linked, and the driver passes the query to the DBMS. An ODBC driver will be providing a standard set of functions for the application to use, and implementing DBMS-specific functionality. An application that can use ODBC is referred to as "ODBC-Compliant". Any ODBC-Compliant application can access any DBMS for which a driver is installed. Drivers exist for all major DBMS's as well as for many other data sources like Microsoft Excel, and even for Text or CSV files.ODBC was originally developed by Microsoft in 1992.

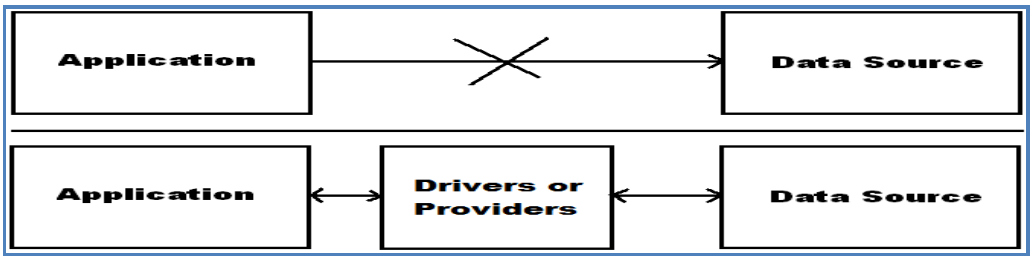
1. It's a collection of drivers, where these drivers sit between the App's and Data Source's to communicate with each other and more over we require a separate driver for each data source.
2. Odbc drivers comes along with your Windows O.S. and we can find them at the following location:
   Control Panel => Administrative Tools => Odbc Data Sources
3. To consume these Odbc Drivers first we need to configure them with the data source by creating a "DSN" (Data Source Name).
4. Odbc drivers are open source i.e. there is an availability of these Odbc Drivers for all the leading O.S's in the market.

Drawbacks with Odbc Drivers:

1. These drivers must be installed on each machine where the application is executing from and then the application, driver and data source should be manually configured with each other.
2. Odbc Drivers are initially designed for communication with Relational DB only.

OLE DB (Object Linking and Embedding, Database, sometimes written as OLEDB or OLE-DB), an API designed by Microsoft, allows accessing data from a variety of data sources in a uniform manner. The API provides a set of interfaces implemented using the Component Object Model (COM) and SQL. Microsoft originally intended OLE DB as a higher-level replacement for, and successor to, ODBC, extending its feature set to support a wider variety of non-relational databases, such as object databases and spreadsheets that do not necessarily implement SQL.OLE DB is conceptually divided into consumers and providers. The consumers are the applications that need access to the data, and the providers are the software components that implement the interface and therebyprovide the data to the consumer.An OLE DB provider is a software component enabling an OLE DB consumer to interact with a data source. OLE DB providers are alike to ODBC drivers. OLE DB providers can be created to access such simple data stores as a text file and spreadsheet, through to such complex databases as Oracle, Microsoft SQL Server, and many others. It can also provide access to hierarchical data stores. These OLE DB Providers are introduced by Microsoft around the year 1996.

1. It's a collection of providers where these providers sit between the App's and Data Source to communicate with each other, and we require a separate provider for each data source.
2. Oledb Providers are designed for communication with relational and non-relational data source also i.e. it provides support for communication with any Data Source.
3. Oledb Providers sits on server machine so they are already configured with data source and when we connect with any data source they will help in the process of communication.
4. Oledb Providers are developed by using COM and Sql Languages, so they are also un-managed.
5. Microsoft introduced OLEDB as a replacement for ODBC for its Windows Systems.
6. Oledb is a pure Microsoft technology which works only on Windows Platform.



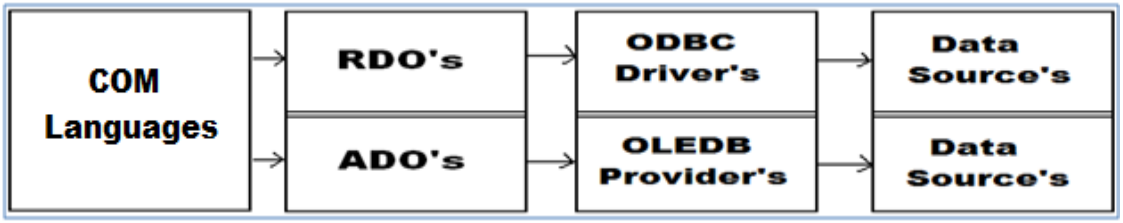
Things to remember while working with Odbc and Oledb:

1. Odbc and Oledb are un-managed or platform dependent.
2. Odbc and Oledb are not designed targeting any particular language i.e. they can be consumed by any language like: C, CPP, Visual Basic, Visual CPP, Java, CSharp etc.

Note: If any language wants to consume Odbc Drivers or OledbProviders they must use some built-in libraries of the language in which we are developing the application without writing complex coding.

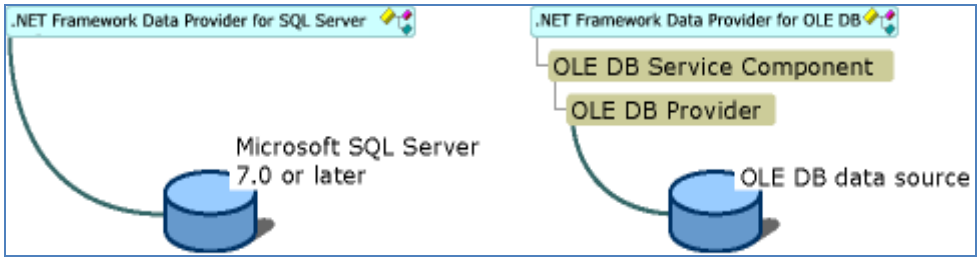
RDO's and ADO's in Visual Basic Language:

Visual Basic Language used RDO's (Remote Data Objects) and ADO's (ActiveX Data Objects) for data source communication without having to deal with the comparatively complex ODBC or OLEDB API.



.NET Framework Providers:

The .NET Framework Data Provider for SQL Server uses its own protocol to communicate with SQL Server. It is lightweight and performs well because it is optimized to access a SQL Server directly without adding an OLE DB or ODBC layer and it supports SQL Server software version 7.0 or later. The .NET Framework Data Provider for Oracle (OracleClient) enables data access to Oracle data sources through Oracle Client connectivity software. The data provider supports Oracle client software version 8.1.7 or a later.



ADO.Net:

It is a set of types that expose data access services to the .NET programmer. ADO.NET provides functionality to developers writing managed code similar to the functionality provided to native COM developers by ADO. ADO.NET provides consistent access to data sources such as Microsoft SQL Server, as well as data sources exposed through OLE DB and XML. Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, manipulate, and update data. It is an integral part of the .NET Framework, providing access to relational data, XML, and application data. ADO.NET supports a variety of development needs, including the creation of front-end database clients and middle-tier business objects used by applications or Internet browsers.

ADO.Net provides libraries for Data Source communicationunder the following namespaces:

- System.Data
- System.Data.Odbc
- System.Data.Oledb
- System.Data.SqlClient
- System.Data.OracleClient

Note: System.Data, System.Data.Odbc, System.Data.Oledb and System.Data.SqlClient namespaces are under the assembly **System.Data.dll** whereas System.Data.OracleClient is under **System.Data.OracleClient.dll** assembly.

System.Data: types of this namespace are used for holding and managing of data on client machines. This namespace contains following set of classes in it: **DataSet, DataTable, DataRow, DataColumn, DataView, DataRelation**, etc.

System.Data.Odbc: types of this namespace can communicate with any Relational DataSource using Un-Managed Odbc Drivers.

System.Data.Oledb: types of this namespace can communicate with any Data Source using Oledb Providers (Un-Managed COM Providers).

System.Data.SqlClient: types of this namespace can purely communicate with Sql Server database only using SqlClient Provider (Managed .Net Framework Provider).

System.Data.OracleClient: types of this namespace can purely communicate with Oracle database only using OracleClient Provider (Managed .Net Framework Provider).

All the above 4 namespaces contains same set of types as following: **Connection, Command, DataReader, DataAdapter, Parameter and CommandBuilder etc**, but here each class is referred by prefixing with theirnamespace before the class name to discriminate between each other as following:

OdbcConnection	OdbcCommand	OdbcDataReader	OdbcDataAdapter	OdbcCommandBuilder	OdbcParameter
OleDbConnection	OleDbCommand	OleDbDataReader	OleDbDataAdapter	OleDbCommandBuilder	OleDbParameter
SqlConnection	SqlCommand	SqlDataReader	SqlDataAdapter	SqlCommandBuilder	SqlParameter
OracleConnection	OracleCommand	OracleDataReader	OracleDataAdapter	OracleCommandBuilder	OracleParameter

Performing operations on a DataSource: the operations we perform on a Data Source will be Select, Insert, Update and Delete, and each and every operation we perform on a Data Source involves in 3 steps, like:

- Establishing a connection with the Data Source.
- Sending a request to Data Source by using SQL.
- Capturing the results sent back by the Data Source.

Establishing a Connection with Data Source:

It's a process of opening a channel for communication between Application and Data Source that is present either on a local or remote machine to perform Database operations and to open the channel for communication we use Connection class.

Constructors of the Class:

Connection()
Connection(string ConnectionString)

Note: ConnectionString is a collection of attributes that are required for connecting with a DataSource, those are:

- DSN
- Provider
- Data Source
- User Id and Password
- Integrated Security
- Database or Initial Catalog
- Extended Properties

DSN: this is the only attribute that is required if we want to connect with a data source by using Odbc Drivers and by using this we need to specify the DSN Name.

Provider: this attribute is required when we want to connect to the data source by using Oledb Providers. So by using this attribute we need to specify the provider name based on the data source we want to connect with.

SQL Server:	SqlOledb
Oracle:	Msdaora or ORAOLEDB.ORACLE
MS-Access or MS-Excel:	Microsoft.Jet.Oledb.4.0

Data Source: this attribute is required to specify the server name if the data source is a database or else if the data source is a file we need to specify path of the file and this attribute is required in case of any provider communication.

User Id and Password: this attribute is required to specify the credentials for connection with a database and this attribute is required in case of any provider communication.

Integrated Security: this attribute is used while connecting with SQL Server Database only to specify that we want to connect with the Server by using Windows Authentication and in this case we should not use User Id and Password attributes and this attribute is required in case of any provider communication.

Database or Initial Catalog: these attributes are used while connecting with Sql Server Database to specify the name of database we want to connect with and this attribute is required in case of any provider communication.

Extended Properties: this attribute is required only while connecting with MS-Excel using Oledb Provider.

List of attributes which are required in case of Odbc Drivers, Oledb and .Net Framework Providers:

Attribute	ODBC Driver	OLEDB Provider	Framework Provider
DSN	Yes	No	No
Provider	No	Yes	No
Data Source	No	Yes	Yes
User Id and Password	No	Yes	Yes
Integrated Security*	No	Yes	Yes
Database or Initial Catalog*	No	Yes	Yes
Extended Properties**	No	Yes	-

*Only for Sql Server **Only for Microsoft Excel

Connection String for SqlServer to connect by using different options:

```
OdbcConnection con = new OdbcConnection("Dsn=<Dsn Name>");
OledbConnection con = new OledbConnection("Provider=SqlOledb;Data Source=<Server Name>;
        Database=<DB Name>;User Id=<User Name>;Password=<Pwd>");
SqlConnection con = new SqlConnection("Data Source=<Server Name>;Database=<DB Name>;
        User Id=<User Name>;Password=<Pwd>");
```

Note: in case of Windows Authentication in place of User Id and Password attributes we need to use Integrated Security = SSPI (Security Support Provider Interface).

Connection String for Oracle to connect by using different options:

```
OdbcConnection con = new OdbcConnection("Dsn=<Dsn Name>");
OledbConnection con = new OledbConnection("Provider=Msdaora;Data Source=<Server Name>;
        User Id=<User Name>;Password=<Pwd>");
OracleConnection con = new OracleConnection("Data Source=<Server Name>;User Id=<User Name>;
        Password=<Pwd>");
```

Connection String for MS-Excel to connect by using different options:

```
OdbcConnection con = new OdbcConnection("Dsn=<Dsn Name>");
OledbConnection con = new OledbConnection("Provider=Microsoft.Jet.Oledb.4.0;
        Data Source=<Path of Excel Document>;Extended Properties=Excel 8.0”);
```

Methods and Properties of Connection class:

1. **Open():** a method which opens a connection with data source.
2. **Close():** a method which closes the connection that is open.
3. **State:** an enumerated readonly property which is used to get the status of connection.
4. **ConnectionString:** a property which is used to get or set a connection string that is associated with the connection object.

Object of class Connection can be created in any of the following ways:

```
Connnection con = new Connection();
con.ConnectionString = "<Connection String>";
                    or
Connection con = new Connection("<Connection String>");
```

Testing the process of establishing a connection: create a new "ASP.Net Web Application" project, naming it as "DBExamples", select "Empty Project Template", check the "Web Forms" CheckBox and click on the "Create" button. Add a WebForm in the project, name it as "TestConnection.aspx" and write the below code in its tag:

```
<asp:Button ID="Button1" runat="server" Text="Connect with Sql Server by using Odbc Driver" /> <br />
<asp:Button ID="Button2" runat="server" Text="Connect with Sql Server by using Oledb Provider" /> <br />
<asp:Button ID="Button3" runat="server" Text="Connect with Sql Server by using SqlClient Provider" /> <br />
<asp:Button ID="Button4" runat="server" Text="Connect with Microsoft Excel by using Odbc Driver" /> <br />
<asp:Button ID="Button5" runat="server" Text="Connect with Microsoft Excel by using Oledb Provider" />
```

Create 2 DSN's for connecting with "Sql Server" and "Excel" Data Sources, and to do that go to Control Panel => Administrative Tools => click on "ODBC Data Sources" which opens a new window and configure the "DSN" as following => click on "Add" button => choose a driver for "Sql Server" => click "Finish" button => Enter "Name" as: "SqlDsn", "Description" as: "Connects with Sql Server DB", "Server" as: "<Your Server Name>", click "Next" button, choose the Authentication Mode as "Windows" or "Sql Server" and if it is "Sql Server" then enter "User Id" and "Password" below, click "Next" for choosing the "Database", default will be Master; leave the same, click "Next" button and click "Finish" button which creates a DSN for "Sql Server" Database. Same as this, again click on "Add" button => choose a driver for "Microsoft Excel" => click "Finish" button => Enter "Name" as: "ExcelDsn", "Description" as: "Connects with Microsoft Excel", click on "Select Workbook" button, choose the "Excel Document" from it's physical location and click on "Ok" button which creates a new DSN for Excel.

Write the below code under "TestConnection.aspx.cs" file:

```
using System.Data.Odbc;
using System.Data.OleDb;
using System.Data.SqlClient;
```

Code under Button1 Click Event:

```
OdbcConnection con = new OdbcConnection("DSN=SqlDsn");
con.Open();
Response.Write("<script>alert('Connection is " + con.State + " with SQL Server using Odbc Driver.')</script>");
con.Close();
Response.Write("<script>alert('Connection is " + con.State + " with SQL Server using Odbc Driver.')</script>");
```

Code under Button2 Click Event:

```
OleDbConnection con = new OleDbConnection();
//con.ConnectionString = "Provider=SqlOledb;Data Source=Server;Database=Master;User Id=Sa;Password=123";
con.ConnectionString = "Provider=SqlOledb;Data Source=Server;Database=Master;Integrated Security=SSPI";
con.Open();
Response.Write("<script>alert('Connection is " + con.State + " with Sql Server using OleDb Provider.')</script>");
con.Close();
Response.Write("<script>alert('Connection is " + con.State + " with Sql Server using OleDb Provider.')</script>");
```

Code under Button3 Click Event:

```
SqlConnection con = new SqlConnection();
//con.ConnectionString = "Data Source=Server;Database=Master;User Id=Sa;Password=123";
con.ConnectionString = "Data Source=Server;Database=Master;Integrated Security=SSPI";
con.Open();
Response.Write("<script>alert('Connection is " + con.State + " with Sql Server using Framework Provider.')
</script>");
```

```
con.Close();
Response.Write("<script>alert('Connection is " + con.State + " with Sql Server using Framework Provider.')
</script>");
```

Code under Button4 Click Event:

```
OdbcConnection con = new OdbcConnection("DSN=ExcelDsn");
con.Open();
Response.Write("<script>alert('Connection is " + con.State + " with MS Excel using Odbc Driver.')</script>");
con.Close();
Response.Write("<script>alert('Connection is " + con.State + " with MS Excel using Odbc Driver.')</script>");
```

Code under Button5 Click Event:

```
OleDbConnection con = new OleDbConnection("Provider=Microsoft.Jet.Oledb.4.0; Extended Properties=Excel
8.0;Data Source=C:\\ExcelDocs\\School.xls");
con.Open();
Response.Write("<script>alert('Connection is " + con.State + " with MS Excel using OleDb Provider.')</script>");
con.Close();
Response.Write("<script>alert('Connection is " + con.State + " with MS Excel using OleDb Provider.')</script>");
```

Sending a request to Data Source by using SQL: in this process we send SQL Statement's like Select, Insert, Update and Delete to the Database for execution or Call Stored Procedures in Database for execution, and to do that we need to use the Command class.

Constructors of the class:

1. Command()
2. Command(string CommandText, Connection con)
   - CommandText means it can be any SQL Stmt like Select or Insert or Update or Delete or SP Name.
   - Connection means instance of the Connection class which we have created in our 1st step.

Properties of Command Class:

1. **Connection:** sets or gets the connection object associated with command object.
2. **CommandText:** sets or gets the Sql Statement or SP Name associated with command object.
3. **CommandType:** gets or sets whether command has to execute a Sql Statement [d] or Stored Procedure.

The object of class Command can be created in any of the following ways:

```
Command cmd = new Command();
cmd.Connection = <con>;
cmd.CommandText = "<Sql Stmt or SP Name>";
                    Or
Command cmd = new Command("<Sql Stmt or SP Name>", <con>);
```

Note: if calling a Stored Procedure then add the below statement also, in both the cases:

```
cmd.CommandType = CommandType.StoredProcedure;
```

Methods of Command class:

1. ExecuteReader() => DataReader
2. ExecuteScalar() => object
3. ExecuteNonQuery() => int

Note: after creating object of Command class we need to call any of the 3 execute methods to execute the stmt's.

- Use **ExecuteReader()** method when we want to execute a Select Statement that returns data as rows and columns. The method returns an instance of class DataReader which holds the data that is retrieved from Data Source in the form of rows and columns.
- Use **ExecuteScalar()** method when we want to execute a Select Statement that returns a single value result. The method returns result of the query in the form of an object.
- Use **ExecuteNonQuery()** method when we want to execute an SQL statement other than select, like Insert or Update or Delete etc. The method returns an integer which tells the no. of rows affected by the Stmt.

Note: The above process of calling a suitable method to capture the results is our third step i.e. capturing the results sent back by the Data Source.

Accessing data from a DataReader: DataReader is a class which can hold data in the form of rows and columns, and to access data from DataReader it provides the following members:

1. **GetName(int ColumnIndex) => string**

This method returns name of the column for given index position.

2. **Read() => bool**

This method moves the record pointer from the current location to next row & returns a Boolean value which tells whether the row to which it moved contains data or not, which will be true if data is present or false if not present.

3. **FieldCount => int**

This is a read-only property which returns the no. of columns DataReader contains or retrieved from the table.

4. **NextResult() => bool**

This method moves the record pointer from the current table to next table & returns a boolean value which tells whether the location to which it moved contains a table or not, which will be true if present or false if not present.

5. **GetValue(int ColoumnIndex) => object**

This method is used for retrieving column values from the row to which pointer was pointing by specifying the column index position. We can also access the row pointed by pointer by using an Indexer defined in the class either by specifying column index position or name, as following:

<DataReader>[int ColumnIndex] => object
<DataReader>[string ColumnName] => object

To test the examples in this document first download the file "Northwind.sql" to create Northwind DB from "Google Class Room" and run the SQL (Script) File and to do that double click on the ".sql" file which will prompt for Credentials to open under "Sql Server Management Studio". After it got opened execute the code which will create the Northwind DB on our Sql Server.

Add a WebForm in the project naming it as **"Northwind_Customers_Select.aspx"** and write the following code in **"Northwind_Customers_Select.aspx.cs"** file:

```
using System.Text;
using System.Data.SqlClient;
```

Code under Page Load Event:

```
SqlConnection con = new SqlConnection("Data Source=Server;User Id=Sa;Password=123;Database=Northwind");
SqlCommand cmd = new SqlCommand("Select CustomerID, CompanyName, ContactName, City, Country, Phone,
PostalCode, From Customers", con);
con.Open();
SqlDataReader dr = cmd.ExecuteReader();
```

```
StringBuilder sb = new StringBuilder();
sb.Append("<table border='1' align='center'>");
sb.Append("<caption>Customer Details</caption>");
sb.Append("<tr>");
for (int i = 0; i < dr.FieldCount; i++) {
  sb.Append("<th>" + dr.GetName(i) + "</th>");
}
sb.Append("</tr>");
while (dr.Read()) {
  sb.Append("<tr>");
  for (int i = 0; i < dr.FieldCount; i++) {
    sb.Append("<td>" + dr[i] + "</td>");
  }
  sb.Append("</tr>");
}
sb.Append("</table>");
con.Close();
Response.Write(sb);
```

Creating our own database on Sql Server to work with:

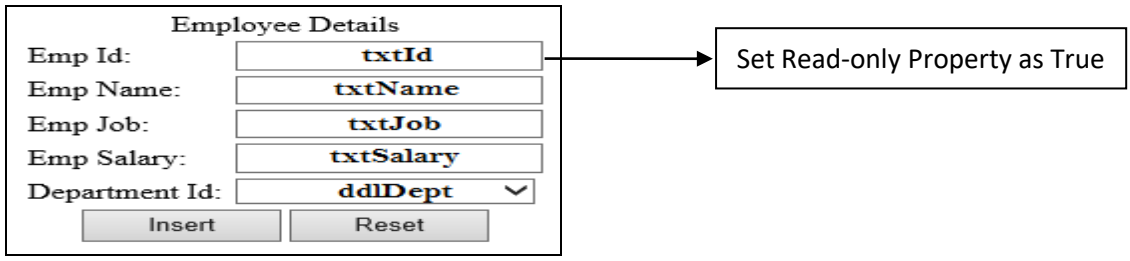
- Open Sql Server Management Studio and Create a new Database in it with the name "**ASPDB**".
- Now under the DB create 2 new tables with the names Department and Employee as following:

```
Create Table Department(Did Int Constraint Did_PK Primary Key Identity(10, 10), Dname Varchar(50), Location Varchar(50));
Create Table Employee(Eid Int Constraint Eid_PK Primary Key Identity(101, 1), Ename Varchar(50), Job Varchar(50), Salary Money, Did Int Constraint Did_FK References Department(Did));
```

- Now insert 4 records into the Department table as following:

```
Insert Into Department Values ('Marketing', 'Mumbai');
Insert Into Department Values ('Sales', 'Chennai');
Insert Into Department Values ('Accounting', 'Hyderabad');
Insert Into Department Values ('Finance', 'Delhi');
```

Add a new WebForm in the project naming it as "ASPDB_Employee_Insert.aspx" and design it as following:



Html code for creating the above form which should be implemented under tag:

```
<table align="center">
  <caption>Employee Details</caption>
  <tr>
    <td>Emp Id:</td>
```

```
    <td><asp:TextBox ID="txtId" runat="server" ReadOnly="true" /></td>
  </tr>
  <tr>
    <td>Emp Name:</td>
    <td><asp:TextBox ID="txtName" runat="server" /></td>
  </tr>
  <tr>
    <td>Emp Job:</td>
    <td><asp:TextBox ID="txtJob" runat="server" /></td>
  </tr>
  <tr>
    <td>Emp Salary:</td>
    <td><asp:TextBox ID="txtSalary" runat="server" /></td>
  </tr>
  <tr>
    <td>Department Id:</td>
    <td><asp:DropDownList ID="ddlDept" runat="server" Width="169px" /></td>
  </tr>
  <tr>
    <td colspan="2" align="center">
      <asp:Button ID="btnInsert" runat="server" Text="Insert" Width="60" />
      <asp:Button ID="btnReset" runat="server" Text="Reset" Width="60" />
    </td>
  </tr>
</table>
```

C# code which should be implemented in "ASPDB_Employee_Insert.aspx.cs" file:

```
using System.Data.SqlClient;
```

Declarations:

```
SqlCommand cmd;
SqlConnection con;
```

Code under Page Load Event:

```
con = new SqlConnection("Data Source=Server;User Id=Sa;Password=123;Database=ASPDB");
cmd = new SqlCommand();
cmd.Connection = con;
if(!IsPostBack) {
  LoadDept();
  txtName.Focus();
}
```

```
private void LoadDept() {
  cmd.CommandText = "Select Did, Dname from Department Order By Did";
  con.Open();
  SqlDataReader dr = cmd.ExecuteReader();
  ddlDept.DataSource = dr;
  ddlDept.DataTextField = "Dname";
  ddlDept.DataValueField = "Did";
```

```
  ddlDept.DataBind();
  ddlDept.Items.Insert(0, "-Select Department-");
  con.Close();
}
```

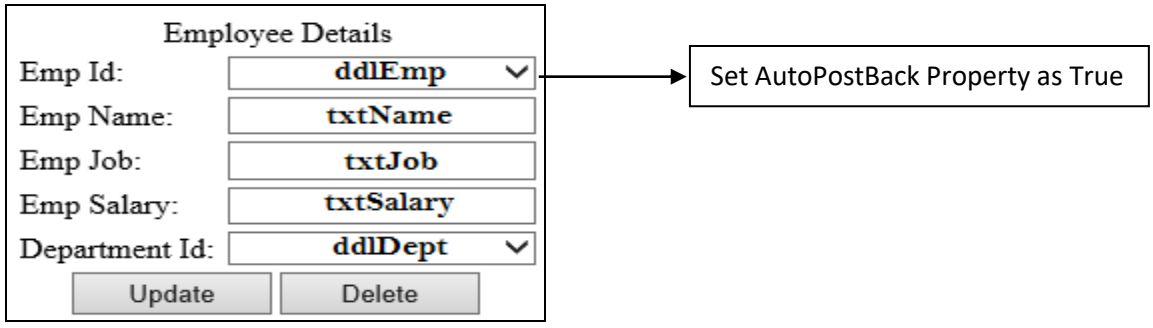
Code under Insert Button Click Event:

```
if (ddlDept.SelectedIndex > 0) {
  cmd.CommandText =
        $"Insert Into Employee Values('{txtName.Text}', '{txtJob.Text}', {txtSalary.Text}, {ddlDept.SelectedValue})";
  con.Open();
  if(cmd.ExecuteNonQuery() > 0) {
    cmd.CommandText = "Select Max(Eid) From Employee";
    txtId.Text = cmd.ExecuteScalar().ToString();
  }
  else {
    Response.Write("<script>alert('Failed inserting record into the table.')</script>");
  }
  con.Close();
}
else {
  Response.Write("<script>alert('Please choose a department to insert.')</script>");
}
```

Code under Reset Button Click Event:

```
txtId.Text = txtName.Text = txtJob.Text = txtSalary.Text = "";
ddlDept.SelectedIndex = 0;
txtName.Focus();
```

Add another WebForm in the project naming it as "ASPDB_Employee_SelectUpdateDelete.aspx" and design it as following:



Html code for creating the above form which should be implemented under tag:

```
<table align="center">
  <caption>Employee Details</caption>
  <tr>
    <td>Emp Id:</td>
    <td><asp:DropDownList ID="ddlEmp" runat="server" Width="169px" AutoPostBack="true" /></td>
  </tr>
  <tr>
    <td>Emp Name:</td>
```

```
    <td><asp:TextBox ID="txtName" runat="server" /></td>
  </tr>
  <tr>
    <td>Emp Job:</td>
    <td><asp:TextBox ID="txtJob" runat="server" /></td>
  </tr>
  <tr>
    <td>Emp Salary:</td>
    <td><asp:TextBox ID="txtSalary" runat="server" /></td>
  </tr>
  <tr>
    <td>Department Id:</td>
    <td><asp:DropDownList ID="ddlDept" runat="server" Width="169px" /></td>
  </tr>
  <tr>
    <td colspan="2" align="center">
      <asp:Button ID="btnUpdate" runat="server" Text="Update" Width="60" />
      <asp:Button ID="btnDelete" runat="server" Text="Delete" Width="60" />
    </td>
  </tr>
</table>
```

C# code which should be implemented in "ASPDB_Employee_SelectUpdateDelete.aspx.cs" file:

```
using System.Data;
using System.Data.SqlClient;
```

Declarations:

```
SqlDataReader dr;
SqlCommand cmd;
SqlConnection con;
```

Code under Page Load Event:

```
con = new SqlConnection("Data Source=Server;User Id=Sa;Password=123;Database=ASPDB");
cmd = new SqlCommand();
cmd.Connection = con;
if(!IsPostBack) {
  LoadEmp();
  LoadDept();
  ddlEmp.Focus();
}
```

```
private void LoadEmp()
{
  cmd.CommandText = "Select Eid from Employee Order By Eid";
  if (con.State != ConnectionState.Open) {
    con.Open();
  }
  dr = cmd.ExecuteReader();
  ddlEmp.DataSource = dr;
```

```
  ddlEmp.DataTextField = "Eid";
  ddlEmp.DataValueField = "Eid";
  ddlEmp.DataBind();
  ddlEmp.Items.Insert(0, "-Select Employee-");
  con.Close();
}
```

```
private void LoadDept() {
  cmd.CommandText = "Select Did, Dname from Department Order By Did";
  con.Open();
  dr = cmd.ExecuteReader();
  ddlDept.DataSource = dr;
  ddlDept.DataTextField = "Dname";
  ddlDept.DataValueField = "Did";
  ddlDept.DataBind();
  ddlDept.Items.Insert(0, "-Select Department-");
  con.Close();
}
```

Code under DropDownList SelectedIndexChanged Event:

```
if(ddlEmp.SelectedIndex > 0) {
  cmd.CommandText = "Select Ename, Job, Salary, Did From Employee Where Eid=" + ddlEmp.SelectedValue;
  con.Open();
  dr = cmd.ExecuteReader();
  if(dr.Read()) {
    txtName.Text = dr["Ename"].ToString();
    txtJob.Text = dr["Job"].ToString();
    txtSalary.Text = dr["Salary"].ToString();
    ddlDept.SelectedValue = dr["Did"].ToString();
  }
  con.Close();
}
else {
  txtName.Text = txtJob.Text = txtSalary.Text = "";
  ddlDept.SelectedIndex = 0;
  ddlEmp.Focus();
}
```

Code under Update Button Click Event:

```
if (ddlEmp.SelectedIndex > 0) {
  if (ddlDept.SelectedIndex > 0) {
    cmd.CommandText = $"Update Employee Set Ename='{txtName.Text}', Job='{txtJob.Text}',
                  Salary={txtSalary.Text}, Did={ddlDept.SelectedValue} Where Eid={ddlEmp.SelectedValue}";
    con.Open();
    if (cmd.ExecuteNonQuery() > 0) {
      Response.Write("<script>alert('Record updated in the table.')</script>");
    }
    else {
```

```
      Response.Write("<script>alert('Failed updating record in the table.')</script>");
    }
    con.Close();
  }
  else {
    Response.Write("<script>alert('Please choose a department to update.')</script>");
  }
}
else {
  Response.Write("<script>alert('Please choose a employee to update.')</script>");
}
```

Code under Delete Button Click Event:

```
if (ddlEmp.SelectedIndex > 0) {
  cmd.CommandText = "Delete From Employee Where Eid=" + ddlEmp.SelectedValue;
  con.Open();
  if (cmd.ExecuteNonQuery() > 0) {
    Response.Write("<script>alert('Record deleted from the table.')</script>");
    txtName.Text = txtJob.Text = txtSalary.Text = "";
    ddlDept.SelectedIndex = 0;
    LoadEmp();
    ddlEmp.Focus();
  }
  else {
    Response.Write("<script>alert('Failed deleting the record from table.')</script>");
    con.Close();
  }
}
else {
  Response.Write("<script>alert('Please choose a employee to delete.')</script>");
}
```

Loading multiple tables into DataReader: it is possible to load more than 1 table into a DataReader, by passing multiple select statements as CommandText to Command separated by a semicolon. We need to use the NextResult method of DataReader class to navigate the next tables after processing a table. To test this add a new WebForm in the project naming it as "**Northwind_MultipleTables.aspx**", place 2 GridView controls on it and write the following code under **"Northwind_MultipleTables.aspx.cs"** file:

```
using System.Data.SqlClient;
```

Code under Page Load Event:

```
SqlConnection con = new SqlConnection("Data Source=Server;User Id=Sa;Password=123;Database=Northwind");
SqlCommand cmd = new SqlCommand("Select CustomerId, ContactName, City, Country, Phone From Customers;
        Select EmployeeId, LastName, FirstName, BirthDate, HireDate From Employees", con);
con.Open();
SqlDataReader dr = cmd.ExecuteReader();
GridView1.DataSource = dr;
GridView1.DataBind();
```

```
dr.NextResult();
GridView2.DataSource = dr;
GridView2.DataBind();
con.Close();
```

Note: GridView is a control which is used for displaying data in a table format without manually creating a table as we have done in our first example. We need to bind the table that has to be displayed in the GridView by using its DataSource property.

DataReader: it's a class designed for holding the data on client machines in the form of Rows and Columns.

Features of DataReader:

1. Faster access to data from the Data Source.
2. Capable of holding multiple tables in it at a time.

Drawbacks of DataReader:

1. It is connection oriented so once the connection is closed between Application and Data Source all the data is lost, so State Management is not possible with a DataReader.
2. It provides forward-only access to data i.e., it allows navigating to next record or next table only.
3. It is read-only, which will not allow any changes to data that is present in it.

DataSet: it's a class which is present under "System.Data" namespace capable of holding data in the form of a Table just like a DataReader.

Differences between DataReader and DataSet:

1. DataReader's work in connection oriented architecture where as DataSet's work in disconnected architecture i.e., to hold data in DataReader we need to keep the connection open whereas in case of DataSet even after closing the connection also data will be persisting so state management is possible.
2. DataReader's provide forward only access to the data whereas DataSet's provides scrollable navigation to data which allows us to move in any direction i.e., either top to bottom or bottom to top.
3. DataReader's are non-updatable where as DataSet's are updatable i.e. changes can be made to data present in DataSet and finally those changes can be sent back to Database for saving.
4. DataReader and DataSet can hold multiple tables whereas in case of DataReader all those tables must be loaded only from a single Data Source but in case of DataSet each and each table can be loaded from Different Data Sources.

ADO.Net supports 2 different architectures for Data Source communication, like:

1. Connection Oriented Architecture
2. Disconnected Architecture

In the first case we require the connection to be kept open between the Web Server and Data Source for accessing the Data whereas in the second case we don't require the connection to be kept open continuously between the Web Server and Data Source i.e. we need the connection to be opened only for loading the Data from Data Source but not for accessing the Data.

Working with DataSet's: The class which is responsible for loading data into DataReader from a DataSource is Command and in the same way DataAdapter class is used for communication between DataSource and DataSet.

DataReader <= Command => DataSource

DataSet <=> DataAdapter <=> DataSource

Note: DataAdapter is internally a collection of 4 commands like "SelectCommand", "InsertCommand", "UpdateCommand" and "DeleteCommand" where each command is an instance of Command class, and by using these Commands DataAdapter will perform Select, Insert, Update and Delete operations on a DB Table.

Constructors of DataAdapter class:

- DataAdapter()
- DataAdapter(Command SelectCmd)
- DataAdapter(string SelectCmdText, Connection con)
- DataAdapter(string SelectCmdText, string ConnectionString)

Note: "Select Command Text" means it can be a Select Statement or a Stored Procedure which contains a Select Statement.

Instance of DataAdapter class can be created in any of the following ways:

```
Connection con = new Connection("<Connection String>");
Command cmd = new Command("<Select Stmt or SP Name>", con);
DataAdapter da = new DataAdapter();
da.SelectCommand = cmd;
```

Or

```
Connection con = new Connection("<Connection String>");
Command cmd = new Command("<Select Stmt or SP Name>", con);
DataAdapter da = new DataAdapter(cmd);
```

Or

```
Connection con = new Connection("<Connection String>");
DataAdapter da = new DataAdapter("<Select Stmt or SPName>", con);
```

or

```
DataAdapter da = new DataAdapter("<Select Stmt or SPName>", "<Connection String>");
```

Properties of DataAdapter class:

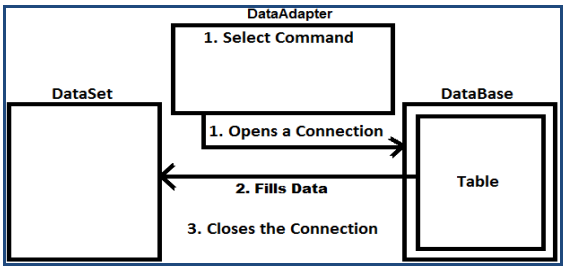
1. SelectCommand
2. InsertCommand
3. UpdateCommand
4. DeleteCommand

Methods of DataAdapter class:

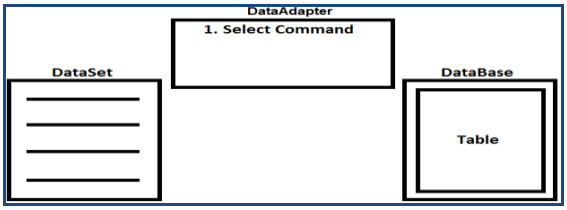
1. Fill(DataSet ds, string tableName) — Data Source => DataAdapter => DataSet
2. Update(DataSet ds, string tableName) — Data Source <= DataAdapter <= DataSet

When we call Fill method on DataAdapter following actions takes place internally:

1. Opens a connection with the Data Source.
2. Executes the SelectCommand that is present in the DataAdapter, on DataSource and loads data from Data Source to DataSet.
3. Closes the connection with Data Source.

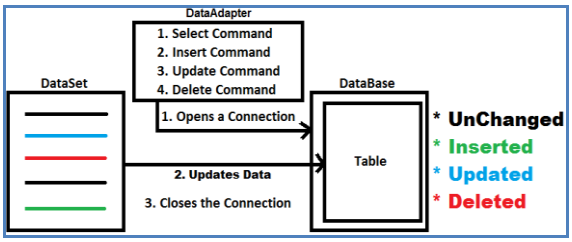


Once the execution of Fill method is completed data gets loaded into the DataSet as below:

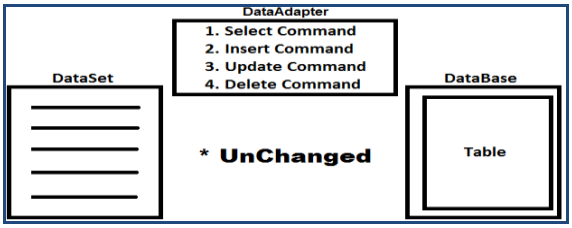


As we are discussing DataSet is updatable i.e. we can make changes to the data that is loaded into it like adding, modifying and deleting of records, and after making changes to data in DataSet if we want to send those changes back to DataSource we need to call Update method on DataAdapter, which performs the following:

1. Re-opens a connection with the Data Source.
2. Changes that are made to data present in DataSet will be sent back to corresponding Table and in this process it will make use of Insert, Update and Delete commands that are present in it.
3. Closes the connection with Data Source.



Once Update method execution is completed data gets re-loaded into DataSet as below with all unchanged rows:



Accessing data from DataSet: DataReader's provides pointer based access to the data, so we can get data only in a sequential order whereas DataSet provides index based access to the data, so we can get data from any location randomly.DataSet is a collection of tables where each table is represented as a class DataTable and identified by its index position or name.Every DataTable is again collection of Rows and collection of Columns where each row is represented as a class DataRow and identified by its index position and each column is represented as a class DataColumn and identified by its index position or name.

- Accessing a DataTable from DataSet: <dataset>.Tables[index] or <dataset>.Tables[name]

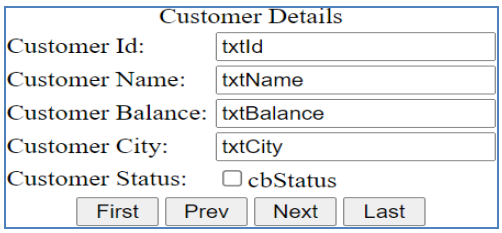
E.g.: ds.Tables[0] or ds.Tables["Customer"]

- Accessing a DataRow from DataTable: <datatable>.Rows[index]
  E.g.: ds.Tables[0].Rows[0]
- Accessing a DataColumn from DataTable: <datatable>.Columns[index] or <datatable>.Columns[name]
  E.g.: ds.Tables[0].Columns[0] or ds.Tables[0].Columns["Custid"]
- Accessing a Cell from DataTable: <datatable>.Rows[row][col]
  E.g.: ds.Tables[0].Rows[0][0] or ds.Tables[0].Rows[0]["Custid"]

To work with DataSet 1st create a table in our "ASPDB" with the name "Customer" and also insert some records into it.

```
Create Table Customer(Custid Int Constraint Custid_PK Primary Key, Name Varchar(50), Balance Money, City Varchar(50), Status Bit Default 1)
```

Now add a new WebForm in the project naming it as "ASPDB_Customer_Select.aspx" and design it as following:



Html code for creating the above form which should be implemented under tag:

```
<table align="center">
  <caption>Customer Details</caption>
  <tr>
    <td>Customer Id:</td>
    <td><asp:TextBox ID="txtId" runat="server" /></td>
  </tr>
  <tr>
    <td>Customer Name:</td>
    <td><asp:TextBox ID="txtName" runat="server" /></td>
  </tr>
  <tr>
    <td>Customer Balance:</td>
    <td><asp:TextBox ID="txtBalance" runat="server" /></td>
  </tr>
  <tr>
    <td>Customer City:</td>
    <td><asp:TextBox ID="txtCity" runat="server" /></td>
  </tr>
  <tr>
    <td>Customer Status:</td>
    <td><asp:CheckBox ID="cbStatus" runat="server" /></td>
  </tr>
  <tr>
    <td colspan="2" align="center">
```

```
      <asp:Button ID="btnFirst" runat="server" Text="First" Width="50" />
      <asp:Button ID="btnPrev" runat="server" Text="Prev" Width="50" />
      <asp:Button ID="btnNext" runat="server" Text="Next" Width="50" />
      <asp:Button ID="btnLast" runat="server" Text="Last" Width="50" />
    </td>
  </tr>
</table>
```

C# code which should be implemented in "ASPDB_Customer_Select.aspx.cs" file:

```
using System.Data;
using System.Data.SqlClient;
```

Declarations:

```
DataSet ds;
int RowIndex;
```

Code under Page Load Event:

```
if(Session["CustomerDS"] == null) {
  SqlDataAdapter da = new SqlDataAdapter("Select Custid, Name, Balance, City, Status From Customer Order By
                Custid", "Data Source=Server;User Id=Sa;Password=123;Database=ASPDB");
  ds = new DataSet();
  da.Fill(ds, "Customer");
  Session["CustomerDS"] = ds;
  Session["RowIndex"] = 0;
  ShowData();
}
else {
  ds = (DataSet)Session["CustomerDS"];
  RowIndex = (int)Session["RowIndex"];
}
```

```
private void ShowData() {
  txtId.Text = ds.Tables[0].Rows[RowIndex]["Custid"].ToString();
  txtName.Text = ds.Tables[0].Rows[RowIndex]["Name"].ToString();
  txtBalance.Text = ds.Tables[0].Rows[RowIndex]["Balance"].ToString();
  txtCity.Text = ds.Tables[0].Rows[RowIndex]["City"].ToString();
  cbStatus.Checked = (bool)ds.Tables[0].Rows[RowIndex]["Status"];
  Session["RowIndex"] = RowIndex;
}
```

Code under First Button Click Event:

```
RowIndex = 0;
ShowData();
```

Code under Previous Button Click Event:

```
if (RowIndex > 0) {
  RowIndex -= 1;
  ShowData();
}
else {
```

```
  Response.Write("<script>alert('First record of the table.')</script>");
}
```

Code under Next Button Click:

```
if (RowIndex < ds.Tables[0].Rows.Count - 1) {
  RowIndex += 1;
  ShowData();
}
else {
  Response.Write("<script>alert('Last record of the table.')</script>");
}
```

Code under Last Button Click:

```
RowIndex = ds.Tables[0].Rows.Count - 1;
ShowData();
```

Storing Connection Strings in Web.config file: in our previous examples whenever we are connecting to a Data Source we are writing "Connection String". Writing this "Connection String" each and every time will have an affect on the size of the application and moreover if any changes occur in the future to "Connection String", then we need to modify that in each and every Web Form, which will be a complex task again. To overcome the above problems, without writing "Connection String" in each and every Web Form we can put that "Connection String" in "Web.config" file, so that we don't require to write that in multiple Web Forms and also when there is a change we can modify it directly in "Web.config" file.

To store "Connection String" in the "Web.config" file we use "<connectionStrings>" tag and to test it open "Web.config" file and write the following code under "<configuration>" tag:

```
<connectionStrings>
  <add name="ASPDBCS" connectionString="User Id=Sa;Password=123;Data Source=Server;Database=ASPDB"
                                          providerName="System.Data.SqlClient" />
  <add name="NorthwindCS" connectionString="User Id=Sa;Password=123;Data Source=Server;
                                  Database=Northwind" providerName="System.Data.SqlClient" />
 </connectionStrings>
```

Reading values from Web.config file into our application: to read values of "Web.config" file in a Web Form we need to take the help of "ConfigurationManager" class which is present in "System.Configuration" namespace as following:

```
        string ASPDBConStr = ConfigurationManager.ConnectionStrings["ASPDBCS"].ConnectionString;
    string NorthwindConStr = ConfigurationManager.ConnectionStrings["NorthwindCS"].ConnectionString;
```

From now without writing "Connection String" again & again in out Web Forms, we can read it from "Web.config" file by using the above statement and consume it wherever it is required as following:

```
SqlConnection AspCon = new SqlConnection(ASPDBConStr);
SqlConnection NWCon = new SqlConnection(NorthwindConStr);
```

To still simply the process of reading connection string values into our WebForm without writing lengthy code every time, add a new class into Project naming it as ReadCS.cs and write the following code under the class:

```
using System.Configuration;
public class ReadCS {
  public static string ASPDB {
    get { return ConfigurationManager.ConnectionStrings["ASPDBCS"].ConnectionString; }
  }
  public static string Northwind {
    get { return ConfigurationManager.ConnectionStrings["NorthwindCS"].ConnectionString; }
  }
}
```

Now in our WebForms we can read the Connection String directly by calling the properties we have defined as following:

```
SqlConnection AspCon = new SqlConnection(ReadCS.ASPDB);
SqlConnection NwCon = new SqlConnection(ReadCS.Northwind);
```

Working with GridView Control: this Control is designed to display the data on a WebForm in table structure and also this Control has various features like Paging, Sorting, Customizing, Editing, etc.

Paging with GridView: this is a mechanism of displaying data on a WebForm as "Blocks of Records", generally used when we have huge volumes of data. To perform paging with a GridView Control first we need to set "AllowPaging" property as "true (default is false)" and then we need to specify the no. of records that has to be displayed within each page by setting "PageSize" Property with a numeric value "(default is 10)".

PagerSettings is a property of GridView using which we need to specify the type of paging the Control has to use with the help of "Mode" attribute and this can be set as "NextPrevious or Numeric or NextPreviousFirstLast or NumericFirstLast". Here we can also specify the Position where "Navigation Buttons" should be displayed and also the "Text or Image" values for Next, Previous, First and Last buttons.

PageIndexChanging is an event under which we need to implement the logic for navigating to next pages by setting the "PageIndex" property, which is "0" by default. To identify the Index of page where user wants to navigate in runtime, we can use "NewPageIndex" attribute of "PageIndexChanging" event.

To test paging, add a new Web Form naming it as "Northwind_Customers_GridView_Paging.aspx", place a GridView Control on it, set the "AllowPaging" property value as "true" and write the below code in "aspx.cs" file:

```
using System.Data;
using System.Data.SqlClient;
```

Declarations:

```
DataSet ds;
```

Code under Page Load Event:

```
if (Session["CustomersDS"] == null) {
  SqlDataAdapter da = new SqlDataAdapter("Select CustomerID, CompanyName, ContactName, City, Country,
                PostalCode, Phone From Customers", ReadCS.Northwind);
  ds = new DataSet(); da.Fill(ds, "Customers");
  Session["CustomerDS"] = ds;
  LoadData();
}
```

```
else {
  ds = (DataSet)Session["CustomerDS"];
}
```

```
private void LoadData() {
  GridView1.DataSource = ds;
  GridView1.DataBind();
}
```

Code under GridView PageIndexChanging Event:

```
GridView1.PageIndex = e.NewPageIndex;
LoadData();
```

Sorting with GridView: this is a mechanism of arranging data on a Web Form in ascending or descending order based on any particular column of a table. To perform "Sorting" with "GridView" Control, first we need to set "AllowSorting" property as "true (default is false)" so that "Column Names" in GridView looks like "Hyper Links" providing an option to click on them.

"Sorting" is an event under which we need to implement the logic that is required for sorting the data based on Selected Column which can be identified in our code by using "SortExpession" property of the "Sorting" event and this event fires when the Column Names (Hyper Links) are clicked by the user in runtime.

To test sorting with GridView add a new WebForm naming it as **"Northwind_Employees_GridView_Sorting.aspx"**, place a GridView Control on it, set the "AllowSorting" property as true and write the below code in **"aspx.cs"** file:

```
using System.Data;
using System.Data.SqlClient;
```

Declarations:

```
DataSet ds;
```

Code under Page Load Event:

```
if (Session["EmployeesDS"] == null) {
  SqlDataAdapter da = new SqlDataAdapter("Select EmployeeId, LastName, FirstName, Title, BirthDate, City,
        Country, PostalCode, HomePhone From Employees Order By EmployeeId Asc", ReadCS.Northwind);
  ds = new DataSet(); da.Fill(ds, "Employees");
  Session["EmployeesDS"] = ds;
  GridView1.DataSource = ds;
  GridView1.DataBind();
  Session["SortOrder"] = "EmployeeId Asc";
}
else {
  ds = (DataSet)Session["EmployeesDS"];
}
```

Code under GridView Sorting Event:

```
string[] sarr = Session["SortOrder"].ToString().Split(' ');
if(sarr[0] == e.SortExpression) {
  if (sarr[1] == "Asc") {
    Session["SortOrder"] = e.SortExpression + " Desc";
```

```
  }
  else {
    Session["SortOrder"] = e.SortExpression + " Asc";
  }
}
else {
  Session["SortOrder"] = e.SortExpression + " Asc";
DataView dv = ds.Tables["Employees"].DefaultView;
dv.Sort = Session["SortOrder"].ToString();
GridView1.DataSource = dv;
GridView1.DataBind();
```

Customizing a GridView: generally when we bind a "Data Source" to "GridView" Control it will automatically display all the columns that are being retrieved by us, because GridView control has a mechanism of "auto-generating" columns and display the data as a HTML Table., so that "Column Names" will be displayed as "Header Text" and "Column Values" will be displayed in the body. We can customize a GridView so that we can display different "Header Text" apart from "Column Names", define styles for "Header Text", "Item Text" as well as we can also use different type of controls for displaying data and even we can perform "Editing" of the data, but to do these all first we need to set "AutoGenerateColumns" property of the control as "false (default is true)". Once we set "AutoGenerateColumns" property value as "false", it is our responsibility to add each and every column individually to the GridView Control with the help of some special fields like:

- **BoundField:** Displays the value of a column in a Data Source. This is the default column type of the GridView control.
- **ButtonField:** Displays a command button for each item in the GridView control. This enables you to create a column of custom button controls, such as the Add or the Remove button.
- **CheckBoxField:** Displays a check box for each item in the GridView control. This column field type is commonly used to display fields with a Boolean value.
- **CommandField:** Displays pre-defined command buttons to perform Select, Edit, or Delete operations.
- **HyperLinkField:** Displays the value of a field in a Data Source as a hyperlink. This column field type enables you to bind a second field to the hyperlink's URL.
- **ImageField:** Displays an image for each item in the GridView control.
- **TemplateField:** Displays user-defined content for each item in the GridView control, according to a specified template. This column field type enables you to create a custom column field.

To test these add a new Web Form, naming it as **"ASPDB_Customer_GridView_Customize.aspx"** and write the following code under its tag:

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="false" Caption="Customer Details"
                                HorizontalAlign="Center" >
  <HeaderStyle BackColor="LightYellow" ForeColor="PaleVioletRed" />
  <RowStyle BackColor="Tan" ForeColor="Teal" />
  <AlternatingRowStyle BackColor="Teal" ForeColor="Tan" />
  <Columns>
    <asp:BoundField DataField="Custid" HeaderText="Cust-ID" ItemStyle-HorizontalAlign="Center" />
    <asp:BoundField DataField="Name" HeaderText="Name" />
    <asp:BoundField DataField="Balance" HeaderText="Balance" ItemStyle-HorizontalAlign="Right" />
    <asp:BoundField DataField="City" HeaderText="City" />
```

```
    <asp:CheckBoxField DataField="Status" HeaderText="Is-Active" ItemStyle-HorizontalAlign="Center" />
                                                or
    <asp:TemplateField HeaderText="Is-Active" ItemStyle-HorizontalAlign="Center">
      <ItemTemplate>
        <asp:RadioButton ID="rbStatus" runat="server" Enabled="false" Checked='<%# Eval("Status") %>' />
      </ItemTemplate>
    </asp:TemplateField>

  </Columns>
</asp:GridView>
```

Now go to "ASPDB_Customer_GridView_Customize.aspx.cs" file and write the following code:

```
using System.Data.SqlClient;
```

Code under Page Load Event:

```
SqlConnection con = new SqlConnection(ReadCS.ASPDB);
SqlCommand cmd = new SqlCommand(
                        "Select Custid, Name, Balance, City, Status From Customer Order By Custid", con);
con.Open();
SqlDataReader dr = cmd.ExecuteReader();
GridView1.DataSource = dr;
GridView1.DataBind();
con.Close();
```

Data editing with GridView: we can edit data that is present in "GridView Control" and update those changes back to the corresponding table in Database. To perform editing of the data, first "GridView Control" requires an "Edit" & "Delete" buttons, and to generate these buttons we have 3 options:

1. Set the properties AutoGenerateDeleteButton and AutoGenerateEditButton as true.
2. Add Edit & Delete options using "CommandField", by manually generating the columns as following:

```
<asp:CommandField ShowEditButton="true" ShowDeleteButton="true" />
```

3. Manual generation of those buttons by using Template Field.

To perform editing operations, we need to write logic under 4 events of the GridView those are: RowEditing, RowCancelingEdit, RowUpdating and RowDeleting.

- **RowEditing:** Occurs when a row's Edit button is clicked which changes Edit button to Update and Cancel buttons and here, we need to set the GridViews EditIndex to selected rows Index.
- **RowCancelingEdit:** Occurs when the Cancel button of a row in edit mode is clicked which changes back to Edit button and here, we need to set the GridViews EditIndex to -1.
- **RowUpdating:** Occurs when a row's Update button is clicked and here, we need to implement the logic for updating the row and finally set the GridViews EditIndex to -1.
- **RowDeleting:** Occurs when a row's Delete button is clicked and here, we need to implement the logic for deleting the row.

Now to test editing with GridView, add a new Web Form in the project naming it as **"ASPDB_Customer_GridView_Editing.aspx"** and write the following code under its tag:

```
<asp:GridView ID="GridView2" runat="server" AutoGenerateColumns="false" Caption="Customer Editing"
                                                    HorizontalAlign="Center" >
  <HeaderStyle BackColor="Yellow" ForeColor="Red" />
  <RowStyle ForeColor="Tomato" BackColor="YellowGreen" />
  <AlternatingRowStyle ForeColor="SlateGray" BackColor="Wheat" />
  <EditRowStyle BackColor="LightCoral" ForeColor="WindowFrame" />
  <Columns>
    <asp:BoundField HeaderText="Customer Id" DataField="Custid" ReadOnly="true"
                                ItemStyle-HorizontalAlign="Center" />
    <asp:BoundField HeaderText="Name" DataField="Name" />
    <asp:BoundField HeaderText="Balance" DataField="Balance" ItemStyle-HorizontalAlign="Right" />
    <asp:BoundField HeaderText="City" DataField="City" />
    <asp:CheckBoxField HeaderText="Is-Active" DataField="Status" ItemStyle-HorizontalAlign="Center"
                                                    ReadOnly="true" />
    <asp:TemplateField HeaderText="Actions" ItemStyle-BackColor="White" ItemStyle-ForeColor="Blue">
      <ItemTemplate>
        <asp:LinkButton ID="btnEdit" runat="server" Text="Edit" CommandName="Edit" />
        <asp:LinkButton ID="btnDelete" runat="server" Text="Delete" CommandName="Delete"
          OnClientClick="return confirm('Are you sure of deleting the current record?')" />
      </ItemTemplate>
      <EditItemTemplate>
        <asp:LinkButton ID="btnUpdate" runat="server" Text="Update" CommandName="Update" />
        <asp:LinkButton ID="LinkCancel" runat="server" Text="Cancel" CommandName="Cancel" />
      </EditItemTemplate>
    </asp:TemplateField>
  </Columns>
</asp:GridView>
```

Now go to Design view of the WebForm, select "GridView", open the Property Window, go to it Events, double click on "RowEditing", "RowCancelingEdit", "RowUpdating" and "RowDeleting" events to generate Event Handlers for implementing the logic and write the below code in **"ASPDB_Customer_GridView_Editing.aspx.cs"**:

```
using System.Data;
using System.Data.SqlClient;
```

Declarations:

```
SqlCommand cmd;
SqlConnection con;
```

Code under Page Load Event:

```
con = new SqlConnection(ReadCS.ASPDB);
cmd = new SqlCommand();
cmd.Connection = con;
if (!IsPostBack) {
  LoadData();
}
```

```
private void LoadData()
{
  cmd.CommandText = "Select Custid,Name,Balance,City,Status From Customer Where Status=1 Order By Custid";
  if (con.State != ConnectionState.Open) {
    con.Open();
  }
  SqlDataReader dr = cmd.ExecuteReader();
  GridView1.DataSource = dr;
  GridView1.DataBind();
  con.Close();
}
```

Code under GridView RowEditing Event:

```
GridView1.EditIndex = e.NewEditIndex;
LoadData();
```

Code under GridView RowCancelingEdit Event:

```
GridView1.EditIndex = -1;
LoadData();
```

Code under GridView RowUpdating Event:

```
try {
  int Custid = int.Parse(GridView1.Rows[e.RowIndex].Cells[0].Text);
  string Name = ((TextBox)GridView1.Rows[e.RowIndex].Cells[1].Controls[0]).Text;
  decimal Balance = decimal.Parse(((TextBox)GridView1.Rows[e.RowIndex].Cells[2].Controls[0]).Text);
  string City = ((TextBox)GridView1.Rows[e.RowIndex].Cells[3].Controls[0]).Text;

  cmd.CommandText =
                  $"Update Customer Set Name='{Name}', Balance={Balance}, City='{City}' Where Custid={Custid}";
  con.Open();
  if (cmd.ExecuteNonQuery() > 0) {
    GridView1.EditIndex = -1;
    LoadData();
  }
}
catch (Exception ex) {
  Response.Write("<script>alert('" + ex.Message.Replace("'", "") + "')</script>");
}
finally {
  con.Close();
}
```

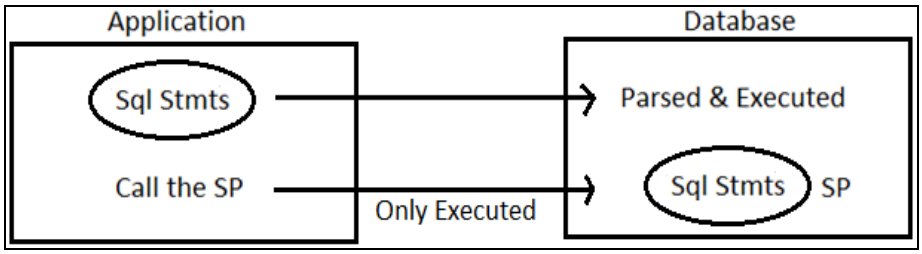
Code under GridView RowDeleting Event:

```
try {
  int Custid = int.Parse(GridView1.Rows[e.RowIndex].Cells[0].Text);
  cmd.CommandText = $"Update Customer Set Status=0 Where Custid={Custid}";
  con.Open();
  if (cmd.ExecuteNonQuery() > 0) {
    LoadData();
```

```
  }
}
catch (Exception ex) {
  Response.Write("<script>alert('" + ex.Message.Replace("'", "") + "')</script>");
}
finally {
  con.Close();
}
```

Stored Procedures

Whenever we want to interact with a database from an application we use Sql stmts. When we use Sql statements within the application we have a problem i.e. when the application runs Sql Statements will be sent to db for execution where the statements will be parsed (compiled) and then executed. The process of parsing takes place each time we run the application, because of this performance of our application decreases. To overcome the above drawback write Sql statements directly under db only, with in an object known as Stored Procedure and call them for execution. As a SP is a pre-compiled block of code that is ready for execution will directly execute the statements without parsing each time.



Syntax to define a Stored Procedure:

```
Create Procedure <Name> [ (<Parameter List>) ]
As
Begin
  <Stmts>
End;
```

- SP's are similar to method's in our language.

  public void Test() **//Method**
  Create Procedure Test() **//Stored Procedure**

- If required we can also define parameters to Stored Procedures but only optional. If we want to pass parameters to a Sql Server SP prefix the special character "@" before parameter name.

  public void Test(int x) **//CSharp**
  Create Procedure Test(@x int) **//Sql Server**

- A SP can also return values, to return a value we use Out or Output keyword in Sql Server.

  public void Test(int x, ref|out int y) **//CSharp**
  Create Procedure Test(@x int, @y int out|output) **//Sql Server**

Creating a Stored Procedure: we can create SP in SQL Server either by using SQL Server Management Studio or Visual Studio also. To create a SP using Visual Studio open the "Server Explorer" window, this will display "ASPDB" and "Northwind" Database there and those Databases are configure based on the "Connection String" value we have used in "Web.config" file. Expand our "ASPDB" database, right click on the node Stored Procedures, select "Add New Stored Procedure" which opens a window write code in it for creating a Procedure and finally right click on the document window and select "Execute" which will create the Procedure on DB Server. Now let's create the following Stored Procedures under "ASPDB" database.

```
Create Procedure Customer_Select(@Custid Int=Null, @Status Bit=Null)
As
Begin
  If @Custid Is Null And @Status Is Null                --Fetches all records from the table
    Select Custid, Name, Balance, City, Status From Customer Order By Custid;
  Else If @Custid Is Not Null And @Status Is Null       --Fetches a record from the table based on given Custid
    Select Custid, Name, Balance, City, Status From Customer Where Custid=@Custid;
  Else If @Custid Is Null And @Status Is Not Null       --Fetches records from the table based on given Status
    Select Custid, Name, Balance, City, Status From Customer Where Status=@Status Order By Custid;
  Else If @Custid Is Not Null And @Status Is Not Null  --Fetches a record from the table based on given Id & Status
    Select Custid, Name, Balance, City, Status From Customer Where Custid=@Custid And Status=@Status;
End;
```

```
Create Procedure Customer_Insert(@Name Varchar(50), @Balance Money, @City Varchar(100), @Status Bit,
                                 @Custid Int Out)
As
Begin
  Begin Transaction
    Select @Custid = IsNull(Max(Custid), 100) + 1 From Customer
    Insert Into Customer Values (@Custid, @Name, @Balance, @City, @Status);
  Commit Transaction;
End;
```

```
Create Procedure Customer_Update(@Custid Int, @Name Varchar(50), @Balance Money, @City Varchar(100))
As
  Update Customer Set Name=@Name, Balance=@Balance, City=@City Where Custid=@Custid;
```

```
Create Procedure Customer_Delete(@Custid Int)
As
  Update Customer Set Status=0 Where Custid=@Custid;
```

Calling a Stored Procedure from .NET application:

To call a SP from .NET Application's we use Command class and the process of calling will be as following:

1. Create instance of class Command by specifying SP Name as CommandText.
2. Change the CommandType property of Command as StoredProcedure because by default CommandType property is set as Text which executes Sql Stmt's and after changing that property Command can call SP's.

   cmd.CommandType = CommandType.StoredProcedure;
3. If the SP has any parameters we need to add those parameters to the Command i.e. if the parameter is input we need to add input parameters by calling "AddWithValue" method and incase if the parameter is output we need to add them by calling "Add" method of Command class.

4. If the procedure is a Query Procedure then call ExecuteReader() method of Command class which executes the procedure and loads data into DataReader, whereas if we want to load data into DataSet then create DataAdapter class instance passing Command as a parameter and call Fill Method. If the SP contains any Non-Query operations then call ExecuteNonQuery method of Command to execute the SP.

Adding Parameter values to Command:

SP can be defined with parameters either to send values for execution or receive values after execution. While calling a SP with parameters from .NET Application for each parameter of the SP we need to add a matching parameter under Command i.e. for input parameter matching input parameter has to be added and for output parameter a matching output parameter has to be added. Every parameter has 5 attributes to it like Name, Value, DbType, Size and Direction which can be Input (d) or Output.

- Name refers to name of the parameter that is defined in SP.
- Value refers to value being assigned in case of input or value we are expecting in case of output.
- DbType refers to data type of the parameter in terms of the DB where the SP exists.
- Size refers to size of data.
- Direction specifies whether parameter is Input or Output.

Based on SP's Input or Output parameters we need to add them under Command by specifying below attributes:

	Input	Output
Name	Yes	Yes
Value	Yes	No
DbType	**Yes [*]**	Yes
Size	No	**Yes [**]**
Direction	No	Yes

*Required only if the value supplied is null.
**Required only in case the output parameter type is a variable length type.

Adding input parameters under Command:

cmd.Parameters.AddWithValue(string <pname>, object <pvalue>)

Adding output parameters under Command:

cmd.Parameters.Add(string <pname>, DbType <type>[, int <size>]).Direction = ParameterDirection.Output;

Capturing output parameter values after execution of SP:

Object obj = cmd.Parameters[string <pname>].Value;

Note: ParameterDirection is an enum which contains all the directions that are supported like Input, Output, InputOutput and Returnvalue.

Now to call the above Stored Procedures in our WebForms add a new class naming it as "Customer.cs" and write methods with all the necessary logic for calling the SP's, so that we can call those methods directly in our WebForms without writing the logic each and every time when we want to call the Stored Procedure.

```
using System.Data;
using System.Data.SqlClient;
public class Customer {
```

```
SqlCommand cmd;
SqlConnection con;
public Customer() {
  con = new SqlConnection(ReadCS.ASPDB);
  cmd = new SqlCommand();
  cmd.Connection = con;
  cmd.CommandType = CommandType.StoredProcedure;
}
public DataSet Customer_Select(int? Custid, bool? Status) {
  DataSet ds;
  try {
    cmd.CommandText = "Customer_Select";
    cmd.Parameters.Clear();
    if (Custid != null && Status == null)
      cmd.Parameters.AddWithValue("@Custid", Custid);
    else if (Custid == null && Status != null)
      cmd.Parameters.AddWithValue("@Status", Status);
    else if(Custid != null && Status != null) {
      cmd.Parameters.AddWithValue("@Custid", Custid);
      cmd.Parameters.AddWithValue("@Status", Status);
    }
    SqlDataAdapter da = new SqlDataAdapter(cmd);
    ds = new DataSet();
    da.Fill(ds, "Customer");
    return ds;
  }
  catch(Exception ex) {
    throw ex;
  }
}
public int Customer_Insert(string Name, decimal? Balance, string City, bool? Status, ref int? Custid) {
  int Count = 0;
  try {
    cmd.CommandText = "Customer_Insert";
    cmd.Parameters.AddWithValue("@Name", Name);
    cmd.Parameters.AddWithValue("@Balance", Balance);
    cmd.Parameters.AddWithValue("@City", City);
    cmd.Parameters.AddWithValue("@Status", Status);
    cmd.Parameters.Add("@Custid", SqlDbType.Int).Direction = ParameterDirection.Output;
    con.Open();
    Count = cmd.ExecuteNonQuery();
    Custid = Convert.ToInt32(cmd.Parameters["@Custid"].Value);
  }
  catch (Exception ex) {
    throw ex;
  }
```

```
    finally {
      con.Close();
    }
    return Count;
  }
  public int Customer_Update(int? Custid, string Name, decimal? Balance, string City)
  {
    int Count = 0;
    try {
      cmd.CommandText = "Customer_Update";
      cmd.Parameters.Clear();
      cmd.Parameters.AddWithValue("@Custid", Custid);
      cmd.Parameters.AddWithValue("@Name", Name);
      cmd.Parameters.AddWithValue("@Balance", Balance);
      cmd.Parameters.AddWithValue("@City", City);
      con.Open();
      Count = cmd.ExecuteNonQuery();
    }
    catch (Exception ex) {
      throw ex;
    }
    finally {
      con.Close();
    }
    return Count;
  }
  public int Customer_Delete(int? Custid) {
    int Count = 0;
    try {
      cmd.CommandText = "Customer_Delete";
      cmd.Parameters.Clear();
      cmd.Parameters.AddWithValue("@Custid", Custid);
      con.Open();
      Count = cmd.ExecuteNonQuery();
    }
    catch (Exception ex) {
      throw ex;
    }
    finally {
      con.Close();
    }
    return Count;
  }
}
```

Now we can call the above methods that are defined in Customer class to perform Select, Insert, Update and Delete operations in any WebForm and to test this add a new WebForm in our project naming it as

"ASPDB_Customer_GridView_Editing_SP.aspx" and design it same as "ASPDB_Customer_GridView_Editing.aspx" and write the following code in "ASPDB_Customer_GridView_Editing_SP.aspx.cs" file:

Declarations:

```
Customer obj = new Customer();
```

Code under Page Load Event:

```
if (!IsPostBack) {
  LoadData();
}
```

```
private void LoadData() {
  GridView1.DataSource = obj.Customer_Select(null, true);
  GridView1.DataBind();
}
```

Code under Page Error Event:

```
Exception ex = Server.GetLastError();
Server.ClearError();
Response.Write("<script>alert('" + ex.Message.Replace("'", "") + "')</script>");
```

Code under GridView RowEditing Event:

```
GridView1.EditIndex = e.NewEditIndex;
LoadData();
```

Code under GridView RowCancelingEdit Event:

```
GridView1.EditIndex = -1;
LoadData();
```

Code under GridView RowUpdating Event:

```
int Custid = int.Parse(GridView1.Rows[e.RowIndex].Cells[0].Text);
string Name = ((TextBox)GridView1.Rows[e.RowIndex].Cells[1].Controls[0]).Text;
decimal Balance = decimal.Parse(((TextBox)GridView1.Rows[e.RowIndex].Cells[2].Controls[0]).Text);
string City = ((TextBox)GridView1.Rows[e.RowIndex].Cells[3].Controls[0]).Text;
if(obj.Customer_Update(Custid, Name, Balance, City) > 0) {
  GridView1.EditIndex = -1;
  LoadData();
}
else {
  Response.Write("<script>alert('Failed updating the record in table.')</script>");
}
```

Code under GridView RowDeleting Event:

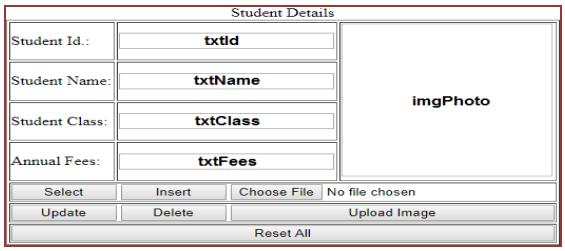
```
int Custid = int.Parse(GridView1.Rows[e.RowIndex].Cells[0].Text);
if (obj.Customer_Delete(Custid) > 0) {
  LoadData();
}
else {
  Response.Write("<script>alert('Failed deleting the record from table.')</script>");
}
```

Storing Images in Database: we can store images, audio and video data into database and while saving them we can either save path of those files in a column or content of those files in database by converting them into byte[] or binary format. To store byte[] or binary data, column should use Image or Varbinary data type.

Create a table under our ASPDB database representing a Student entity as following:

```
Create Table Student (Sid Int Constraint Sid_Pk Primary Key, Name Varchar(50), Class Int, Fees Money, PhotoName Varchar(50), PhotoBinary VarBinary(Max), Status Bit Not Null Default 1);
```

Add a WebForm in the project naming it as "ASPDB_Student_DataMgmt.aspx" and design it as following:



Html code for creating the above form which should be implemented under tag:

```
<table align="center" style="background-color: bisque">
  <caption>Student Data Management</caption>
  <tr>
    <td>Student Id:</td>
    <td><asp:TextBox ID="txtId" runat="server" /></td>
    <td rowspan="4">
      <asp:Image ID="imgPhoto" runat="server" Height="200px" Width="200px" BorderStyle="Groove" />
    </td>
  </tr>
  <tr>
    <td>Student Name:</td>
    <td><asp:TextBox ID="txtName" runat="server" /></td>
  </tr>
  <tr>
    <td>Student Class:</td>
    <td><asp:TextBox ID="txtClass" runat="server" /></td>
  </tr>
  <tr>
    <td>Annual Fees:</td>
    <td><asp:TextBox ID="txtFees" runat="server" /></td>
  </tr>
  <tr>
    <td colspan="3">
      <asp:Button ID="btnSelect" runat="server" Text="Select" Width="100px" />
      <asp:Button ID="btnInsert" runat="server" Text="Insert" Width="100px" />
      <asp:FileUpload ID="FileUpload1" runat="server" />
    </td>
  </tr>
```

```
  <tr>
    <td colspan="3">
      <asp:Button ID="btnUpdate" runat="server" Text="Update" Width="100px" />
      <asp:Button ID="btnDelete" runat="server" Text="Delete" Width="100px" />
      <asp:Button ID="btnUpload" runat="server" Text="Upload Image" Width="270px" />
    </td>
  </tr>
  <tr>
    <td colspan="3"><asp:Button ID="btnReset" runat="server" Text="Reset All" Width="478px" /></td>
  </tr>
</table>
<asp:Label ID="lblMsgs" runat="server" ForeColor="Red" />
```

C# code which should be implemented in "ASPDB_Student_DataMgmt.aspx.cs" file:

```
using System.IO;
using System.Data;
using System.Data.SqlClient;
```

Declarations:

```
SqlCommand cmd;
SqlConnection con;
```

Code under Page Load Event:

```
con = new SqlConnection(ReadCS.ASPDB);
cmd = new SqlCommand();
cmd.Connection = con;
```

Code under Upload Image Button Click Event:

```
if(FileUpload1.HasFiles) {
  HttpPostedFile selectedFile = FileUpload1.PostedFile;
  string fileExtension = Path.GetExtension(selectedFile.FileName);
  if(fileExtension == ".jpg" || fileExtension == ".bmp" || fileExtension == ".png") {
    string imgName = selectedFile.FileName;
    string folderPath = Server.MapPath("~/Images/");
    if(!Directory.Exists(folderPath)) {
      Directory.CreateDirectory(folderPath);
    }
    selectedFile.SaveAs(folderPath + imgName);
    imgPhoto.ImageUrl = "~/Images/" + imgName;
    BinaryReader br = new BinaryReader(selectedFile.InputStream);
    byte[] imgData = br.ReadBytes(selectedFile.ContentLength); //Converting the image into byte[] (Binary Format)

    //Storing image name & image binary values in session to access them in Insert & Update
    Session["PhotoName"] = imgName;
    Session["PhotoBinary"] = imgData;
  }
  else {
    Response.Write("<script>alert('Supported image file formats are .jpg, .bmp and .png only.')</script>");
  }
```

```
}
else {
  Response.Write("<script>alert('Please select an image file to upload.')</script>");
}
```

```
private void ClearData() {
  txtId.Text = txtName.Text = txtClass.Text = txtFees.Text = lblMsgs.Text = imgPhoto.ImageUrl = "";
  Session["PhotoName"] = Session["PhotoBinary"] = null;
  txtId.Focus();
}
```

Code under Reset All Button Click Event:

```
ClearData();
```

Code under Select Button Click Event:

```
try {
  cmd.CommandText =
      "Select Sid, Name, Class, Fees, PhotoName, PhotoBinary From Student Where Status=1 and Sid=" + txtId.Text;
  con.Open();
  SqlDataReader dr = cmd.ExecuteReader();
  if (dr.Read()) {
    txtName.Text = dr["Name"].ToString();
    txtClass.Text = dr["Class"].ToString();
    txtFees.Text = dr["Fees"].ToString();
    if (dr["PhotoName"] != DBNull.Value) {
      imgPhoto.ImageUrl = "~/Images/" + dr["PhotoName"];
      Session["PhotoName"] = dr["PhotoName"].ToString();
    }
    else {
      imgPhoto.ImageUrl = "";
      Session["PhotoName"] = null;
    }
    if (dr["PhotoBinary"] != DBNull.Value) {
      Session["PhotoBinary"] = (byte[])dr["PhotoBinary"];
    }
    else {
      Session["PhotoBinary"] = null;
    }
  }
  else {
    Response.Write("<script>alert('No student exists with given ID.')</script>");
    ClearData();
  }
}
catch (Exception ex) {
  lblMsgs.Text = ex.Message;
}
finally {
```

```
  con.Close();
}
```

```
private void AddParameters() {
  cmd.Parameters.AddWithValue("@Sid", txtId.Text);
  cmd.Parameters.AddWithValue("@Name", txtName.Text);
  cmd.Parameters.AddWithValue("@Class", txtClass.Text);
  cmd.Parameters.AddWithValue("@Fees", txtFees.Text);
  if (Session["PhotoName"] != null) {
    cmd.Parameters.AddWithValue("@PhotoName", Session["PhotoName"].ToString());
  }
  else {
    cmd.Parameters.AddWithValue("@PhotoName", DBNull.Value);
    cmd.Parameters["@PhotoName"].SqlDbType = SqlDbType.VarChar;
  }
  if (Session["PhotoBinary"] != null) {
    cmd.Parameters.AddWithValue("@PhotoBinary", (byte[])Session["PhotoBinary"]);
  }
  else {
    cmd.Parameters.AddWithValue("@PhotoBinary", DBNull.Value);
    cmd.Parameters["@PhotoBinary"].SqlDbType = SqlDbType.VarBinary;
  }
}
```

Code under Insert Button Click Event:

```
try {
  cmd.CommandText = "Insert Into Student (Sid, Name, Class, Fees, PhotoName, PhotoBinary) Values(@Sid,
                    @Name, @Class, @Fees, @PhotoName, @PhotoBinary)";
  AddParameters();
  con.Open();
  cmd.ExecuteNonQuery();
  Response.Write("<script>alert('Record inserted into the table.')</script>");
}
catch(Exception ex) {
  lblMsgs.Text = ex.Message;
}
finally {
  con.Close();
}
```

Code under Update Button Click Event:

```
try {
  cmd.CommandText = "Update Student Set Name=@Name, Class=@Class, Fees=@Fees,
                                    PhotoName=@PhotoName, PhotoBinary=@PhotoBinary Where Sid=@Sid";
  AddParameters();
  con.Open();
  cmd.ExecuteNonQuery();
  Response.Write("<script>alert('Record updated in the table.')</script>");
```

```
}
catch (Exception ex) {
  lblMsgs.Text = ex.Message;
}
finally {
  con.Close();
}
```

Code under Delete Button Click Event:

```
try {
  cmd.CommandText = "Update Student Set Status=0 Where Sid=@Sid";
  cmd.Parameters.AddWithValue("@Sid", txtId.Text);
  con.Open();
  cmd.ExecuteNonQuery();
  ClearData();
  Response.Write("<script>alert('Record deleted from the table.')</script>");
}
catch (Exception ex) {
  lblMsgs.Text = ex.Message;
}
finally {
  con.Close();
}
```

Repeater Control: this control is a container control that allows you to create custom lists out of any data that is available to the page which can be loaded from DB, XML File or any other source of data. The Repeater control does not have a built-in rendering of its own i.e. doesn't have any generation of columns like a GridView, which means that you must provide the layout for the Repeater control by using templates. When the page runs the Repeater control loops through the records in the data source and renders an item for each record. To use the Repeater control, we must create templates that define the layout of the control's content. Templates can contain any combination of markup and controls. If no templates are defined, or if none of the templates contain elements, the control does not appear on the page when the application is run. Repeater control supports the following templates:

1. **HeaderTemplate:** contains the text and controls to render at the beginning of the list.
2. **ItemTemplate:** contains the HTML elements and controls to render once for each data item loaded from data source.
3. **AlternatingItemTemplate:** contains the HTML elements and controls to render once for every other data item loaded from data source. Typically, this template is used to create a different look for the alternating items, such as a different background color than the color that is specified in the ItemTemplate.
4. **SeparatorTemplate:** contains the elements to render between each item. A typical example might be a line or break (using an <hr /> or
 elements).
5. **FooterTemplate:** contains the text and controls to render at the end of the list.

To work with Repeater control, add a new Web Form in the project naming it as "ASPDB_Customer_Repeater_Demo1.aspx" and write the following code under its " " tag:

```
<asp:Repeater ID="Repeater1" runat="server">
```

```
  <HeaderTemplate>
    <table border="1" align="center">
      <caption style="color:red;font-size:xx-large">Customer Details</caption>
      <tr style="background-color:palevioletred;color:greenyellow">
        <th>Custid</th><th>Name</th><th>Balance</th><th>City</th><th>Status</th>
      </tr>
  </HeaderTemplate>
  <ItemTemplate>
    <tr style="background-color:tan;color:teal">
      <td align="center"><%# Eval("Custid") %></td>
      <td><%# Eval("Name") %></td>
      <td align="right"><%# Eval("Balance") %></td>
      <td><%# Eval("City") %></td>
      <td align="center">
        <asp:CheckBox ID="cbStatus" runat="server" Enabled="false" Checked='<%# Eval("Status") %>' />
      </td>
    </tr>
  </ItemTemplate>
  <AlternatingItemTemplate>
    <tr style="background-color:teal;color:tan">
      <td align="center"><%# Eval("Custid") %></td>
      <td><%# Eval("Name") %></td>
      <td align="right"><%# Eval("Balance") %></td>
      <td><%# Eval("City") %></td>
      <td align="center">
        <asp:CheckBox ID="cbStatus" runat="server" Enabled="false" Checked='<%# Eval("Status") %>' />
      </td>
    </tr>
  </AlternatingItemTemplate>
  <FooterTemplate>
    </table>
  </FooterTemplate>
</asp:Repeater>
```

Now go to "ASPDB_Customer_Repeater_Demo1.aspx.cs" file and write the following code in Page Load Event:

```
Customer obj = new Customer();
Repeater1.DataSource = obj.Customer_Select(null, null);
Repeater1.DataBind();
```

Now add another WebForm naming it as "ASPDB_Customer_Repeater_Demo2.aspx" and write the following code under tag which will now display the data in a List format:

```
<asp:Repeater ID="Repeater1" runat="server">
  <HeaderTemplate>
    <h1 style="background-color: lightgoldenrodyellow; color: red; text-decoration: underline; text-align: center">
      Customer Details
    </h1>
```

```
  </HeaderTemplate>
  <ItemTemplate>
    Customer <%# Eval("Custid") %>
    <ul style="background-color: aliceblue">
      <li>Name: <%# Eval("Name") %></li>
      <li>Balance: <%# Eval("Balance") %></li>
      <li>City: <%# Eval("City") %></li>
      <li>Is-Active: <%# Eval("Status") %></li>
    </ul>
  </ItemTemplate>
  <AlternatingItemTemplate>
    Customer <%# Eval("Custid") %>
    <ul style="background-color: bisque">
      <li>Name: <%# Eval("Name") %></li>
      <li>Balance: <%# Eval("Balance") %></li>
      <li>City: <%# Eval("City") %></li>
      <li>Is-Active: <%# Eval("Status") %></li>
    </ul>
  </AlternatingItemTemplate>
  <SeparatorTemplate>
    <hr style="border: 1px solid red" />
  </SeparatorTemplate>
  <FooterTemplate>
    <h3 style="background-color: lightgoldenrodyellow; color: red; text-align: center">End of Customer Data</h3>
  </FooterTemplate>
</asp:Repeater>
```

Now go to "ASPDB_Customer_Repeater_Demo2.aspx.cs" file and write the following code in Page Load Event:

```
Customer obj = new Customer();
Repeater1.DataSource = obj.Customer_Select(null, null);
Repeater1.DataBind();
```

DataList Control: this control displays data in a format that you can define using templates and styles like a Repeater control. The DataList control is useful for displaying data in any repeating structure, such as a table. The DataList control can display rows in different layouts, such as ordering them in columns or rows or rows and columns. Optionally, you can configure the DataList control to allow users to edit or delete information. You can also customize the control to support other functionality, such as selecting rows, editing rows and Deleting rows. DataList control support the following Templates and Styles for displaying data as following:

Templates	Styles
HeaderTemplate	HeaderStyle
FooterTemplate	FooterStyle
ItemTemplate	ItemStyle
AlternatingItemTemplate	AlternatingItemStyle
SelectedItemTemplate	SelectedItemStyle
EditItemTemplate	EditItemStyle

SeparatorTemplate	SeparatorStyle

SelectedItemTemplate: Used to render the selected item; the selected item is the item whose index corresponds to the DataList's SelectedIndex property.

EditItemTemplate: Used to render the item being edited and provide editable controls for editing the row.

Add a new WebForm in the project naming it as "ASPDB_Student_DataList_Demo.aspx" and write the following code under its " " tag:

```
<asp:DataList ID="DataList1" runat="server" RepeatColumns="3" GridLines="Both" RepeatDirection="Horizontal"
                                        HorizontalAlign="Center" Width="100%">
  <HeaderStyle BackColor="LightGoldenrodYellow" ForeColor="MediumVioletRed" HorizontalAlign="Center"
                                                Font-Size="XX-Large" />
  <HeaderTemplate>Student Details</HeaderTemplate>
  <ItemStyle BackColor="Thistle" />
  <AlternatingItemStyle BackColor="SteelBlue" />
  <ItemTemplate>
    <table border="1" width="100%">
      <tr>
        <td rowspan="5"><asp:Image ID="imgPhoto" runat="server" Width="220" Height="200"
                                ImageUrl='<%# "~/Images/" + Eval("PhotoName") %>' /></td>
        <td>SID: <%# Eval("Sid") %></td>
      </tr>
      <tr><td>Name: <%# Eval("Name") %></td></tr>
      <tr><td>Class: <%# Eval("Class") %></td></tr>
      <tr><td>Fees: <%# Eval("Fees") %></td></tr>
      <tr><td>Is-Active: <%# Eval("Status") %></td></tr>
    </table>
  </ItemTemplate>
</asp:DataList>
```

Note: In the above case we are displaying photo in Image Control with the help of its path that's stored in DB where as if we want to display image based on the byte[] stored in DB table we need to write code as following:

```
<asp:Image ID="imgPhoto" runat="server" Width="220" Height="200"
        ImageUrl='<%# "" + Convert.ToBase64String((byte[])Eval("PhotoBinary")) %>' />
```

Now write the following code in "ASPDB_Student_DataList_Demo.aspx.cs" file:

```
using System.Data.SqlClient;
```

Code under Page Load Event:

```
SqlConnection con = new SqlConnection(ReadCS.ASPDB);
SqlCommand cmd = new SqlCommand(
        "Select Sid, Name, Class, Fees, PhotoName, PhotoBinary, Status From Student Order By Sid", con);
con.Open();
SqlDataReader dr = cmd.ExecuteReader();
```

```
DataList1.DataSource = dr;
DataList1.DataBind();
con.Close();
```

Performing Editing operations by using DataList Control: add a new WebForm in the project naming it as "ASPDB_Customer_DataList_Editing.aspx" and write the following code under its " " tag:

```
<asp:DataList ID="DataList1" runat="server" DataKeyField="Custid" RepeatColumns="4" GridLines="Both"
                                        RepeatDirection="Horizontal" Width="100%">
  <HeaderStyle BackColor="LightGoldenrodYellow" ForeColor="MediumVioletRed" Font-Size="XX-Large"
                                                HorizontalAlign="Center" />
  <HeaderTemplate>Customer Details</HeaderTemplate>
  <ItemStyle BackColor="Thistle" />
  <AlternatingItemStyle BackColor="SteelBlue" />
  <ItemTemplate>
    <table>
      <tr><td>Custid:</td><td><%# Eval("Custid") %></td></tr>
      <tr><td>Name:</td><td><%# Eval("Name") %></td></tr>
      <tr><td>Balance:</td><td><%# Eval("Balance") %></td></tr>
      <tr><td>City:</td><td><%# Eval("City") %></td></tr>
      <tr>
        <td>Status:</td>
        <td><asp:CheckBox ID="cbStatus" runat="server" Enabled="false" Checked='<%# Eval("Status") %>' /></td>
      </tr>
      <tr>
        <td colspan="2" align="center">
          <asp:LinkButton ID="btnEdit" runat="server" Text="Edit" CommandName="Edit" />
          <asp:LinkButton ID="btnDelete" runat="server" Text="Delete" CommandName="Delete"
                  OnClientClick="return confirm('Are you sure of deleting the record?')" />
        </td>
      </tr>
    </table>
  </ItemTemplate>
  <EditItemTemplate>
    <table>
      <tr><td>Custid:</td><td><%# Eval("Custid") %></td></tr>
      <tr>
        <td>Name:</td>
        <td><asp:TextBox ID="txtName" runat="server" Text='<%# Eval("Name") %>' /></td>
      </tr>
      <tr>
        <td>Balance:</td>
        <td><asp:TextBox ID="txtBalance" runat="server" Text='<%# Eval("Balance") %>' /></td>
      </tr>
      <tr>
        <td>City:</td>
        <td><asp:TextBox ID="txtCity" runat="server" Text='<%# Eval("City") %>' /></td>
```

```
      </tr>
      <tr>
        <td>Status:</td>
        <td><asp:CheckBox ID="cbStatus" runat="server" Enabled="false" Checked='<%# Eval("Status") %>' /></td>
      </tr>
      <tr>
        <td colspan="2" align="center">
          <asp:LinkButton ID="btnUpdate" runat="server" Text="Update" CommandName="Update" />
          <asp:LinkButton ID="btnCancel" runat="server" Text="Cancel" CommandName="Cancel" />
        </td>
      </tr>
    </table>
  </EditItemTemplate>
</asp:DataList>
```

Note: DataKeyField property is used for specifying the Key Column of table which is used in update and delete.

To perform editing with DataList control we need to write code under the following events:

1. **EditCommand:** Occurs when the Edit button is clicked for an item in the DataList control.
2. **CancelCommand:** Occurs when the Cancel button is clicked for an item in the DataList control.
3. **UpdateCommand:** Occurs when the Update button is clicked for an item in the DataList control.
4. **DeleteCommand:** Occurs when the Delete button is clicked for an item in the DataList control.

Now go to "ASPDB_Customer_DataList_Editing.aspx.cs" file and write the following code:

Declarations:

```
Customer obj = new Customer();
```

Code under Page Load Event:

```
if (!IsPostBack) {
  LoadData();
}
```

```
private void LoadData() {
  DataList1.DataSource = obj.Customer_Select(null, true);
  DataList1.DataBind();
}
```

Code under DataList EditCommand Event:

```
DataList1.EditItemIndex = e.Item.ItemIndex;
LoadData();
```

Code under DataList CancelCommand Event:

```
DataList1.EditItemIndex = -1;
LoadData();
```

Code under DataList UpdateCommand Event:

```
int Custid = (int)DataList1.DataKeys[e.Item.ItemIndex];
string Name = ((TextBox)e.Item.FindControl("txtName")).Text;
decimal Balance = decimal.Parse(((TextBox)e.Item.FindControl("txtBalance")).Text);
string City = ((TextBox)e.Item.FindControl("txtCity")).Text;
```

```
if(obj.Customer_Update(Custid, Name, Balance, City) > 0) {
  DataList1.EditItemIndex = -1;
  LoadData();
}
else {
  Response.Write("<script>alert('Failed updating record in the table.')</script>");
}
```

Code under DataList UpdateCommand Event:

```
int Custid = (int)DataList1.DataKeys[e.Item.ItemIndex];
if (obj.Customer_Delete(Custid) > 0) {
  LoadData();
}
else {
  Response.Write("<script>alert('Failed deleting record from the table.')</script>");
}
```

DetailsView Control: this control displays a single record from a data source, where each data row represents a field in the record. It is often used in combination with a GridView control for master/detail scenarios. This control gives you the ability to display, edit, insert, or delete a single record at a time from its associated data source. This control does not support sorting.

By default, the controls AutoGenerateRows property is set as true so when we bind a DataTable to the control, it will by default display first record of the table. The control provides user interface for navigating between data records and to enable paging behavior, set the AllowPaging property to true and then write code under "PageIndexChanging" event for navigating to next records.

To test working with DetailView control add a new WebForm in the project naming it as "ASPDB_Student_DetailsView_Demo.aspx", place a "DetailsView" control on it and write the following code under "ASPDB_Student_DetailsView_Demo.aspx.cs" file:

```
using System.Data;
using System.Data.SqlClient;
```

Code under Page Load Event:

```
if (!IsPostBack) {
  LoadData();
}
```

```
private void LoadData() {
  SqlDataAdapter da = new SqlDataAdapter(
                "Select Sid, Name, Class, Fees, PhotoName, Status From Student Order By Sid", ReadCS.ASPDB);
  DataSet ds = new DataSet();
  da.Fill(ds, "Student");
  DetailsView1.DataSource = ds;
  DetailsView1.DataBind();
}
```

Now when we run the Web Form by default it displays the first record of the table only without any paging option, so to enable paging go to design view of the Web Form and set AllowPaging property of DetailsView

control as true which displays pager below the control. We can use PagerSettings property to specify the type of paging we want. After enabling paging, go to events of the DetailsView control double click on PageIndexChanging event and write the below code under the generated Event Handler:

```
DetailsView1.PageIndex = e.NewPageIndex;
LoadData();
```

Right now, in the output we will not see Photograph of the student but we see the image name because in our table it is a Varchar column and treated same like other columns (Sid, Name, etc.). So if we want the image to be displayed over there we need to explicitly load the image by placing Image Control and to do that we need to generate rows manually without using auto generation of rows. To test the process set the AutoGenerateRows property of the DetailsView control as false and write code for generating rows manually, so that "DetailsView" control creating code should now look as following inside of the " " tag:

```
<asp:DetailsView ID="DetailsView1" runat="server" AllowPaging="true" PagerStyle-HorizontalAlign="Center"
        HorizontalAlign="Center" Caption="Student Details" AutoGenerateRows="false">
  <Fields>
    <asp:BoundField DataField="Sid" HeaderText="Student ID:" />
    <asp:BoundField DataField="Name" HeaderText="Student Name:" />
    <asp:BoundField DataField="Class" HeaderText="Student Class:" />
    <asp:BoundField DataField="Fees" HeaderText="Student Fees:" />
    <asp:TemplateField HeaderText="Student Image:">
      <ItemTemplate>
        <asp:Image ID="imgPhoto" runat="server" Width="200" Height="200"
                                                ImageUrl='<%# "~/Images/" + Eval("PhotoName") %>' />
      </ItemTemplate>
    </asp:TemplateField>
    <asp:CheckBoxField DataField="Status" HeaderText="Is-Active:" />
  </Fields>
</asp:DetailsView>
```

Performing editing operations using DetailsView control: to perform editing operations by using DetailsView control first we need to generate Insert, Update and Delete buttons for the control either by setting the properties "ShowInsertButton", "ShowUpdateButton" and "ShowDeleteButton" as true in property window or we can do that manually by using "CommandFields" or "TemplateFields" also. To test these add a new Web Form in the project naming it as "ASPDB_Customer_DetailsView_Editing.aspx" and write the following code under its " " tag:

```
<asp:DetailsView ID="DetailsView1" runat="server" AllowPaging="true" AutoGenerateRows="false"
        Caption="Customer Details"" HorizontalAlign="Center" DataKeyNames="Custid">
  <Fields>
    <asp:TemplateField HeaderText="Custid">
      <ItemTemplate>
        <asp:Label ID="lblCustid" runat="server" Text='<%# Eval("Custid") %>' />
      </ItemTemplate>
    </asp:TemplateField>
    <asp:BoundField HeaderText="Name" DataField="Name" />
    <asp:BoundField HeaderText="Balance" DataField="Balance" />
    <asp:BoundField HeaderText="City" DataField="City" />
    <asp:CheckBoxField HeaderText="Status" DataField="Status" ReadOnly="true" />
```

```
    <asp:TemplateField HeaderText="Actions: ">
      <ItemTemplate>
        <asp:LinkButton ID="btnNew" runat="server" Text="New" CommandName="New" />
        <asp:LinkButton ID="btnEdit" runat="server" Text="Edit" CommandName="Edit" />
        <asp:LinkButton ID="btnDelete" runat="server" Text="Delete" CommandName="Delete"
          OnClientClick="return confirm('Are your sure of deleting the current record?')" />
      </ItemTemplate>
      <InsertItemTemplate>
        <asp:LinkButton ID="btnInsert" runat="server" Text="Insert" CommandName="Insert" />
        <asp:LinkButton ID="btnCancel" runat="server" Text="Cancel" CommandName="Cancel" />
      </InsertItemTemplate>
      <EditItemTemplate>
        <asp:LinkButton ID="btnUpdate" runat="server" Text="Update" CommandName="Update" />
        <asp:LinkButton ID="btnCancel" runat="server" Text="Cancel" CommandName="Cancel" />
      </EditItemTemplate>
    </asp:TemplateField>
  </Fields>
</asp:DetailsView>
```

Now we need to write the logic under the following events of the control:

1. **ModeChanging:** fires when we click on the "New, Edit and Cancel" buttons and under this we need to change Mode of the control to "Insert Mode", "Edit Mode" and "ReadOnly Mode" respectively and all these 3 modes are defined under an enum "DetailsViewMode".
2. **ItemInserting:** fires when we click on Insert button and here we need to implement the logic for insert.
3. **ItemUpdating:** fires when we click on Update button and here we need to implement the logic for update.
4. **ItemDeleting:** fires when we click on Delete button and here we need to implement the logic for delete.

Now go to design view of the Web Form and double click on PageIndexChanging, ModeChanging, ItemInserting, ItemUpdating and ItemDeleting events of DetailsView control to generate methods for implementing the logic and write the following code under "ASPDB_Customer_DetailsView_Editing.aspx.cs" file:

Declarations:

```
Customer obj = new Customer();
```

Code under Page Load Event:

```
if (!IsPostBack) {
  LoadData();
}
```

```
private void LoadData() {
  DetailsView1.DataSource = obj.Customer_Select(null, true);
  DetailsView1.DataBind();
}
```

Code under DetailsView PageIndexChanging Event:

```
DetailsView1.PageIndex = e.NewPageIndex;
LoadData();
```

Code under DetailsView ModeChanging Event:

```
if (e.NewMode == DetailsViewMode.Insert) {
  DetailsView1.ChangeMode(DetailsViewMode.Insert);
}
else if (e.NewMode == DetailsViewMode.Edit) {
  DetailsView1.ChangeMode(DetailsViewMode.Edit);
  LoadData();
}
else {
  DetailsView1.ChangeMode(DetailsViewMode.ReadOnly);
  LoadData();
}
```

Code under DetailsView ItemInserting Event:

```
string Name = ((TextBox)DetailsView1.Rows[1].Cells[1].Controls[0]).Text;
decimal Balance = decimal.Parse(((TextBox)DetailsView1.Rows[2].Cells[1].Controls[0]).Text);
string City = ((TextBox)DetailsView1.Rows[3].Cells[1].Controls[0]).Text;
bool Status = ((CheckBox)DetailsView1.Rows[4].Cells[1].Controls[0]).Checked;
int? Custid = null;
if(obj.Customer_Insert(Name, Balance, City, Status, ref Custid) > 0) {
  DetailsView1.ChangeMode(DetailsViewMode.ReadOnly);
  LoadData();
}
else {
  Response.Write("<script>alert('Failed inserting record into the table.')</script>");
}
```

Code under DetailsView ItemUpdating Event:

```
int Custid = (int)DetailsView1.DataKey.Value;
string Name = ((TextBox)DetailsView1.Rows[1].Cells[1].Controls[0]).Text;
decimal Balance = decimal.Parse(((TextBox)DetailsView1.Rows[2].Cells[1].Controls[0]).Text);
string City = ((TextBox)DetailsView1.Rows[3].Cells[1].Controls[0]).Text;
if (obj.Customer_Update(Custid, Name, Balance, City) > 0) {
  DetailsView1.ChangeMode(DetailsViewMode.ReadOnly);
  LoadData();
}
else {
  Response.Write("<script>alert('Failed updating record in the table.')</script>");
}
```

Code under DetailsView ItemDeleting Event:

```
int Custid = (int)DetailsView1.DataKey.Value;
if (obj.Customer_Delete(Custid) > 0) {
  LoadData();
}
else {
  Response.Write("<script>alert('Failed deleting record from the table.')</script>");
}
```

FormView Control: this control lets you work with a single record from a data source, similar to the DetailsView control. The difference between the FormView and the DetailsView controls is that the DetailsView control uses a tabular layout where each field of the record is displayed as a row of its own. In contrast, the FormView control does not specify a pre-defined layout for displaying the record. Instead, you create a template that contains controls to display individual fields from the record. To work with FormView control, add a new Web Form in the project naming it as "ASPDB_Student_FormView_Demo.aspx" and write the following code under its " " tag:

```
<asp:FormView ID="FormView1" runat="server" AllowPaging="true" PagerStyle-HorizontalAlign="Center"
                                                HorizontalAlign="Center">
  <HeaderTemplate>Student Details</HeaderTemplate>
  <HeaderStyle BackColor="LightGoldenrodYellow" ForeColor="MediumVioletRed" Font-Size="X-Large"
                                                HorizontalAlign="Center" />
  <ItemTemplate>
    <table border="1">
      <tr>
        <th>Sid</th>              <th>Name</th>              <th>Class</th>
        <th>Fees</th>             <th>Is-Active</th>         <th>Photograph</th>
      </tr>
      <tr>
        <td align="center"><%# Eval("Sid") %></td>
        <td><%# Eval("Name") %></td>
        <td align="center"><%# Eval("Class") %></td>
        <td align="right"><%# Eval("Fees") %></td>
        <td align="center"><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
                              Checked='<%# Eval("Status") %>' /></td>
        <td><asp:Image ID="imgPhoto" runat="server" Width="200" Height="200"
                    ImageUrl='<%# "~/Images/" + Eval("PhotoName") %>' /></td>
      </tr>
    </table>
  </ItemTemplate>
  <FooterTemplate>
    Click to navigate:
  </FooterTemplate>
  <FooterStyle BackColor="SlateBlue" ForeColor="Tan" Font-Size="Large" />
</asp:FormView>
```

Now go to design view of the Web Form and double click on "PageIndexChanging" event to generate an Event Handler to implement the logic for navigating to next and previous records and then write the following code in "ASPDB_Student_FormView_Demo.aspx.cs" file:

```
using System.Data;
using System.Data.SqlClient;
```

Code under Page Load Event:

```
if (!IsPostBack) {
  LoadData();
}
```

```
private void LoadData() {
```

```
  SqlDataAdapter da = new SqlDataAdapter(
                    "Select Sid, Name, Class, Fees, PhotoName, Status From Student Order By Sid", ReadCS.ASPDB);
  DataSet ds = new DataSet();
  da.Fill(ds, "Student");
  FormView1.DataSource = ds;
  FormView1.DataBind();
}
```

Code Under FormView PageIndexChanging Event:

```
FormView1.PageIndex = e.NewPageIndex;
LoadData();
```

Performing editing operations using FormView control: to perform editing operations by using FormView first we need to explicitly generate the text and controls that has to be rendered for ReadOnly, Insert and Edit operations by using "ItemTemplate", "InsertItemTemplate" and "EditItemTemplate" of FormView control To test this add a new Web Form under the project naming it as "ASPDB_Customer_FormView_Editing.aspx" and write the following under its " " tag:

```
<asp:FormView ID="FormView1" runat="server" DataKeyNames="Custid" " PagerStyle-HorizontalAlign="Center"
                                        AllowPaging="true" HorizontalAlign="Center >
  <HeaderTemplate>Customer Details</HeaderTemplate>
  <HeaderStyle HorizontalAlign="Center" BackColor="LightGoldenrodYellow" ForeColor="MediumVioletRed"
                                              Font-Size="XX-Large" />
  <FooterTemplate>Click to navigate:</FooterTemplate>
  <FooterStyle BackColor="SlateBlue" ForeColor="Tan" Font-Size="Large" />
  <ItemTemplate>
    <table border="1" style="background-color:coral">
      <tr><th>Custid</th>      <th>Name</th>      <th>Balance</th>      <th>City</th>      <th>Is-Active</th></tr>
      <tr>
        <td align="center"><asp:Label ID="lblCustid" runat="server" Text='<%# Eval("Custid") %>' /></td>
        <td><%# Eval("Name") %></td>
        <td><%# Eval("Balance") %></td>
        <td><%# Eval("City") %></td>
        <td align="center"><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
                              Checked='<%# Eval("Status") %>' /></td>
      </tr>
      <tr>
        <td colspan="5" align="center" style="background-color:white">
          <asp:LinkButton ID="btnNew" runat="server" Text="New" CommandName="New" />
          <asp:LinkButton ID="btnEdit" runat="server" Text="Edit" CommandName="Edit" />
          <asp:LinkButton ID="btnDelete" runat="server" Text="Delete" CommandName="Delete"
                    OnClientClick="return confirm('Are you sure of deleting the current record?')" />
        </td>
      </tr>
    </table>
  </ItemTemplate>
  <InsertItemTemplate>
    <table border="1" style="background-color:coral">
```

```
      <tr><th>Name</th>          <th>Balance</th>          <th>City</th>          <th>Is-Active</th></tr>
      <tr>
        <td><asp:TextBox ID="txtName" runat="server" /></td>
        <td><asp:TextBox ID="txtBalance" runat="server" /></td>
        <td><asp:TextBox ID="txtCity" runat="server" /></td>
        <td align="center"><asp:CheckBox ID="cbStatus" runat="server" /></td>
      </tr>
      <tr>
        <td colspan="4" align="center" style="background-color:white">
          <asp:LinkButton ID="btnInsert" runat="server" Text="Insert" CommandName="Insert" />
          <asp:LinkButton ID="btnCancel" runat="server" Text="Cancel" CommandName="Cancel" />
        </td>
      </tr>
    </table>
  </InsertItemTemplate>
  <EditItemTemplate>
    <table border="1" style="background-color:coral">
      <tr><th>Custid</th>    <th>Name</th>    <th>Balance</th>    <th>City</th>    <th>Is-Active</th></tr>
      <tr>
        <td><%# Eval("Custid") %></td>
        <td><asp:TextBox ID="txtName" runat="server" Text='<%# Eval("Name") %>' /></td>
        <td><asp:TextBox ID="txtBalance" runat="server" Text='<%# Eval("Balance") %>' /></td>
        <td><asp:TextBox ID="txtCity" runat="server" Text='<%# Eval("City") %>' /></td>
        <td align="center"><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
                                Checked='<%# Eval("Status") %>' /></td>
      </tr>
      <tr>
        <td colspan="5" align="center" style="background-color:white">
          <asp:LinkButton ID="btnUpdate" runat="server" Text="Update" CommandName="Update" />
          <asp:LinkButton ID="btnCancel" runat="server" Text="Cancel" CommandName="Cancel" />
        </td>
      </tr>
    </table>
  </EditItemTemplate>
</asp:FormView>
```

Now go to design view of the Web Form and double click on PageIndexChanging, ModeChanging, ItemInserting, ItemUpdating and ItemDeleting events of FormView control to generate corresponding event handlers and write the following code under "ASPDB_Customer_FormView_Editing.aspx.cs" file:

Declarations:

```
Customer obj = new Customer();
```

Code under Page Load Event:

```
if (!IsPostBack) {
  LoadData();
}
```

```
private void LoadData() {
```

```
  FormView1.DataSource = obj.Customer_Select(null, true);
  FormView1.DataBind();
}
```

Code under FormView PageIndexChanging Event:

```
FormView1.PageIndex = e.NewPageIndex;
LoadData();
```

Code under FormView ModeChanging Event:

```
if (e.NewMode == FormViewMode.Insert) {
  FormView1.ChangeMode(FormViewMode.Insert);
}
else if (e.NewMode == FormViewMode.Edit) {
  FormView1.ChangeMode(FormViewMode.Edit);
  LoadData();
}
else {
  FormView1.ChangeMode(FormViewMode.ReadOnly);
  LoadData();
}
```

Code under FormView ItemInserting Event:

```
string Name = ((TextBox)FormView1.FindControl("txtName")).Text;
decimal Balance = decimal.Parse(((TextBox)FormView1.FindControl("txtBalance")).Text);
string City = ((TextBox)FormView1.FindControl("txtCity")).Text;
bool Status = ((CheckBox)FormView1.FindControl("cbStatus")).Checked;
int? Custid = null;
if (obj.Customer_Insert(Name, Balance, City, Status, ref Custid) > 0) {
  FormView1.ChangeMode(FormViewMode.ReadOnly);
  LoadData();
}
else {
  Response.Write("<script>alert('Failed inserting record into the table.')</script>");
```

Code under FormView ItemUpdating Event:

```
int Custid = (int)e.Keys["Custid"];
string Name = ((TextBox)FormView1.FindControl("txtName")).Text;
decimal Balance = decimal.Parse(((TextBox)FormView1.FindControl("txtBalance")).Text);
string City = ((TextBox)FormView1.FindControl("txtCity")).Text;
if (obj.Customer_Update(Custid, Name, Balance, City) > 0) {
  FormView1.ChangeMode(FormViewMode.ReadOnly);
  LoadData();
}
else {
  Response.Write("<script>alert('Failed updating record in the table.')</script>");
}
```

Code under FormView ItemDeleting Event:

```
int Custid = int.Parse(((Label)FormView1.FindControl("lblCustid")).Text);
if (obj.Customer_Delete(Custid) > 0) {
```

```
  LoadData();
}
else {
  Response.Write("<script>alert('Failed deleting record from the table.')</script>");
}
```

ListView Control: this control enables you to bind data items that are returned from a Data Source and display them either in the form of pages, individual records, or we can even group them. The ListView control displays data in a format that you define by using templates and styles. It is useful for data in any repeating structure, similar to the DataList and Repeater controls. However, unlike those controls, with the ListView control we can enable users to edit, insert, and delete data, and to sort and page data. To work with ListView Control add a new WebForm naming it as "ASPDB_Student_ListViewDemo1.aspx" and write the below code under its " " tag:

```
<asp:ListView ID="ListView1" runat="server">
  <ItemTemplate>
    <table border="1">
      <tr><th>Sid</th> <th>Name</th> <th>Class</th> <th>Fees</th> <th>Photo</th> <th>Is-Active</th></tr>
      <tr>
        <td><%# Eval("Sid") %></td>                                    <td><%# Eval("Name") %></td>
        <td><%# Eval("Class") %></td>                                  <td><%# Eval("Fees") %></td>
        <td><asp:Image ID="imgPhoto" runat="server" Width="200" Height="200"
                    ImageUrl='<%# "~/Images/" + Eval("PhotoName") %>' /></td>
        <td align="center"><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
                          Checked='<%# Eval("Status") %>' /></td>
      </tr>
    </table>
  </ItemTemplate>
</asp:ListView>
```

Now write the below code under "ASPDB_Student_ListViewDemo1.aspx.cs" file:

```
using System.Data;
using System.Data.SqlClient;
```

Code under Page Load Event:

```
if (!IsPostBack) {
  LoadData();
}
private void LoadData() {
  SqlDataAdapter da = new SqlDataAdapter(
                    "Select Sid, Name, Class, Fees, PhotoName, Status From Student Order By Sid", ReadCS.ASPDB);
  DataSet ds = new DataSet();
  da.Fill(ds, "Student");
  ListView1.DataSource = ds;
  ListView1.DataBind();
}
```

When we run the above Web Form it displays each record in a tabular format i.e. every record will be one table whereas if we want to display the data, record by record just like DetailsView or FormView controls we need

to enable paging and to do that we need to use the DataPager control. To test it go to source view of the WebForm and write the following code under </asp:ListView> tag:

```
<asp:DataPager ID="DataPager1" runat="server" PageSize="1" PagedControlID="ListView1">
  <Fields>
    <asp:NumericPagerField ButtonCount="5" ButtonType="Link" />
  </Fields>
</asp:DataPager>
```

- PagedControlId property is to specify to which control the paging has to be enabled.
- PageSize property is to specify the no. of records that has to be displayed at a time.
- Fields attribute is to specify the type of paging we want to enable which can either be NumericPagerField or NextPreviousPagerField.

Now when we run the Web Form it displays only a single record with paging option enabled. To navigate to next records we need to implement logic under ListView controls "PagePropertiesChanging" event as following:

```
DataPager1.SetPageProperties(e.StartRowIndex, e.MaximumRows, false);
LoadData();
```

In the above example we have displayed records, row by row in the form of "DetailsView" and "FormView" controls where as if we want to display data as a single table just like a "GridView" or "Repeater" controls we need to take the help of a LayoutTemplate with a PlaceHolder control so that ItemTemplate comes and sits under that PlaceHolder as a row.

Layout Template (Table)

Student Details

Sid	Name	Class	Fees	Photo	Status
<asp:PlaceHolder ID="ItemPlaceHolder" runat="server" />					
End of Student Data					

In the above, Layout Template is defined as a table with 2 rows and between the 2 rows we have a "PlaceHolder" control and in that "PlaceHolder", "ItemTemplate" will come and sit as a row for each record fetched from the table. To test it, add a new Web Form under the project naming it as "ASPDB_Student_ListViewDemo2.aspx" and write the following code under its " " tag:

```
<asp:ListView ID="ListView1" runat="server">
  <LayoutTemplate>
    <table border="1" align="center">
      <caption>Student Details</caption>
      <tr><th>Sid</th> <th>Name</th> <th>Class</th> <th>Fees</th> <th>Photo</th> <th>Is-Active</th></tr>
      <asp:PlaceHolder ID="ItemPlaceHolder" runat="server" />
      <tr><td colspan="6" align="center">End of table data.</td></tr>
    </table>
  </LayoutTemplate>
  <ItemTemplate>
    <tr>
```

```
      <td align="center"><%# Eval("Sid") %></td>                <td><%# Eval("Name") %></td>
      <td align="center"><%# Eval("Class") %></td>              <td align="right"><%# Eval("Fees") %></td>
      <td><asp:Image ID="imgPhoto" runat="server" Width="200" Height="200"
                                          ImageUrl='<%# "~/Images/" + Eval("PhotoName") %>' /></td>
      <td align="center"><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
                                                          Checked='<%# Eval("Status") %>' /></td>
    </tr>
  </ItemTemplate>
</asp:ListView>
```

Now write the below code under "ASPDB_Student_ListViewDemo2.aspx.cs" file:

```
using System.Data;
using System.Data.SqlClient;
```

Code under Page Load Event:

```
SqlDataAdapter da = new SqlDataAdapter(
                    "Select Sid, Name, Class, Fees, PhotoName, Status From Student Order By Sid", ReadCS.ASPDB);
DataSet ds = new DataSet();
da.Fill(ds, "Student");
ListView1.DataSource = ds;
ListView1.DataBind();
```

Performing Grouping operations with ListView Control: we use the GroupTemplate property to create a tiled layout in the ListView control. In a tiled table layout, the items are repeated horizontally in a row. The numbers of times that an item is repeated is specified by the GroupItemCount property. In our previous example ItemTemplate is included into LayoutTemplate with the help of a place holder control whereas in case of Grouping under LayoutTemplate, GroupTemplate is included and under GroupTemplate, ItemTemplate is included.

Layout Template (Table)

```
<asp:PlaceHolder ID="GroupPlaceHolder" runat="server" />
```

Group Template (TableRow)

```
<asp:PlaceHolder ID="ItemPlaceHolder" runat="server" />
```

In the above case Layout Template is a table and under it we have a "PlaceHolder" where the "Group Template" comes and sits as a row and under the "Group Template" we have a "PlaceHolder" where the "Item Template" comes and sits as a column(s) and no. of columns for each row is based on the "GroupItemCount" property of "ListView" control.

To test this, add a new Web Form in the project naming it as "ASPDB_Student_ListViewDemo3.aspx" and write the following code under its " " tag:

```
<asp:ListView ID="ListView1" runat="server" GroupItemCount="3">
  <LayoutTemplate>
    <table>
      <caption>Student Details</caption>
      <asp:PlaceHolder ID="GroupPlaceHolder" runat="server" />
    </table>
  </LayoutTemplate>
  <GroupTemplate>
```

```
    <tr><asp:PlaceHolder ID="ItemPlaceHolder" runat="server" /></tr>
  </GroupTemplate>
  <ItemTemplate>
    <td>
      <table border="1" width="100%">
        <tr>
          <td rowspan="5"><asp:Image ID="imgPhoto" runat="server" Width="220" Height="200"
                                ImageUrl='<%# "~/Images/" + Eval("PhotoName") %>' /></td>
          <td>SID: <%# Eval("Sid") %></td>
        </tr>
        <tr><td>Name: <%# Eval("Name") %></td></tr>
        <tr><td>Class: <%# Eval("Class") %></td></tr>
        <tr><td>Fees: <%# Eval("Fees") %></td></tr>
        <tr><td>Is-Active: <%# Eval("Status") %></td></tr>
      </table>
    </td>
  </ItemTemplate>
</asp:ListView>
```

Now write the below code under "ASPDB_Student_ListViewDemo3.aspx.cs" file:

```
using System.Data;
using System.Data.SqlClient;
```

Code under Page_Load Event:

```
SqlDataAdapter da = new SqlDataAdapter(
                    "Select Sid, Name, Class, Fees, PhotoName, Status From Student Order By Sid", ReadCS.ASPDB);
DataSet ds = new DataSet();
da.Fill(ds, "Student");
ListView1.DataSource = ds;
ListView1.DataBind();
```

Peforming editing operations with ListView control: add a new WebForm under the project naming it as "ASPDB_Customer_ListView_Editing.aspx" and write the following code under its " " tag:

```
<asp:ListView ID="ListView1" runat="server" DataKeyNames="Custid" InsertItemPosition="LastItem">
  <LayoutTemplate>
    <table border="1" align="center">
      <caption>Customer Details</caption>
      <tr><th>Custid</th><th>Name</th><th>Balance</th><th>City</th><th>Is-Active</th><th>Actions</th></tr>
      <asp:PlaceHolder ID="ItemPlaceHolder" runat="server" />
    </table>
  </LayoutTemplate>
  <ItemTemplate>
    <tr>
      <td align="center"><%# Eval("Custid") %></td>
      <td><%# Eval("Name") %></td>
      <td align="right"><%# Eval("Balance") %></td>
      <td><%# Eval("City") %></td>
```

```
      <td align="center"><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
                                Checked='<%# Eval("Status") %>' /></td>
      <td align="center">
        <asp:LinkButton ID="btnEdit" runat="server" Text="Edit" CommandName="Edit" />
        <asp:LinkButton ID="LinkDelete" runat="server" Text="Delete" CommandName="Delete"
                    OnClientClick="return confirm('Are you sure of deleting the record?')" />
      </td>
    </tr>
  </ItemTemplate>
  <InsertItemTemplate>
    <tr style="background-color:cornflowerblue">
      <td></td>
      <td><asp:TextBox ID="txtName" runat="server" /></td>
      <td><asp:TextBox ID="txtBalance" runat="server" /></td>
      <td><asp:TextBox ID="txtCity" runat="server" /></td>
      <td align="center"><asp:CheckBox ID="cbStatus" runat="server" /></td>
      <td align="center"><asp:LinkButton ID="btnInsert" runat="server" Text="Insert" CommandName="Insert" />
      </td>
    </tr>
  </InsertItemTemplate>
  <EditItemTemplate>
    <tr style="background-color:aquamarine">
      <td><%# Eval("Custid") %></td>
      <td><asp:TextBox ID="txtName" runat="server" Text='<%# Eval("Name") %>' /></td>
      <td><asp:TextBox ID="txtBalance" runat="server" Text='<%# Eval("Balance") %>' /></td>
      <td><asp:TextBox ID="txtCity" runat="server" Text='<%# Eval("City") %>' /></td>
      <td align="center"><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
                                Checked='<%# Eval("Status") %>' /></td>
      <td align="center">
        <asp:LinkButton ID="btnUpdate" runat="server" Text="Update" CommandName="Update" />
        <asp:LinkButton ID="btnCancel" runat="server" Text="Cancel" CommandName="Cancel" />
      </td>
    </tr>
  </EditItemTemplate>
</asp:ListView>
```

Now go to design view of the Web Form and double click on "ItemEditing", "ItemCanceling", "ItemInserting", "ItemUpdating" and "ItemDeleting" events of "ListView" control and write the below code under "ASPDB_Customer_ListView_Editing.aspx" file:

Declarations:

```
Customer obj = new Customer();
```

Code under Page Load Event:

```
if (!IsPostBack) {
  LoadData();
}
```

```
private void LoadData() {
  ListView1.DataSource = obj.Customer_Select(null, true);
```

```
  ListView1.DataBind();
}
```

Code under ListView ItemEditing Event:

```
ListView1.EditIndex = e.NewEditIndex;
LoadData();
```

Code under ListView ItemCanceling Event:

```
ListView1.EditIndex = -1;
LoadData();
```

Code under ListView ItemInserting Event:

```
string Name = ((TextBox)e.Item.FindControl("txtName")).Text;
decimal Balance = decimal.Parse(((TextBox)e.Item.FindControl("txtBalance")).Text);
string City = ((TextBox)e.Item.FindControl("txtCity")).Text;
bool Status = ((CheckBox)e.Item.FindControl("cbStatus")).Checked;
int? Custid = null;
if(obj.Customer_Insert(Name, Balance, City, Status, ref Custid) > 0)
  LoadData();
else
  Response.Write("<script>alert('Failed inserting record into the table.')</script>");
```

Code under ListView ItemUpdating Event:

```
int Custid = (int)e.Keys["Custid"];
string Name = ((TextBox)ListView1.Items[e.ItemIndex].FindControl("txtName")).Text;
decimal Balance = decimal.Parse(((TextBox)ListView1.Items[e.ItemIndex].FindControl("txtBalance")).Text);
string City = ((TextBox)ListView1.Items[e.ItemIndex].FindControl("txtCity")).Text;
if (obj.Customer_Update(Custid, Name, Balance, City) > 0) {
  ListView1.EditIndex = -1; LoadData();
}
else
  Response.Write("<script>alert('Failed updating record in the table.')</script>");
```

Code under ListView ItemDeleting Event:

```
int Custid = (int)e.Keys["Custid"];
if (obj.Customer_Delete(Custid) > 0)
  LoadData();
else
  Response.Write("<script>alert('Failed deleting record from the table.')</script>");
```

SqlDataSource Control: this control uses ADO.NET classes to interact with any database supported by ADO.NET. This includes Microsoft SQL Server (using the System.Data.SqlClient provider), System.Data.OleDb, System.Data.Odbc, and Oracle (using the System.Data.OracleClient provider).

Using a SqlDataSource control allows you to access and manipulate data in an ASP.NET page without using ADO.NET classes directly. You provide a connection string to connect to your database and define the SQL statements or stored procedures that work with your data. At run time, the SqlDataSource control automatically opens the database connection, executes the SQL statement or stored procedure, returns the selected data (if any), and then closes the connection. It also supports performing insert, update and delete operations also.

To work with SqlDataSource control add a new WebForm naming it as "ASPDB_Customer_DataSource_Demo.aspx" and place a SqlDataSource control on it, now we need to configure the control with required data source and to do that click on the Configure Button present at the top right corner and select the option Configure Data Source, which opens a window and click New Connection button on it, which opens a new window choose the Data Source as Microsoft Sql Server and click ok on it, which opens a new window to enter connection details, enter the connection details and choose the Database as ASPDB and click ok, now click on the next button which asks for saving the connection string into config file, select yes and click next, which takes to another window asking for writing a custom Sql Statement or choosing a Stored Procedure or to specify columns from any particular table or view => choose this option and below that we find all the tables in our database, choose "Customer" and select the required columns and beside that we find buttons to add where clause, order by clause and a button "Advanced", when we click on this button it will ask for generating Insert, Update and Delete statements, select the CheckBox and click ok, now click on the Next button where we can test our Query and click finish to complete the configuration.

Now on the WebForm place a GridView conrol and click on the Configure button present at top right corner which opens a window, in that under the Choose Data Source: select => SqlDataSource1 we configured right now and in the below it will ask for "Enable Paging", "Enable Sorting", "Enable Editing" and "Enable Deleting", choose the required options by selecting the CheckBox's and run the WebForm which display the data by providing all options we have choosen. Now if we go to the Source View of the WebForm we will find all the code that is generated by the SqlDataSource control.

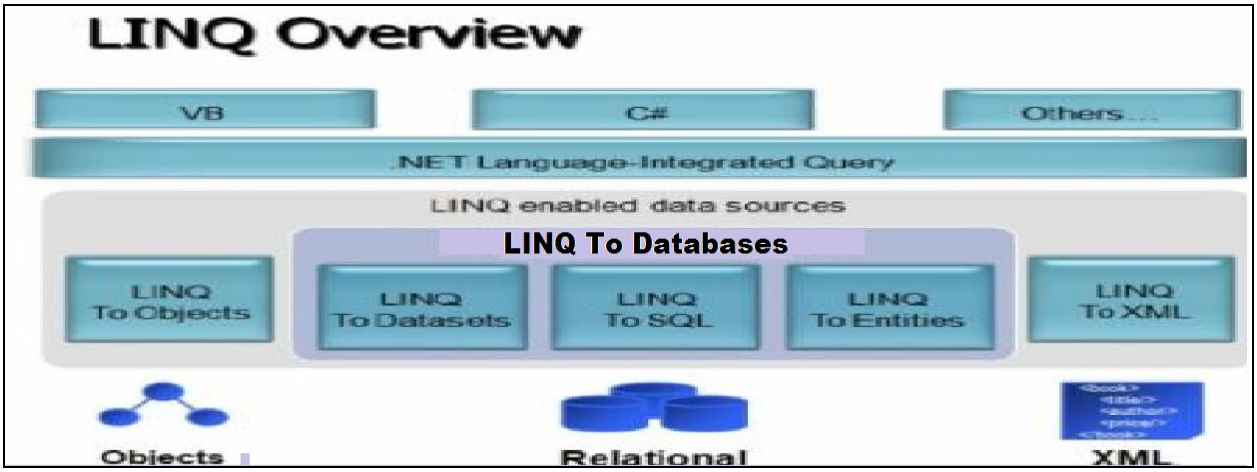
Chart Control: this control enables you to create ASP.NET pages or Windows Forms applications with simple, intuitive, and visually compelling charts for complex statistical or financial analysis.

To work with Chart control add a new WebForm naming it as "ASPDB_Employee_Chart_Demo.aspx", place a Chart Control on the WebForm and click on the button present at top right corner which opens a window, in that under Choose Data Source: select "New Data Source" Option which opens a window and in that select "Sql Database" option and in the below specify an Id for the data source as "EmployeeDS" and click ok which opens a window asking for choosing Data Source, below that we find a ComboBox showing the list of databases configured under Server Explorer and if our database is in that list select it or click on the "New Connection" button to configure with a new data source and click Next button, and in that window choose the option "Specify a custom SQL Statement or Stored Procedure radio button and click next, in that under the Sql Statements option write the following query => "Select Did, TotalSalary = Sum(Salary) From Employee Group By Did Order By Did" and click next and click finish, so that chart control uses "EmployeeDS" we have created now and below that choose the chart type we want like column, pie, line etc, and below that under "X Value Member" choose "Did" column and under "Y Value Member" choose "TotalSalary" and run the WebForm to watch the Chart.

LINQ (Language Integrated Query)

In C# 3.0 .Net has introduced a new language known as "LINQ" much like SQL (which we use universally with relational databases to perform queries). LINQ allows you to write query expressions (similar to SQL Queries) that can retrieve information from a wide variety of data sources like Objects, Databases and XML.

LINQ stands for Language Integrated Query. LINQ is a data querying methodology which provides querying capabilities to .NET languages with syntax similar to a SQL Query. LINQ has a great power of querying on any source of data, where the data source could be collections of objects, database or XML files.



LINQ to Objects: used to perform queries against the in-memory data like an array or collection.

LINQ to Databases:

- LINQ to DataSet is used to perform queries against ADO.NET Data Table's.
- LINQ to SQL is used to perform queries against the relation database, but only Microsoft SQL Server.
- LINQ to Entities is used to perform queries against any relation database like SQL Server, Oracle, etc.

LINQ to XML (XLinq): used to perform queries against the XML source.

Advantages of LINQ:

- LINQ offers an object-based, language-integrated way to query over data, no matter where that data come from. So through LINQ we can query on Database, XML Source as well as Collections & Arrays.
- Compile time syntax checking.
- It allows you to query collections, arrays, and classes etc. in the native language of our application like VB or C#.

LINQ to Objects

Using this we can perform queries against the in-memory data like an array or collection and filter or sort the information under them. Syntax of the query we want to use on objects will be as following:

from <alias> in <array_name | collection_name> [<clauses>] select <alias> | new { <list of columns> }

- Linq queries start with from keyword and ends with select keyword.
- While writing conditions we need to use the alias name we have specified if column names are not present, just like we use column names in case of SQL.
- Clauses can be like where, groupby and orderby.
- To use LINQ in your application first we need to import System.Linq namespace.

To work with LINQ, open a new "ASP.Net Web Application" project naming it as "LinqExamples", add a Web Form in the project naming it as "LinqWithArray.aspx" and write the following code under its " " tag:

```
<asp:Button ID="Button1" runat="server" Text="Get values > 40 using Imperative Code" Width="300px" /><br />
<asp:Label ID="Label1" runat="server" ForeColor="Red" /><br />
<asp:Button ID="Button2" runat="server" Text="Get values > 40 using Declarative Code" Width="300px" /><br />
<asp:Label ID="Label2" runat="server" ForeColor="Red" />
```

Now write the below code in "LinqWithArray.aspx.cs" file:

Declarations:

```
int[] arr = { 13, 56, 29, 74, 31, 45, 91, 7, 34, 86, 24, 62, 19, 3, 98, 16, 36, 41, 52, 83, 38, 79, 67, 27, 5 };
```

Code under Button1 Click Event:

```
int Count = 0, Index = 0;
foreach (int i in arr) {
  if (i > 40) {
    Count += 1;
  }
}
int[] brr = new int[Count];
foreach (int i in arr) {
  if (i > 40) {
    brr[Index] = i;
    Index += 1;
  }
}
Array.Sort(brr);
Array.Reverse(brr);
Label1.Text = String.Join(" ", brr);
```

Code under Button2 Click Event:

```
var brr = from i in arr where i > 40 orderby i descending select i;
Label2.Text = String.Join(" ", brr);
```

Now to test Linq with Collections add a new Web Form naming it as "LinqWithCollections.aspx" and write the following code under "LinqWithCollections.aspx.cs" file:

Code under Page_Load Event:

```
List<string> Cities = new List<string>() { "Amaravati", "Itanagar", "Dispur", "Patna", "Panaji", "Gandhinagar",
"Shimla", "Srinagar", "Bangalooru", "Thiruvananthapuram", "Bhopal", "Mumbai", "Imphal","Shillong", "Aizawl",
"Kohima", "Bhubaneswar", "Chandigarh", "Jaipur", "Gangtok", "Chennai", "Agartala","Lucknow", "Kolkata",
"Raipur", "Dehradun", "Ranchi", "Hyderabad", "Delhi" };

//Query to fetch all the Cities as is:
var Coll1 = from s in Cities select s;
Response.Write(String.Join(", ", Coll1) + "<hr />");

//Query to fetch all the Cities in ascending order:
var Coll2 = from s in Cities orderby s select s;
Response.Write(String.Join(", ", Coll2) + "<hr />");
//Query to fetch all the Cities in descending order:
var Coll3 = from s in Cities orderby s descending select s;
Response.Write(String.Join(", ", Coll3) + "<hr />");

//Query to fetch all the Cities with a length of 6 characters:
var Coll4 = from s in Cities where s.Length == 6 select s;
Response.Write(String.Join(", ", Coll4) + "<hr />");

//Query to fetch cities starting with a particular character:
var Coll5 = from s in Cities where s.IndexOf('A') == 0 select s;
Response.Write(String.Join(", ", Coll5) + "<hr />");
var Coll6 = from s in Cities where s.Substring(0, 1) == "B" select s;
Response.Write(String.Join(", ", Coll6) + "<hr />");
var Coll7 = from s in Cities where s.StartsWith("C") select s;
Response.Write(String.Join(", ", Coll7) + "<hr />");
var Coll8 = from s in Cities where s[0] == 'D' select s;
Response.Write(String.Join(", ", Coll8) + "<hr />");

//Query to fetch cities ending with a particular character:
var Coll9 = from s in Cities where s.IndexOf('a') == s.Length - 1 select s;
Response.Write(String.Join(", ", Coll9) + "<hr />");
var Coll10 = from s in Cities where s.Substring(s.Length - 1) == "i" select s;
Response.Write(String.Join(", ", Coll10) + "<hr />");
var Coll11 = from s in Cities where s.EndsWith("l") select s;
Response.Write(String.Join(", ", Coll11) + "<hr />");
var Coll12 = from s in Cities where s[s.Length - 1] == 'r' select s;
Response.Write(String.Join(", ", Coll12) + "<hr />");

//Query to fetch cities containing character “a” in third place:
var Coll13 = from s in Cities where s.IndexOf('a') == 2 select s;
Response.Write(String.Join(", ", Coll13) + "<hr />");
var Coll14 = from s in Cities where s.Substring(2, 1) == "a" select s;
Response.Write(String.Join(", ", Coll14) + "<hr />");
var Coll15 = from s in Cities where s[2] == 'a' select s;
```

```
Response.Write(String.Join(", ", Coll15) + "<hr />");

//Query to fetch cities containing character “h” or “H” under them:
var Coll16 = from s in Cities where s.IndexOf('h') >= 0 || s.IndexOf('H') >= 0 select s;
Response.Write(String.Join(", ", Coll16) + "<hr />");
var Coll17 = from s in Cities where s.ToLower().IndexOf('h') >= 0 select s;
Response.Write(String.Join(", ", Coll17) + "<hr />");
var Coll18 = from s in Cities where s.Contains('h') || s.Contains('H') select s;
Response.Write(String.Join(", ", Coll18) + "<hr />");
var Coll19 = from s in Cities where s.ToUpper().Contains('H') select s;
Response.Write(String.Join(", ", Coll19) + "<hr />");

//Query to fetch cities not containing character “h” or “H” under them:
var Coll20 = from s in Cities where s.IndexOf('h') == -1 && s.IndexOf('H') == -1 select s;
Response.Write(String.Join(", ", Coll20) + "<hr />");
var Coll21 = from s in Cities where s.ToLower().IndexOf('h') == -1 select s;
Response.Write(String.Join(", ", Coll21) + "<hr />");
var Coll22 = from s in Cities where s.Contains('h') == false && s.Contains('H') == false select s;
Response.Write(String.Join(", ", Coll22) + "<hr />");
var Coll23 = from s in Cities where s.ToUpper().Contains('H') == false select s;
Response.Write(String.Join(", ", Coll23) + "<hr />");
```

Note: the values that are returned by LINQ queries can be captured by using implicitly typed local variables, so in the above programs “brr and Coll” are implicitly declared array/collection which stores values retrieved by the query.

In traditional process of filtering data from an array or collection we have repetition statements that filter arrays focusing on the process of getting the results i.e. iterating through the elements and checking whether they satisfy the desired criteria, whereas LINQ specifies, not the steps necessary to get the results, but rather the conditions that selected elements must satisfy and this is known as **declarative programming**- as opposed to **imperative programming** (which we’ve been using so far) in which we specify the actual steps to perform a task. Procedural & Object Oriented Languages are a subset of imperative.

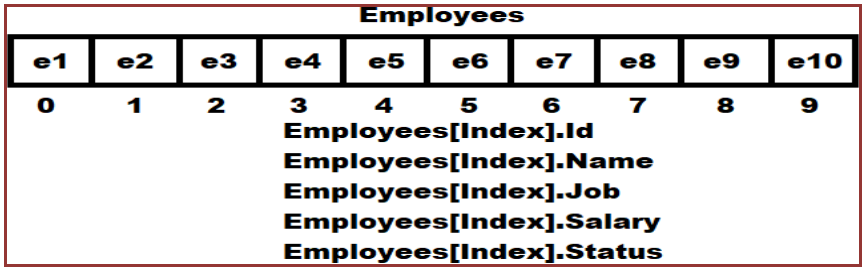
The queries we have used above specifies that the result should consist of all the integers in the List that are greater than 40, but it does not specify how to obtain the result, the C# compiler generates all the necessary code automatically, which is one of the great strengths of LINQ.

LINQ Providers: the syntax of LINQ is built into the language, but LINQ queries may be used in many different contexts because of libraries known as providers. A LINQ provider is a set of classes that implement LINQ operations and enable programs to interact with data sources to perform tasks such as sorting, grouping and filtering elements. When we import the “System.Linq” namespace it contains the LINQ to Objects provider, without importing it the compiler cannot locate a provider for the LINQ queries and issues errors on LINQ queries.

Storing user-defined type values in Collection: the type of values being stored in a Generic Collection can be of user-defined types also, like a class type or structure type that is defined to represent an entity. To test this, add a new class under the project naming it as “Employee.cs” and write the below code in it:

```
public class Employee {
  public int Id { get; set; }
  public string Name { get; set; }
  public string Job { get; set; }
  public double Salary { get; set; }
  public bool Status { get; set; }
}
```

Now we can store instances of Employee type under the List just like we stored String type instances in our previous example but the difference is, string is a scalar type and it's instance represents a single value whereas Employee is a complex type and it's instance is internally a collection of different attributes like Id, Name, Job, Salary and Status, and we can also write queries on this list and access data based on the Employee attributes.



To test this process go to "Global.asax" file and write the below code under Application_Start Method:

```
Employee e1 = new Employee { Id = 1001, Name = "Naresh", Job = "President", Salary = 12000.00, Status = true };
Employee e2 = new Employee { Id = 1002, Name = "Raju", Job = "Manager", Salary = 10000.00, Status = true };
Employee e3 = new Employee { Id = 1003, Name = "Ajay", Job = "Salesman", Salary = 6000.00, Status = true };
Employee e4 = new Employee { Id = 1004, Name = "James", Job = "Salesman", Salary = 6000.00, Status = true };
Employee e5 = new Employee { Id = 1005, Name = "Jones", Job = "Clerk", Salary = 3000.00, Status = true };
Employee e6 = new Employee { Id = 1006, Name = "Scott", Job = "Manager", Salary = 10000.00, Status = true };
Employee e7 = new Employee { Id = 1007, Name = "Smith", Job = "Analyst", Salary = 8000.00, Status = true };
Employee e8 = new Employee { Id = 1008, Name = "Vijay", Job = "Analyst", Salary = 8000.00, Status = true };
Employee e9 = new Employee { Id = 1009, Name = "Venkat", Job = "Clerk", Salary = 3000.00, Status = true };
Employee e10 = new Employee { Id = 1010, Name = "Satish", Job = "Admin", Salary = 5000.00, Status = true };
List<Employee> Employees = new List<Employee>() { e1, e2, e3, e4, e5, e6, e7, e8, e9, e10 };
Application["Employees"] = Employees;
```

Now add a new WebForm under the project naming it as "LinqWithComplexTypes.aspx" and write the following code under its tag:

```
<table align="center">
  <tr>
    <td align="center"><asp:DropDownList ID="ddlJobs" runat="server" AutoPostBack="true" /></td>
  </tr>
  <tr>
    <td><asp:GridView ID="gvEmp" runat="server" /></td>
  </tr>
</table>
```

Now write the below code under "LinqWithComplexTypes.aspx.cs" file:

Declarations:

```
List<Employee> Employees;
```

Code under Page Load Event:

```
Employees = (List<Employee>)Application["Employees"];
if (!IsPostBack) {
  ddlJobs.DataSource = (from E in Employees select new { E.Job }).Distinct();
  ddlJobs.DataTextField = "Job";
  ddlJobs.DataValueField = "Job";
  ddlJobs.DataBind();
  ddlJobs.Items.Insert(0, "-Select Job-");
  gvEmp.DataSource = Employees;
  gvEmp.DataBind();
}
```

Code under DropDownList SelectedIndexChanged Event:

```
if (ddlJobs.SelectedIndex > 0) {
  gvEmp.DataSource = from E in Employees where E.Job == ddlJobs.SelectedValue select E;
}
else {
  gvEmp.DataSource = from E in Employees select E;
}
gvEmp.DataBind();
```

LINQ to XML (XLINQ)

Using this we can write queries on XML Source or XML Data just like we have written queries on Arrays and Collections. To work with LINQ to XML first we need to import the namespace “System.Xml.Linq” which provides all the classes for writing queries on XML Source. In this namespace we are provided with a class known as “XElement” which is actually responsible for loading an XML Document into our application and then we need to write the query on loaded document. To test this first add a XML File under the project naming it as “Travel.xml” and write the following code in it:

```
<Locations>
  <Location>
    <Name>Bangkok</Name>
    <Continent>Asia</Continent>
    <Video>Bangkok.mp4</Video>
    <Image>Bangkok.jpg</Image>
    <Price>25000</Price>
  </Location>
  <Location>
    <Name>Dubai</Name>
    <Continent>Asia</Continent>
    <Video>Dubai.mp4</Video>
    <Image>Dubai.jpg</Image>
    <Price>22000</Price>
  </Location>
  <Location>
    <Name>Hong Kong</Name>
    <Continent>Asia</Continent>
    <Video>HongKong.mp4</Video>
    <Image>HongKong.jpg</Image>
    <Price>35000</Price>
  </Location>
  <Location>
    <Name>London</Name>
    <Continent>Europe</Continent>
```

```
    <Video>London.mp4</Video>
    <Image>London.jpg</Image>
    <Price>50000</Price>
  </Location>
  <Location>
    <Name>Mumbai</Name>
    <Continent>Asia</Continent>
    <Video>Mumbai.mp4</Video>
    <Image>Mumbai.jpg</Image>
    <Price>15000</Price>
  </Location>
  <Location>
    <Name>New York</Name>
    <Continent>North America</Continent>
    <Video>NewYork.mp4</Video>
    <Image>NewYork.jpg</Image>
    <Price>60000</Price>
  </Location>
  <Location>
    <Name>Paris</Name>
    <Continent>Europe</Continent>
    <Video>Paris.mp4</Video>
    <Image>Paris.jpg</Image>
    <Price>45000</Price>
  </Location>
  <Location>
    <Name>Rio de Janeiro</Name>
    <Continent>South America</Continent>
    <Video>RiodeJaneiro.mp4</Video>
    <Image>RiodeJaneiro.jpg</Image>
    <Price>55000</Price>
  </Location>
  <Location>
    <Name>Singapore</Name>
    <Continent>Asia</Continent>
    <Video>Singapore.mp4</Video>
    <Image>Singapore.jpg</Image>
    <Price>30000</Price>
  </Location>
  <Location>
    <Name>Sydney</Name>
    <Continent>Australia</Continent>
    <Video>Sydney.mp4</Video>
    <Image>Sydney.jpg</Image>
    <Price>75000</Price>
  </Location>
```

```
</Locations>
```

Now create 2 folders under the project naming them as “Images” and “Videos”. Copy an image corresponding to each city into Images folder and a video corresponding of each city into Videos folder. Add a new WebForm under the project naming it as “Travel.aspx”. Drag and Drop the “VideoPlayer” (we created this earlier in Controls Development Classes) control on to the Web Form and delete it and when we do this it will write the below statement on top of the page:

```
<%@ Register Assembly="AspControls" Namespace="AspControls" TagPrefix="cc1" %>
```

Now write the following code under the tag of the Web Form:

```
<h1 style="background-color: yellow; color: red; text-align: center">Travel Analog</h1>
<table align="center">
  <tr>
    <td align="center">
      <asp:DropDownList ID="ddlContinents" runat="server" AutoPostBack="true" />
    </td>
  </tr>
  <tr>
    <td>
      <asp:Repeater ID="rptLocations" runat="server">
        <HeaderTemplate>
          <table border="1" align="center">
            <tr>
              <th>Location</th>    <th>Continent</th>    <th>Video</th>    <th>Image</th>    <th>Price</th>
            </tr>
        </HeaderTemplate>
        <ItemTemplate>
          <tr>
            <td><%# Eval("Name") %></td>
            <td><%# Eval("Continent") %></td>
            <td>
              <cc1:VideoPlayer ID="VideoPlayer1" runat="server" Width="300" Height="200" Controls="true"
                                    Mp4Url='<%# "/Videos/" + Eval("Video") %>' />
            </td>
            <td>
              <asp:Image ID="Image1" runat="server" Width="200" Height="200"
                                                          ImageUrl='<%# "/Images/" + Eval("Photo") %>' />
            </td>
            <td align="right"><%# Eval("Price") %></td>
          </tr>
        </ItemTemplate>
        <FooterTemplate>
          </table>
        </FooterTemplate>
      </asp:Repeater>
    </td>
  </tr>
```

```
</table>
```

Now write the below code under "Travel.aspx.cs" file:

```
using System.Xml.Linq;
```

Code under Page_Load Event:

```
if (!IsPostBack) {
  var data = XElement.Load(Server.MapPath("Travel.xml"));
  var table = data.Elements("Location");
  var Continents = from t in table select new { Continent = t.Element("Continent").Value };
  ddlContinents.DataSource = Continents.Distinct();
  ddlContinents.DataTextField = "Continent";
  ddlContinents.DataValueField = "Continent";
  ddlContinents.DataBind();
  ddlContinents.Items.Insert(0, "-Select Continent-");
  rptLocations.DataSource = from t in table select new { Name = t.Element("Name").Value,
        Continent = t.Element("Continent").Value, Video = t.Element("Video").Value,
        Photo = t.Element("Image").Value, Price = t.Element("Price").Value };
  rptLocations.DataBind();
}
```

Code under DropDownList SelectedIndexChanged Event:

```
var data = XElement.Load(Server.MapPath("Travel.xml"));
var table = data.Elements("Location");
if(ddlContinents.SelectedIndex == 0) {
  rptLocations.DataSource = from t in table select new { Name = t.Element("Name").Value,
        Continent = t.Element("Continent").Value, Video = t.Element("Video").Value,
        Photo = t.Element("Image").Value, Price = t.Element("Price").Value };
}
else {
  rptLocations.DataSource = from t in table where t.Element("Continent").Value ==
        ddlContinents.SelectedValue select new { Name = t.Element("Name").Value,
        Continent = t.Element("Continent").Value, Video = t.Element("Video").Value,
        Photo = t.Element("Image").Value, Price = t.Element("Price").Value };
}
rptLocations.DataBind();
```

LINQ to Databases

This is designed for working with relational databases like Sql Server, Oracle, etc and this is the biggest and most exciting addition to the .NET Framework 3.5. Basically, what LINQ provides is a lightweight interface over programmatic data integration. This is such a big deal because **Data is King**. Pretty much every application deals with data in some manner, whether that data comes from memory, databases, XML files, text files, or something else. Many developers find it very difficult to move from the strongly typed object-oriented world of .NET languages to the data tier where objects are second-class citizens. The transition from one world to the next was a kludge at best and was full of error-prone actions. In .NET, programming with objects means a wonderful strongly typed ability to work with code. You can navigate very easily through the namespaces; work with a debugger in the Visual Studio IDE, and more. However, when you have to access data, you will notice that things are dramatically different. You end up in a world that is not strongly typed, where debugging is a pain or even non-existent, and you end up spending most of the time sending strings to the database as commands. As a developer, you also have to be aware of the underlying data and how it is.

Advantages of LINQ:

- LINQ is a part of .NET Framework so we can use any .NET Language for writing LINQ Queries, so that we can write all the Queries in the syntax of the .NET Language we are working with.
- It is "Type Safe" i.e. we get data values from database in the similar type of the columns "data type" whereas in ADO.Net we receive all data values as "object type" only.
- Full support of intellisense while writing the queries and also supports debugging.
- Compile-time syntax checking rather than runtime syntax checking.
- It is a pure object oriented way of communication with Data Sources where as in ADO.Net we used SQL for communication with Databases which is a blend of Relational and Object Oriented Code i.e. in LINQ, Tables (Entities) are Classes, Columns (Attributes) are Properties of Classes, Records or Rows are Instances of Classes and Stored Procedures are Methods.

LINQ to Databases is again divided into 2 parts:

1. LINQ to SQL
2. LINQ to Entities or Entity Framework

Working with LINQ to SQL:

LINQ to SQL in particular is a means to have a strongly typed interface against a SQL Server Database. You will find the approach that LINQ to SQL provides is by far the easiest approach for querying SQL Server available at the moment. It is important to remember that LINQ to SQL is not only about querying data, but you can also

perform Insert/Update/Delete operations that you need to perform which are known as CRUD operations (Create(Insert)/Read(Select)/Update/Delete).

To work with "Linq to Sql" first we need to convert relational objects in DB into object oriented types under the language and the process of this conversion is known as ORM (Object Relational Mapping) and to do this we are provided with 2 tools under Visual Studio which does an outstanding job of making it as easy as possible.

1. **OR (Object-Relational) Designer**
2. **EDM (Entity Data Model)**

Note: "OR-Designer" is used in "LINQ to SQL" and "EDM" is used in "LINQ to Entities".

Working with O/R Designer: to start working with "LINQ to SQL" open a new ASP.Net Web Application Project naming it as "LinqToSqlProject", then open the "Add New Item" Window for adding OR-Designer into the project which is found with the name "LINQ to SQL Classes" that adds a new item with the extension .dbml (Database Markup Language). We can give any name to the .dbml item but it is always suggested to use our Database name as a name to this, so let us name this as ASPDB.dbml as our database name is ASPDB and click "Add" button which will do the following:

1. Adds a reference to "System.Data.Linq" assembly which is required to work with "LINQ to Sql".
2. Under Solution Explorer we will find "ASPDB.dbml" and under it we will find 2 sub-items "ASPDB.dbml.layout" and "ASPDB.Designer.cs" and under this file only OR-Designer writes all the ORM code converting Relational Objects into Object Oriented Types.
3. The O/R Designer is added in the studio which will appear as a tab within the document window directly in the IDE and this is made up of two parts. The first part on the left is for Data Classes, which map to Tables, Views, etc, dragging such items on this surface will give us a visual representation of those objects that can be worked with. The second part on the right is for Methods, which map to the Stored Procedures within the DB.

If we look into the code of "ASPDB.designer.cs" file we will find a class ASPDBDataContent inheriting from DataContext class; we can view this class something that maps to a Connection class binding with the DB. This class works with the connection string and connects to the database for any required operations when we create instance of the class. This class also provides methods like CreateDatabase, DeleteDatabase, GetTable, ExecuteCommand, ExecuteQuery, SubmitChanges etc inherited from its parent, using which we can perform action directly on the DB. Currently in this class we find 4 parameterized constructors for creating instance of the class.

Creating the Customer Class to work with Customer Table: for this example, let's work with the Customer Table (Entity) from our ASPDB database, which means that we are going to create a Customer Class (Entity) that will create a map to Customer table. To accomplish this task simply open the "Server Explorer" within Visual Studio, configure our ASPDB Database under it, and drag and drop the Customer table onto the design surface of O/R Designer in LHS which will add a bunch of code in to ASPDB.designer.cs file on our behalf with a Customer Class in it, and this class will give us a strongly typed access to Customer Table (Entity). When we drag and drop a table on OR Designer the following actions gets performed internally:

1. Defines a class representing the table (Entity) we have dragged and dropped on the OR Designer where the name of the class will be same as the table name, as we dropped the Customer table on OR Designer Customer class gets defined.
2. Defines properties under the class defined representing the table (Entity), where each property represents each column of the table.

3. Defines a property under the ASPDBDataContext class for referring to the table we are working with and the type of the property will be Table<Entity>, for example because we are working with Customer Entity the property name will be Customers and the type of the property will be Table<Customer>.

Note: Table<Entity> is a generic class under System.Data.Linq namespace which contains a set of methods DeleteOnSubmit, InsertOnSubmit, SingleOrDefault etc. for performing the CRUD operations.

Apart from the above 3 when we place the first object on OR Designer it will also perform these activities:

1. Writes the Connection String in the Web.config file targeting to the database we are working with.
2. Defines a new parameter less constructor under the ASPDBDataContext class and we can use this for creating the instance for connecting to DB and it only will read the connection string from Web.config.

Add a new Web Form in the project naming it as "DisplayData.aspx" and write the below code in its tag:

```
<table align="center">
  <tr>
    <td align="center"><asp:DropDownList ID="ddlCities" runat="server" AutoPostBack="true" /></td>
  </tr>
  <tr>
    <td><asp:GridView ID="gvCustomers" runat="server" /></td>
  </tr>
</table>
```

Now write the below code in "DisplayData.aspx.cs" file:

Declarations:

```
ASPDBDataContext dc = new ASPDBDataContext();
```

Code under Page Load Event:

```
if(!IsPostBack) {
  ddlCities.DataSource = (from C in dc.Customers select new { C.City }).Distinct();
  ddlCities.DataTextField = "City";
  ddlCities.DataValueField = "City";
  ddlCities.DataBind();
  ddlCities.Items.Insert(0, "-Select City-");
  gvCustomers.DataSource = dc.Customers;
  gvCustomers.DataBind();
}
```

Code under DropDownList SelectedIndexChanged Event:

```
if (ddlCities.SelectedIndex == 0) {
  gvCustomers.DataSource = dc.Customers;
}
else {
  gvCustomers.DataSource = from C in dc.Customers where C.City == ddlCities.SelectedValue select C;
}
gvCustomers.DataBind();
```

Note: "ASPDBDataConext" class instance when created will read the "Connection String" from "Web.config" file and connects to our "ASPDB" Database. "Customers" is a property under "ASPDBDataContext" class which provides access to the data in the form of a Table i.e., each record from the Database table comes and adds into this Table in the form of an instance and we can use this Table for performing CRUD operations on Database table.

Performing CRUD operations using LINQ: To perform CRUD operations by using LINQ, we need to follow the below steps for each operation.

Steps for Inserting:

1. Create an instance of Customer class, which is defined representing the Customer Entity because each instance is a record and assign values to the properties of that instance, because those properties represents columns.
2. Call InsertOnSubmit method on the table i.e., Customers which adds the record into the table (local copy) in a pending state.
3. Call SubmitChanges method on DataContext object for saving the changes to DB Server.

Steps for Updating:

1. Create a reference of the Customer Entity (class) that must be updated by calling Single or First or SingleOrDefault or FirstOrDefault methods on the table i.e., Customers.
2. Re-assign values to properties of the reference so that old values get changed to new values.
3. Call SubmitChanges method on DataContext object for saving the changes to DB server.

Steps for Deleting:

1. Create a reference of the Customer Entity (class) that must be deleted by calling Single or First or SingleOrDefault or FirstOrDefault method on the table i.e., Customers.
2. Call DeleteOnSubmit method on the table i.e., Customers that deletes the record from table (local copy) in a pending state.
3. Call SubmitChanges method on DataContext object for saving the changes to DB server.

To perform CRUD operations, add a new WebForm in the project naming it as "CRUDWithLinq.aspx" and write the following code under its tag:

```
<asp:ListView ID="ListView1" runat="server" DataKeyNames="Custid" InsertItemPosition="LastItem">
  <LayoutTemplate>
    <table border="1" align="center">
      <tr> <th>Custid</th> <th>Name</th> <th>Balance</th> <th>City</th> <th>Status</th> <th>Action</th> </tr>
      <asp:PlaceHolder ID="ItemPlaceHolder" runat="server" />
    </table>
  </LayoutTemplate>
  <ItemTemplate>
    <tr>
      <td><%# Eval("Custid") %></td>                    <td><%# Eval("Name") %></td>
      <td><%# Eval("Balance") %></td>                   <td><%# Eval("City") %></td>
      <td align="center">
        <asp:CheckBox ID="cbStatus" runat="server" Enabled="false" Checked='<%# Eval("Status") %>' /></td>
      <td align="center">
        <asp:LinkButton ID="btnEdit" runat="server" Text="Edit" CommandName="Edit" />
        <asp:LinkButton ID="btnDelete" runat="server" Text="Delete" CommandName="Delete"
                                    OnClientClick="return confirm('Are you sure of deleting the record?')" />
      </td>
```

```
    </tr>
  </ItemTemplate>
  <InsertItemTemplate>
    <tr style="background-color:aquamarine">
      <td><asp:TextBox ID="txtCustid" runat="server" Width="50" /></td>
      <td><asp:TextBox ID="txtName" runat="server" Width="150" /></td>
      <td><asp:TextBox ID="txtBalance" runat="server" Width="150" /></td>
      <td><asp:TextBox ID="txtCity" runat="server" Width="150" /></td>
      <td align="Center"><asp:CheckBox ID="cbStatus" runat="server" /></td>
      <td align="Center">
        <asp:LinkButton ID="btnInsert" runat="server" Text="Insert" CommandName="Insert" /></td>
    </tr>
  </InsertItemTemplate>
  <EditItemTemplate>
    <tr style="background-color:burlywood">
      <td><%# Eval("Custid")%></td>
      <td><asp:TextBox ID="txtName" runat="server" Width="150" Text='<%# Eval("Name")%>' /></td>
      <td><asp:TextBox ID="txtBalance" runat="server" Width="150" Text='<%# Eval("Balance")%>' /></td>
      <td><asp:TextBox ID="txtCity" runat="server" Width="150" Text='<%# Eval("City")%>' /></td>
      <td align="Center">
        <asp:CheckBox ID="cbStatus" runat="server" Enabled="false" Checked='<%# Eval("Status")%>' /></td>
      <td align="center">
        <asp:LinkButton ID="btnUpdate" runat="server" Text="Update" CommandName="Update" />
        <asp:LinkButton ID="btnCancel" runat="server" Text="Cancel" CommandName="Cancel" />
      </td>
    </tr>
    </EditItemTemplate>
</asp:ListView>
```

Go to design view of Web Form, double click on ItemEditing, ItemCanceling, ItemInserting, ItemUpdating, and ItemDeleting events of ListView control and write the below code in "CRUDWithLinq.aspx.cs" file:

Declarations:

```
ASPDBDataContext dc = new ASPDBDataContext();
```

Code under Page Load Event:

```
if (!IsPostBack) {
  LoadData();
}
```

```
private void LoadData() {
  ListView1.DataSource = from C in dc.Customers where C.Status == true select C;
  ListView1.DataBind();
}
```

Code under ListView ItemInserting Event:

```
int Id = int.Parse(((TextBox)e.Item.FindControl("txtCustid")).Text);
string Name = ((TextBox)e.Item.FindControl("txtName")).Text;
decimal Balance = decimal.Parse(((TextBox)e.Item.FindControl("txtBalance")).Text);
```

```
string City = ((TextBox)e.Item.FindControl("txtCity")).Text;
bool Status = ((CheckBox)e.Item.FindControl("cbStatus")).Checked;
Customer obj = new Customer { Custid = Id, Name = Name, Balance = Balance, City = City, Status = Status };
dc.Customers.InsertOnSubmit(obj);
dc.SubmitChanges();
LoadData();
```

Code under ListView ItemEditing Event:

```
ListView1.EditIndex = e.NewEditIndex;
LoadData();
```

Code under ListView ItemCanceling Event:

```
ListView1.EditIndex = -1;
LoadData();
```

Code under ListView ItemUpdating Event:

```
int Id = (int)e.Keys["Custid"];
string Name = ((TextBox)ListView1.Items[e.ItemIndex].FindControl("txtName")).Text;
decimal Balance = decimal.Parse(((TextBox)ListView1.Items[e.ItemIndex].FindControl("txtBalance")).Text);
string City = ((TextBox)ListView1.Items[e.ItemIndex].FindControl("txtCity")).Text;
Customer obj = dc.Customers.Single(C => C.Custid == Id);
obj.Name = Name;                obj.Balance = Balance;                obj.City = City;
dc.SubmitChanges();
ListView1.EditIndex = -1;
LoadData();
```

Code under ListView ItemDeleting Event:

```
int Id = (int)e.Keys["Custid"];
Customer obj = dc.Customers.Single(C => C.Custid == Id);
obj.Status = false;
dc.SubmitChanges();
LoadData();
```

Calling Stored Procedures using LINQ: if we want to call any SP of Sql Server DB using LINQ we need to first drag and drop the SP on RHS panel of OR-designer, so that it gets converted into a method under "ASPDBDataContext" class with same name of the SP. If the SP has any parameters those parameters will be defined for the method also, where input parameters of procedure become input parameters and output parameters of procedure becomes ref parameters of the method. For example, if the below SP was dropped on RHS panel of OR-designer:

Create Procedure Add(@x int, @y int, @z int out)

The method gets defined as following: **public int Add(int? x, int? y, ref int? z)**

If the SP contains any non-query operations in it, in such cases the return type of method will be int, where as if the SP has any select statements in it that returns table results then the return type of the method will be ISingleResult<T>, where T represents a class that is newly defined when we drag and drop the select SP whose name will be SP Name suffixed with "Result" i.e. for example if the procedure name is Customer_Select then the class name will be Customer_SelectResult.

To call Stored Procedures using LINQ first drag and drop all our 4 Stored Procedures we defined earlier "Customer_Select", "Customer_Insert", "Customer_Update" and "Customer_Delete" on the RHS panel of the OR-

Designer from Server Explorer so that all the methods gets generated under “ASPDBDataContext” class. Now add a new Web Form in the project naming it as “CRUDWithSP.aspx.cs” and write the below code under its “ ” tag:

```
<asp:ListView ID="ListView1" runat="server" DataKeyNames="Custid">
  <LayoutTemplate>
    <table border="1" align="center">
      <tr> <th>Custid</th> <th>Name</th> <th>Balance</th> <th>City</th> <th>Status</th> <th>Action</th> </tr>
      <asp:PlaceHolder ID="ItemPlaceHolder" runat="server" />
    </table>
  </LayoutTemplate>
  <ItemTemplate>
    <tr>
      <td><%# Eval("Custid") %></td>                                    <td><%# Eval("Name") %></td>
      <td><%# Eval("Balance") %></td>                                   <td><%# Eval("City") %></td>
      <td align="center">
        <asp:CheckBox ID="cbStatus" runat="server" Enabled="false" Checked='<%# Eval("Status") %>' /></td>
      <td align="center">
        <asp:LinkButton ID="btnEdit" runat="server" Text="Edit" CommandName="Edit" />
        <asp:LinkButton ID="btnDelete" runat="server" Text="Delete" CommandName="Delete"
                    OnClientClick="return confirm('Are you sure of deleting the record?')" />
      </td>
    </tr>
  </ItemTemplate>
  <InsertItemTemplate>
    <tr style="background-color:aquamarine">
      <td></td>
      <td><asp:TextBox ID="txtName" runat="server" Width="150" /></td>
      <td><asp:TextBox ID="txtBalance" runat="server" Width="150" /></td>
      <td><asp:TextBox ID="txtCity" runat="server" Width="150" /></td>
      <td align="center"><asp:CheckBox ID="cbStatus" runat="server" /></td>
      <td align="center">
        <asp:LinkButton ID="btnInsert" runat="server" Text="Insert" CommandName="Insert" /></td>
    </tr>
  </InsertItemTemplate>
  <EditItemTemplate>
    <tr style="background-color:burlywood">
      <td><%# Eval("Custid")%></td>
      <td><asp:TextBox ID="txtName" runat="server" Width="150" Text='<%# Eval("Name")%>' /></td>
      <td><asp:TextBox ID="txtBalance" runat="server" Width="150" Text='<%# Eval("Balance")%>' /></td>
      <td><asp:TextBox ID="txtCity" runat="server" Width="150" Text='<%# Eval("City")%>' /></td>
      <td align="center">
        <asp:CheckBox ID="cbStatus" runat="server" Enabled="false" Checked='<%# Eval("Status")%>' /></td>
      <td align="center">
        <asp:LinkButton ID="btnUpdate" runat="server" Text="Update" CommandName="Update" />
        <asp:LinkButton ID="btnCancel" runat="server" Text="Cancel" CommandName="Cancel" />
      </td>
```

```
    </tr>
  </EditItemTemplate>
</asp:ListView>
```

Go to design view of Web Form, double click on ItemEditing, ItemCanceling, ItemInserting, ItemUpdating, and ItemDeleting events of ListView control and write the below code in "CRUDWithSP.aspx.cs" file:

Declarations:

```
ASPDBDataContext dc = new ASPDBDataContext();
```

Code Under Page Load Event:

```
if(!IsPostBack) {
  LoadData();
}
```

```
private void LoadData() {
  ListView1.DataSource = dc.Customer_Select(null, true);
  ListView1.DataBind();
}
```

Code under ListView ItemInserting Event:

```
string Name = ((TextBox)e.Item.FindControl("txtName")).Text;
decimal Balance = decimal.Parse(((TextBox)e.Item.FindControl("txtBalance")).Text);
string City = ((TextBox)e.Item.FindControl("txtCity")).Text;
bool Status = ((CheckBox)e.Item.FindControl("cbStatus")).Checked;
int? Custid = null;
dc.Customer_Insert(Name, Balance, City, Status, ref Custid);
if(Custid != null) {
  LoadData();
}
else {
  Response.Write("<script>alert('Failed inserting record into the table.')</script>");
}
```

Code under ListView ItemEditing Event:

```
ListView1.EditIndex = e.NewEditIndex;
LoadData();
```

Code under ListView ItemCanceling Event:

```
ListView1.EditIndex = -1;
LoadData();
```

Code under ListView ItemUpdating Event:

```
int Custid = (int)e.Keys["Custid"];
string Name = ((TextBox)ListView1.Items[e.ItemIndex].FindControl("txtName")).Text;
decimal Balance = decimal.Parse(((TextBox)ListView1.Items[e.ItemIndex].FindControl("txtBalance")).Text);
string City = ((TextBox)ListView1.Items[e.ItemIndex].FindControl("txtCity")).Text;
if(dc.Customer_Update(Custid, Name, Balance, City) == 0) {
  ListView1.EditIndex = -1;
  LoadData();
}
else {
```

```
  Response.Write("<script>alert('Failed updating record in the table.')</script>");
}
```

Code under ListView ItemDeleting Event:

```
int Custid = (int)e.Keys["Custid"];
if (dc.Customer_Delete(Custid) == 0) {
  LoadData();
}
else {
  Response.Write("<script>alert('Failed deleting record from the table.')</script>");
}
```

Implementing Complex Queries using LINQ: to try implementing Complex Queries using LINQ, first drag and drop "Department" and "Employee" tables from our "ASPDB" database on the LHS panel of "OR Designer". Add a new Web Form under the project naming it as "ComplexQueries.aspx" and place a "GridView" control on it. Now go to "ComplexQueries.aspx.cs" file and write the below code:

Code under Page Load Event:

```
ASPDBDataContext dc = new ASPDBDataContext();
//Implementing Queries on Customer Table:
//var tab = from C in dc.Customers select C;
//var tab = from C in dc.Customers orderby C.Balance select C;
//var tab = from C in dc.Customers orderby C.Name descending select C;
//var tab = from C in dc.Customers select new { C.Custid, C.Name, IsActive = C.Status };
//var tab = from C in dc.Customers where C.City == "Hyderabad" select C;
//var tab = from C in dc.Customers where C.Balance > 20000 select C;
//var tab = from C in dc.Customers where C.City == "Hyderabad" && C.Balance > 10000 select C;
//var tab = from C in dc.Customers where C.City == "Hyderabad" || C.Balance > 20000 select C;
//var tab = from C in dc.Customers group C by C.City into G select new { City = G.Key, CustCount = G.Count() };
//var tab = from C in dc.Customers group C by C.City into G where G.Count() >=5 select new { City = G.Key,
CustCount = G.Count() };
//var tab = from C in dc.Customers group C by C.City into G where G.Count() >= 5 orderby G.Key descending select
new { City = G.Key, CustCount = G.Count() };
//var tab = from C in dc.Customers group C by C.City into G select new { City = G.Key, MaxBalance = G.Max(C =>
C.Balance) };
//var tab = from C in dc.Customers group C by C.City into G select new { City = G.Key, MinBalance = G.Min(C =>
C.Balance) };
//var tab = from C in dc.Customers group C by C.City into G select new { City = G.Key, TotalBalance = G.Sum(C =>
C.Balance) };
//var tab = from C in dc.Customers group C by C.City into G select new { City = G.Key, AvgBalance = G.Average(C =>
C.Balance) };

//Implementing Queries on Employee Table:
//var tab = from E in dc.Employees group E by E.Job into G select new { Job = G.Key, EmpCount = G.Count() };
//var tab = from E in dc.Employees group E by E.Did into G select new { Did=G.Key,TotalSal=G.Sum(E => E.Salary) };
//var tab = from E in dc.Employees where E.Job == "Clerk" group E by E.Did into G where G.Count() > 1 orderby
G.Key descending select new { Did = G.Key, ClerkCount = G.Count() };
```

```
//Implementing a Join Query between Employee and Department Tables:
//var tab = from E in dc.Employees join D in dc.Departments on E.Did equals D.Did select new { E.Eid, E.Ename,
E.Job, E.Salary, E.Did, D.Dname, D.Location };

//Implementing a Left Outer Join Query between Employee and Department Tables:
//var tab = from E in dc.Employees join D in dc.Departments on E.Did equals D.Did into Dept from Department in
Dept.DefaultIfEmpty() select new { E.Eid, E.Ename, E.Job, E.Salary, Department.Dname, Department.Location };

//Implementing a Right Outer Join Query between Employee and Department Tables:
//var tab = from D in dc.Departments join E in dc.Employees on D.Did equals E.Did into Emp from Employee in
Emp.DefaultIfEmpty() select new { Employee.Ename, Employee.Job, Employee.Salary, D.Did, D.Dname, D.Location
};

//Implementing a Full Outer Join Query between Employee and Department Tables:
var LeftOuter = from E in dc.Employees join D in dc.Departments on E.Did equals D.Did into Dept from Department
in Dept.DefaultIfEmpty() select new { E.Ename, E.Job, E.Salary, Department.Dname, Department.Location };
var RightOuter = from D in dc.Departments join E in dc.Employees on D.Did equals E.Did into Emp from Employee
in Emp.DefaultIfEmpty() select new { Employee.Ename, Employee.Job, Employee.Salary, D.Dname, D.Location };
var tab = LeftOuter.Union(RightOuter);

GridView1.DataSource = tab;
GridView1.DataBind();
```

LINQDataSource Control: Just like we have SqlDataSource control, we have LinqDataSource control also using which we can perform CRUD operations using LINQ without writing any code manually.

To test this add a new Web Form under the project naming it as “LinqDS.aspx” and place a GridView control on it and also add a LinqDataSource control, go to its properties => change the ID as “EmployeeDS” and click on the top right corner button, select “Configure Data Source” option which open a window in that select our “LinqToSqlProject.ASPDBDataContext” and click Next button, now under the tables select “Employees Table<Employee>” and click “Finish” button and select the CheckBox’s “Enable Delete”, “Enable Insert” and “Enable Update” CheckBoxs.

Now click on the top right corner button of GridView control we placed earlier and under Choose DataSource option select “EmployeeDS” and select the CheckBoxs “Enable Paging”, “Enable Sorting”, “Enable Editing” and “Enable Deleting”, and run the WebForm which displays the data of Employee table.

In the above case when we click on the Edit Button of a record it provides the TextBox’s for editing that record but in case of Did column the value we want to change must be a value that is present under the Department table, so without a normal TextBox to edit, it will be better if a DropDownList control comes there displaying all the Department Names present in Department table so that user can choose a value from the list to update. To do this add one more LinqDataSource control, change its Id as “DepartmentDS”, click on the top right corner of it and select our “LinqToSqlProject.ASPDBDataContext” and click Next button, now under the tables select “Departments Table<Department>” and click finish. Now go to source view and under the GridView columns delete the “Did” Bound Field and write the following code in that place:

```
<asp:TemplateField HeaderText="Did" SortExpression="Salary">
```

```
  <ItemTemplate><%# Eval("Did") %></ItemTemplate>
  <EditItemTemplate>
    <asp:DropDownList ID="ddlDepts" runat="server" DataSourceID="DeptDS" DataTextField="Dname"
                      DataValueField="Did" SelectedValue='<%# Bind("Did") %>' />
  </EditItemTemplate>
</asp:TemplateField>
```

Now run the Web Form again and now we see a DropDownList in Edit Mode under "Did" column displaying Department Names to choose.

Authentication and Authorization

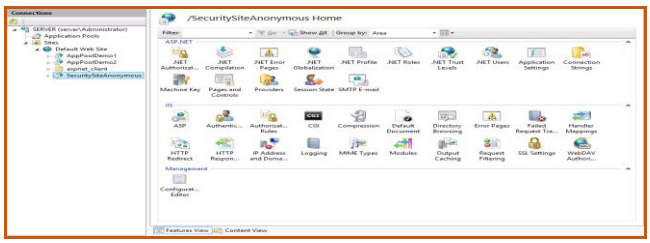
Authentication is the process of obtaining identification credentials such as username and password from a user and validating those credentials against some authority. If the credentials are valid, the user that submitted the credentials is considered as authenticated user. Once the user has been authenticated, the authorization process determines whether that user has access to a resource or not.

ASP.Net supports different types of authentications like:

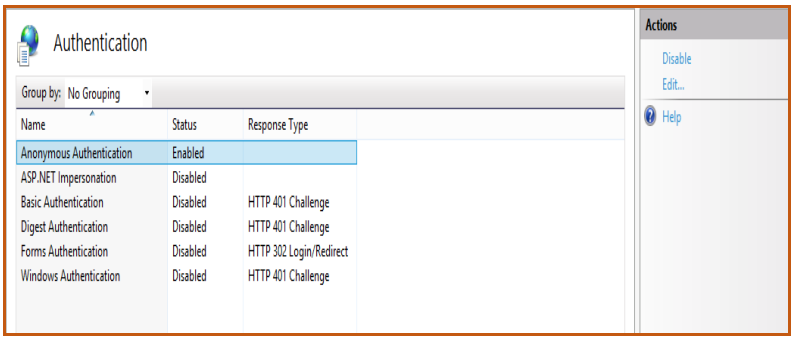
1. Anonymous Authentication
2. Windows Authentication
3. Forms Authentication
4. Open Authentication

Anonymous Authentication: this gives users access to the public areas of our Web Site/Web Application or Web Service without asking them for a username or password. By default, the IUSR account, which was introduced in IIS 7.0 and replaces the IIS 6.0 IUSR_ComputerName account, is used to allow anonymous access. By default, IIS uses Anonymous authentication for every Web Site, Web Application, or Web Service.

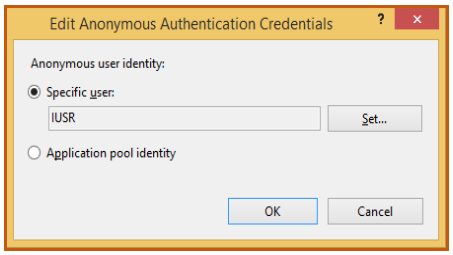
To test anonymous authentication, create a new ASP.NET Web Application project naming it as "SecuritySiteAnonymous", host the project on "IIS" by creating a "Virtual Directory". Open "IIS Manager" and here we find our application under "Default Web Site" as following:



Now when we select the site, on the right side in the "Features View" we find "Authentication" option double click on it which displays Authentication details over there and we find Anonymous Authentication enabled and the other authentication modes disabled as following:



Now in the right-hand side we find "Actions Panel" displaying "Edit…" option and when we click on it a window called "Edit Anonymous Authentication Credentials" get opened and under it we can find the user account IIS is using for anonymous access i.e., IUSR as following:



Now add a Web Form under the project naming it as "CheckAuthentication.aspx" and write the following code in "CheckAuthentication.aspx.cs" file under "Page Load" event:

```
Response.Write("User Name: " + User.Identity.Name + "<br />");
Response.Write("Is Authenticated: " + User.Identity.IsAuthenticated + "<br />");
Response.Write("Authentication Type: " + User.Identity.AuthenticationType + "<br />");
```

When we run the WebForm it will not display "Authentication Type" and "Username" in case of "Anonymous Authentication" and "Is Authenticated" value will be "false". We need to disable "Anonymous Authentication" for our site in case we want to enable any other Authentication Mode and to disable "Anonymous Authentication", in IIS Manager under "Authentication Panel" right click on "Anonymous Authentication" and select "Disable". Now if we run our Web Form again, we get "HTTP Error 401.2 – Unauthorized" error.

Even if "Anonymous Authentication" is enabled in IIS, it is still possible to restrict access to pages of our application by setting the "Authorization" header in "Web.config" file of the Project by writing the below code:

```
<authorization>
  <deny users="?"/>                    ("?" indicates anonymous users)
</authorization>
```

To test the above enable "Anonymous Authentication" again under "IIS Manager", come back to our "SecuritySiteAnonymous" Web Application Project and write the above code in "Web.config" file under "<system.web>" tag and run the Web Form again which displays a Server Error "Access is denied".

Windows Authentication: we use Windows authentication when our IIS server runs on a corporate network that is using Microsoft Active Directory service domain identities or other Windows accounts to identify users. Windows

authentication is a secure form of authentication because the username and password are hashed (encrypted) before being sent across the network. When you enable Windows authentication, the client browser sends a strongly hashed version of the password in a cryptographic exchange to the Web Server. This authentication mode is best suitable for Intranet Applications.

To test anonymous authentication, create a new ASP.NET Web Application project naming it as "SecuritySiteWindows", host the project on "IIS Web Server" by creating a "Virtual Directory". Now go to IIS Manager, select "SecuritySiteWindows", double click on Authentication option on the right side, "Disable" "Anonymous Authentication" and "Enable" "Windows Authentication".

Now add a new Web Form naming it as "CheckAuthentication.aspx" and write the following code in "CheckAuthentication.aspx.cs" file under "Page Load" event:

```
Response.Write("User Name: " + User.Identity.Name + "<br />");
Response.Write("Is Authenticated: " + User.Identity.IsAuthenticated + "<br />");
Response.Write("Authentication Type: " + User.Identity.AuthenticationType + "<br />");
```

Now when we run the Web Form it displays "Authentication Type: Negotiate" in case of "Windows Authentication", "User Name: MachineName\UserName (Server\Administrator)" (where Server is my machine name and Administrator is the name of the user I am logged in) and "Is Authenticated: True".

In this authentication mode we can access the application by using any windows user account that is configured and we can try this by logging in to the computer by using any other user account.

Note: if no other user account is available on our machine, we can create a new user account, and to do that, search for "Computer Management" in Window Search and launch it. In the window opened we find "Local Users and Groups" in LHS, expand it, and select "Users" which display users on our computer. To create a new user account right click on "Users" and select "New User" option which opens a window and in that enter "Username", "Password" and "Confirm Password" values and create a new user with the name "TestUser". Now using the switch user option, log in in the machine by using the new user account, open browser and enter the following URL to open the web page: "http://localhost/SecuritySiteWindows/CheckAuthentication.aspx".

It is possible to provide access to specific users only by specifying those users list in "Web.config" file under "<system.web>" tag as following:

```
<authorization>
  <allow users="Server\TestUser"/>        =>    ("Server" is my machine name and "TestUser" is new user.)
  <deny users="*"/>                       =>    ("*" indicates all other authenticated users)
</authorization>
```

Note: Now run the Web Form again and watch the difference because now it provides access only to "TestUser" but not to any other user.

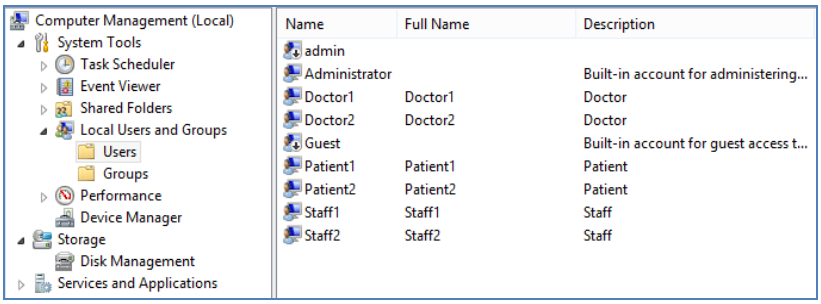
User Groups: under "Computer Management" apart from Users we also find "Groups" and to check that open "Computer Management" window, expand "Local Users and Groups" option which displays Users and Groups also, when we select Users, it displays existing users and when we select Groups it displays existing groups. Every group is a collection of users, and every user must be members of some group, to check this, select users, double click on

any username which displays a window "Administrator Properties" and in that we find a tab "Member Of" click on it which displays the list of groups to which this user belongs to.

It is possible to provide access to a Web Application for users based on their roles (groups) also and to do that first create a new group. To create a new group right click on "Groups" and select "New Group" option which opens a window and in that enter "Group name" as "TestGroup", click on "Add" button which opens a new window and in that window under the last TextBox enter "TestUser", click on "Check Names" button, click "Ok" and click "Create" button to create the new Group. Now we can specify the "Group Name" in "Web.Config" file so that all users under that group will be getting access to the site and to that we need to write following code in "Web.config" file, in place of our previous code:

```
<authorization>
  <allow roles="TestGroup" />
  <deny users="*" />
</authorization>
```

Folder Level Authorization: it is possible to provide access to specific pages of a Web Application to Users or Groups with the help of Folder Level Authorization and to do this let us create a Hospital Management Site which contains 3 types of users: Doctor, Staff and Patient, and access should be given to only the pages that are required for a user or group. To test this first open Computer Management window and create 6 users with the names "Doctor1, Doctor2, Staff1, Staff2, Patient1 and Patient2" which should look as following:



Now create 3 new groups naming them as "DoctorGroup", "StaffGroup" and "PatientGroup", and to do this right click on Groups, select "New Group" which opens a Window, enter group name in it as "DoctorGroup", click on "Add" button which opens a window in that under the last TextBox enter "Doctor1" click "Check Names" button which displays "ComputerName\Doctor1" and repeat the same process to add "Doctor2" also to this group. Now follow the same process to create "StaffGroup" and "PatientGroup" also and now under the Groups we find our 3 new groups.

Now create a new "ASP.Net Web Application" project naming it as "HospitalMgmt" and host it on "IIS Web Server" by creating a Virtual Directory. Open "IIS Manager" select "HospitalMgmt" site, and double click on Authentication option on RHS, disable "Anonymous Authentication" and enable "Windows Authentication".

Now open Solution Explorer, add 3 new folders under the project naming them as "Doctor", "Staff" and "Patient", and now under all the 3 folders add a "Web.config" file and write the following code inside "<system.web>" tag:

Doctor	Staff	Patient

<authorization> <allow roles="DoctorGroup"/> <deny users="*"/> </authorization>	<authorization> <allow roles="StaffGroup"/> <deny users="*"/> </authorization>	<authorization> <allow roles="PatientGroup"/> <deny users="*"/> </authorization>

Now add one Web Form in each folder naming it as "DoctorHomePage.aspx", "StaffHomePage.aspx" and "PatientHomePage.aspx" and write the following code under the " " tag of each page:

DoctorHomePage.aspx:

```
<h1 style="text-align: center; background-color: yellowgreen; color: orangered">Appollo Hospitals Ltd.</h1>
<h2>Welcome to Doctor Page</h2>
Back to <asp:HyperLink ID="hlBack" runat="server" Text="Home" NavigateUrl="~/SiteHomePage.aspx" /> Page
```

StaffHomePage.aspx:

```
<h1 style="text-align: center; background-color: yellowgreen; color: orangered">Appollo Hospitals Ltd.</h1>
<h2>Welcome to Staff Page</h2>
Back to <asp:HyperLink ID="hlBack" runat="server" Text="Home" NavigateUrl="~/SiteHomePage.aspx" /> Page
```

PatientHomePage.aspx:

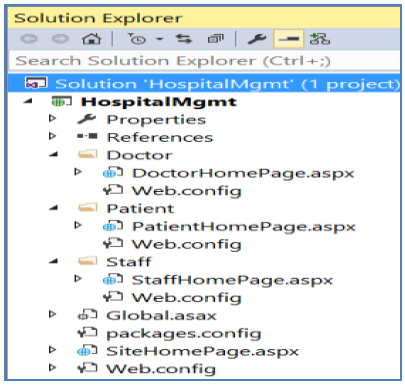
```
<h1 style="text-align: center; background-color: yellowgreen; color: orangered">Appollo Hospitals Ltd.</h1>
<h2>Welcome to Patient Page</h2>
Back to <asp:HyperLink ID="hlBack" runat="server" Text="Home" NavigateUrl="~/SiteHomePage.aspx" /> Page
```

Now add a new Web Form under the project with the name "SiteHomePage.aspx" and write the following code under its " " tag:

```
<div style="text-align: center">
  <h1 style="background-color: yellowgreen; color: orangered">Appollo Hospitals Ltd.</h1>
  <h3>Log in as:</h3>
  <table border="1" align="center">
    <tr>
      <td>
        <asp:HyperLink ID="hlDoctor" runat="server" NavigateUrl="~/Doctor/DoctorHomePage.aspx"
                                                    Text="Doctor" />
      </td>
    </tr>
    <tr>
      <td>
        <asp:HyperLink ID="hlStaff" runat="server" NavigateUrl="~/Staff/StaffHomePage.aspx" Text="Staff" />
      </td>
    </tr>
    <tr>
      <td>
        <asp:HyperLink ID="hlPatient" runat="server" NavigateUrl="~/Patient/PatientHomePage.aspx"
                                                    Text="Patient" />
```

```
      </td>
    </tr>
  </table>
</div>
```

Finally, the application should look as following in Solution Explorer:



Forms Authentication: this authentication lets us authenticate users by using our own code and then maintain an authentication token in a cookie or in the page URL. To use forms authentication, we must first create a login page that collects credentials from the user and that includes code to authenticate the credentials. Typically, we configure the application to redirect requests to the login page when users try to access a protected resource, such as a page that requires authentication. If the user's credentials are valid, we can call methods of the "FormsAuthentication" class to redirect the request back to the originally requested resource with an appropriate authentication ticket (cookie), on subsequent requests the user's browser passes the authentication cookie with the request, which then by passes the login page.

We configure forms authentication by using the authentication configuration element. In the simplest case, we have a login page and in the configuration file, we specify a URL to redirect un-authenticated requests to the login page. We then define valid credentials, either in the Web.config file or in a separate file or validate thru a database by our own code. This is best suitable for internet application than intranet application.

To understand the process, first open a new ASP.Net Web Application project naming it as "SecuritySiteForms" and enable it to run under IIS by creating a Virtual Directory. Now add 4 WebForms under the project naming it as "HomePage.aspx", "Page1.aspx", "Page2.aspx" and "LoginPage.aspx", and write the below code under each page:

Code under tag of "HomePage.aspx":

```
<h1 style="text-align: center; background-color: greenyellow; color: orangered">Naresh I Technologies</h1>
<h3>This is Home Page of the site.</h3>
<ul>
  <li><asp:HyperLink ID="hlPage1" runat="server" Text="First Page" NavigateUrl="~/Page1.aspx" /></li>
  <li><asp:HyperLink ID="hlPage2" runat="server" Text="Second Page" NavigateUrl="~/Page2.aspx" /></li>
</ul>
```

Code under tag of "Page1.aspx":

```
<div>
  <h1 style="text-align: center; background-color: greenyellow; color: orangered">Naresh I Technologies</h1>
  <h3>This is First Page of the site.</h3>
```

```
  Back to <asp:HyperLink ID="hlBack" runat="server" Text="Home" NavigateUrl="~/HomePage.aspx" /> Page
</div>
```

Code under tag of "Page2.aspx":

```
<div>
  <h1 style="text-align: center; background-color: greenyellow; color: orangered">Naresh I Technologies</h1>
  <h3>This is Second Page of the site.</h3>
  Back to <asp:HyperLink ID="hlBack" runat="server" Text="Home" NavigateUrl="~/HomePage.aspx" /> Page
</div>
```

Code under tag of "LoginPage.aspx":

```
<h1 style="text-align: center; background-color: greenyellow; color: orangered">Naresh I Technologies</h1>
<table align="center">
  <caption>Login Page</caption>
  <tr>
    <td>User Id:</td>
    <td><asp:TextBox ID="txtName" runat="server" Width="200px" /></td>
  </tr>
  <tr>
    <td>Password:</td>
    <td><asp:TextBox ID="txtPwd" runat="server" Width="200px" TextMode="Password" /></td>
  </tr>
  <tr>
    <td><asp:CheckBox ID="cbRemember" runat="server" Text="Remember Me" /></td>
    <td>
      <asp:Button ID="btnLogin" runat="server" Text="Login" Width="100px" />
      <asp:Button ID="btnReset" runat="server" Text="Reset" Width="100px" />
    </td>
  </tr>
  <tr>
    <td colspan="2"><asp:Label ID="lblStatus" runat="server" ForeColor="Red" /></td>
  </tr>
</table>
```

Right now, we can directly access Homepage or any other page that is present under the site without any restriction but if we want them to be accessed only after supplying the valid credentials, then we need to enable Forms Authentication thru the "Web.config" file, and to do that write the below code under "<system.web>" tag:

```
<authentication mode="Forms">
  <forms loginUrl="LoginPage.aspx">
    <credentials passwordFormat="Clear">
      <user name="Raju" password="raju@1234"/>
      <user name="Admin" password="admin@1234"/>
    </credentials>
  </forms>
</authentication>
```

Note: Password format is to specify whether password should be clear text or encrypted to SHA1 or MD5 formats.

Attributes of <forms> element:

- **loginUrl** is to specify the name of the page which must be launched first when the site is accessed.
- **defaultUrl** is to specify the name of the default page which should be launched after logging in to the site if no other page is requested, if not specified default "defaultUrl" is "default.aspx"
- **name** is to specify name for the login cookie which will be ".ASPXAUTH" if not specified.
- **timeout** is to specify amount of time in minutes after which the cookie expires. The default value is 30.
- **slidingExpiration** is to specify whether sliding expiration is enabled, and default is true. Sliding expiration value if true resets an active authentication cookie's time to expiration upon each request during a single session, whereas if it is false then the cookie expires at a set interval from the time it was originally issued.

After doing the above also we can access any page under the site directly because under IIS Anonymous Authentication is enabled for this site which must be either disabled in IIS or we can do that by adding authorization element under authentication element in "Web.config" file as following:

```
<authorization>
  <deny users="?"/>
</authorization>
```

Now if we try to open any page under the site by default it redirects to "LoginPage.aspx" where we need to supply valid credentials to login, which will then redirect to the requested page, and to achieve it go to "LoginPage.aspx.cs" file and write the following code:

```
using System.Web.Security;
```

Code under Page Load Event:

```
if (!IsPostBack) {
  txtName.Focus();
}
```

Code under Login Button Click Event:

```
if (FormsAuthentication.Authenticate(txtName.Text, txtPwd.Text)) {
  FormsAuthentication.RedirectFromLoginPage(txtName.Text, cbRemember.Checked);
}
else {
  lblStatus.Text = "Login failed, given credentials doesn't match.";
}
```

Code under Reset Button Click Event:

```
txtName.Text = txtPwd.Text = "";
txtName.Focus();
```

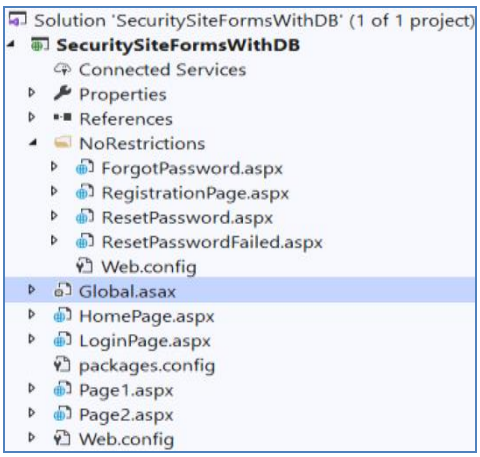
Authenticate is a static method of "FormsAuthentication" class which validates the given username and password against credentials stored in configuration file and returns a boolean value which will be true if the given credentials are valid or else returns false.

RedirectFromLoginPage is a static method of "FormsAuthentication" class which will redirect an authenticated user back to the original requested URL or the default URL. The second parameter of the method is a Boolean value which should be true to create a persistent or durable cookie or else false for in-memory cookie.

Now if we try opening any page, first it will go to "LoginPage" and then after confirming the credentials it will redirect back to the page we requested, but if we directly open "LoginPage", then after entering the valid credentials also we get an error "resource not found (404)" because when we directly go to "LoginPage" then it will redirect us to "default.aspx" page which is not present in our site so to overcome this problem either we need to add "default.aspx" in our site or change this by overriding the "defaultUrl" under "<forms>" tag of "Web.config" file as following:

```
<forms loginUrl="LoginPage.aspx" defaultUrl="HomePage.aspx">
```

Forms Authentication with Database: Create a new "ASP.Net Web Application" project naming it as "SecuritySiteFormsWithDB" and host it on "IIS Web Server" by creating a Virtual Directory. Now add a new Folder under the project naming it as "NoRestrictions" and add 4 Web Forms into the folder naming them as "RegistrationPage.aspx", "ForgotPassword.aspx", "ResetPassword.aspx" and "ResetPasswordFailed.aspx", and add a "Web.config" file also into the folder. Now under the Project add 4 new Web Forms naming them as "LoginPage.aspx", "HomePage.aspx", "Page1.aspx" and "Page2.aspx". Items under Solution Explorer of the project should now look as following:



Write the below code in "Web.config" file that is present in project folder:

Code under "<system.web>" tag:

```
<authentication mode="Forms">
  <forms loginUrl="LoginPage.aspx" defaultUrl="HomePage.aspx" />
</authentication>
<authorization> <deny users="?" /></authorization>
```

Code under "<configuration>" tag:

```
<appSettings>
  <add key="ValidationSettings:UnobtrusiveValidationMode" value="none" />
</appSettings>
```

Write the below code in "Web.config" file we added in "NoRestrictions" folder, under <system.web> tag:

```
<authorization>
  <allow users="?"/>
</authorization>
```

Write the following code under " " tag of "HomePage.aspx":

```
<h1 style="text-align: center; background-color: yellow; color: red">Welcome to Naresh I Technologies</h1>
<table border="1" align="center" style="text-align: center; width: 20%">
  <caption>Click to navigate:</caption>
  <tr>
    <td><asp:HyperLink ID="hlPage1" runat="server" Text="Page1" NavigateUrl="~/Page1.aspx" /></td>
  </tr>
  <tr>
    <td><asp:HyperLink ID="hlPage2" runat="server" Text="Page2" NavigateUrl="~/Page2.aspx" /></td>
  </tr>
</table>
```

Write the following code under " " tag of "Page1.aspx":

```
<h1 style="text-align: center; background-color: greenyellow; color: orangered">Naresh I Technologies</h1>
<h3>This is First Page of the site.</h3>
Back to <asp:HyperLink ID="hlBack" runat="server" Text="Home" NavigateUrl="~/HomePage.aspx" /> Page
```

Code under tag of "Page2.aspx":

```
<h1 style="text-align: center; background-color: greenyellow; color: orangered">Naresh I Technologies</h1>
<h3>This is Second Page of the site.</h3>
Back to <asp:HyperLink ID="hlBack" runat="server" Text="Home" NavigateUrl="~/HomePage.aspx" /> Page
```

Create the following tables under our ASPDB Database:

```
Create Table Users(UserId Varchar(20) Primary Key, Name Varchar(50), Password Varchar(100), Email Varchar(100), IsLocked Bit default 0, FailureCount TinyInt default 0, LockedDate Date)

Create Table ResetPasswordRequest(RequestId UniqueIdentifier Primary Key, UserId Varchar(20) References Users(UserId), RequestDate DateTime, Status TinyInt Not Null default 1)
```

Create the following Stored Procedures under our ASPDB Database:

```
Create Procedure Users_Insert(@UserId Varchar(20), @Name Varchar(50), @Password Varchar(100), @Email Varchar(100), @Count int Out)
As
Begin
  Select @Count = Count(*) From Users Where UserId = @UserId;
  If @Count = 0
    Insert Into Users (UserId, Name, Password, Email) Values (@UserId, @Name, @Password, @Email);
End;

Create Procedure Users_Validate(@UserId Varchar(20), @Password Varchar(100), @Status Int Out)
As
Begin
  Declare @IsLocked Bit, @FailureCount TinyInt;
  If Not Exists(Select * From Users Where UserId=@UserId)
  Begin
    Set @Status = -1;
  End
  Else
  Begin
    Select @IsLocked=IsLocked From Users Where UserId=@UserId;
```

```
    If @IsLocked = 1
    Begin
      Set @Status = -2;
    End;
    Else
    Begin
      If Exists(Select * From Users Where UserId=@UserId And Password=@Password)
      Begin
        Set @Status = 0;
        Update Users Set FailureCount = 0 Where UserId=@UserId;
      End;
      Else
      Begin
        Select @FailureCount=FailureCount From Users Where UserId=@UserId;
        If @FailureCount = 2
        Begin
          Set @Status = -3;
          Update Users Set IsLocked=1, LockedDate=GetDate() Where UserId=@UserId;
        End;
        Else
        Begin
          Set @FailureCount += 1;
          Set @Status = @FailureCount;
          Update Users Set FailureCount=@FailureCount Where UserId=@UserId;
        End;
      End;
    End;
  End;
End;

Create Procedure ResetPasswordRequest_Insert(@UserId Varchar(20), @Name Varchar(50) Out, @Email
Varchar(100) Out, @RequestId UniqueIdentifier Out)
As
Begin
  Set @Name = Null; Set @Email = Null; Set @RequestId = Null;
  If Exists(Select * From Users Where UserId=@UserId And IsLocked=0)
  Begin
    Set @RequestId = NewId();
    Select @Name=Name, @Email=Email From Users Where UserId=@UserId;
    Insert Into ResetPasswordRequest(RequestId, UserId, RequestDate) Values(@RequestId, @UserId, GetDate());
  End;
End;

Create Procedure ResetPasswordLink_IsValid(@RequestId UniqueIdentifier, @Status Int Out)
As
Begin
```

```
  Declare @Minutes Int;
  Select @Status=Status From ResetPasswordRequest Where RequestId=@RequestId;
  If @Status = 1
  Begin
    Select @Minutes = DateDiff(mi, RequestDate, GetDate()) From  ResetPasswordRequest Where RequestId=@RequestId;
    If @Minutes > 15
    Begin
      Set @Status = 0;
      Update ResetPasswordRequest Set Status=0 Where RequestId=@RequestId;
    End;
  End;
End;

Create Procedure Users_ResetPassword(@RequestId UniqueIdentifier, @Password Varchar(100), @Status Int Out)
As
Begin
  Declare @UserId Varchar(20);
  Select @UserId = UserId From ResetPasswordRequest Where RequestId=@RequestId;
  Update Users Set Password=@Password, FailureCount=0 Where UserId=@UserId;
  If @@ROWCOUNT = 1
  Begin
    Update ResetPasswordRequest Set Status=2 Where RequestId=@RequestId;
    Set @Status = 1;
  End;
  Else
  Begin
    Set @Status = 0;
  End;
End;
```

Add an "OR Designer" under the project naming it as "ASPDB.dbml", configure our "ASPDB" Database under Server Explorer of Visual Studio, drag and drop all the above 5 stored procedures on the RHS Panel of the OR Designer to generate the mapping methods.

Write the below code under " " tag of "RegistrationPage.aspx":

```
<h1 style="text-align: center; background-color: greenyellow; color: orangered">Naresh I Technologies</h1>
<table align="center">
  <caption>Registration Form</caption>
  <tr>
    <td>Name:</td>
    <td><asp:TextBox ID="txtName" runat="server" Width="150px" /></td>
    <td><asp:RequiredFieldValidator ID="rfvName" runat="server" ControlToValidate="txtName"
                Display="Dynamic" ErrorMessage="Can't leave the field empty." ForeColor="Red" /></td>
  </tr>
  <tr>
    <td>User Id:</td>
```

```
    <td><asp:TextBox ID="txtId" runat="server" Width="150px" /></td>
    <td><asp:RequiredFieldValidator ID="rfvId" runat="server" ControlToValidate="txtId" Display="Dynamic"
                ErrorMessage="Can't leave the field empty." ForeColor="Red" /></td>
  </tr>
  <tr>
    <td>Password:</td>
    <td><asp:TextBox ID="txtPwd" runat="server" TextMode="Password" Width="150px" /></td>
    <td><asp:RequiredFieldValidator ID="rfvPwd" runat="server" ControlToValidate="txtPwd" Display="Dynamic"
                ErrorMessage="Can't leave the field empty." ForeColor="Red" /></td>
  </tr>
  <tr>
    <td>Confirm Pwd:</td>
    <td><asp:TextBox ID="txtCPwd" runat="server" TextMode="Password" Width="150px" /></td>
    <td><asp:CompareValidator ID="cvCPwd" runat="server" ControlToCompare="txtPwd" Display="Dynamic"
                ControlToValidate="txtCPwd" ErrorMessage="Confirm password should match with pwd."
                ForeColor="Red" /></td>
  </tr>
  <tr>
    <td>Email Id:</td>
    <td><asp:TextBox ID="txtEmail" runat="server" TextMode="Email" Width="150px" /></td>
    <td><asp:RequiredFieldValidator ID="rfvEmail" runat="server" ControlToValidate="txtEmail"
                Display="Dynamic" ErrorMessage="Can't leave the field empty." ForeColor="Red" /></td>
  </tr>
  <tr>
    <td align="center" colspan="2">
      <asp:Button ID="btnRegister" runat="server" Text="Register" />
      <asp:Button ID="btnReset" runat="server" Text="Reset" />
    </td>
    <td> </td>
  </tr>
  <tr>
    <td colspan="3">
      <asp:Label ID="lblStatus" runat="server" ForeColor="Red" />
    </td>
  </tr>
</table>
```

Write the following code under "RegistrationPage.aspx.cs" file:

```
using System.Web.Security;
```

Code under Page Load Event:

```
if (!IsPostBack) {
  txtName.Focus();
}
```

Code under Register Button Click Event:

```
ASPDBDataContext dc = new ASPDBDataContext();
```

```
string EncPwd = FormsAuthentication.HashPasswordForStoringInConfigFile(txtPwd.Text, "MD5");
int? Count = null;
dc.Users_Insert(txtId.Text, txtName.Text, EncPwd, txtEmail.Text, ref Count);
if (Count > 0) {
  lblStatus.Text = "A user already exists with the given user id.";
}
else {
  Response.Redirect("~/LoginPage.aspx");
}
```

Code under Reset Button Click Event:

```
txtId.Text = txtName.Text = txtPwd.Text = txtCPwd.Text = txtEmail.Text = "";
txtId.Focus();
```

Write the below code under " " tag of "LoginPage.aspx":

```
<h1 style="text-align: center; background-color: greenyellow; color: orangered">Naresh I Technologies</h1>
<table align="center">
  <caption>Login Form</caption>
  <tr>
    <td align="right">User Id:</td>
    <td><asp:TextBox ID="txtUserId" runat="server" Width="150px" /></td>
  </tr>
  <tr>
    <td align="right">Password:</td>
    <td><asp:TextBox ID="txtPwd" runat="server" TextMode="Password" Width="150px" /></td>
  </tr>
  <tr>
    <td><asp:CheckBox ID="cbRemember" runat="server" Text="Remember Me" /></td>
    <td>
      <asp:Button ID="btnLogin" runat="server" Text="Login" Width="75px" />
      <asp:Button ID="btnReset" runat="server" Text="Reset" Width="75px" />
    </td>
  </tr>
  <tr>
    <td colspan="2">New user <asp:HyperLink ID="hlRegister" runat="server"
          NavigateUrl="~/NoRestrictions/RegistrationPage.aspx">click</asp:HyperLink> to register</td>
  </tr>
  <tr>
    <td colspan="2">Forgot password <asp:HyperLink ID="HyperLink1" runat="server"
          NavigateUrl="~/NoRestrictions/ForgotPassword.aspx">click</asp:HyperLink> to reset</td>
  </tr>
  <tr>
    <td colspan="2"><asp:Label ID="lblStatus" runat="server" ForeColor="Red"></asp:Label></td>
  </tr>
</table>
```

Write the below code under "LoginPage.aspx.cs" file:

```
using System.Web.Security;
```

Code under Page Load Event:

```
if (!IsPostBack) {
  txtUserId.Focus();
}
```

Code under Login Button Click Event:

```
ASPDBDataContext dc = new ASPDBDataContext();
string EncPwd = FormsAuthentication.HashPasswordForStoringInConfigFile(txtPwd.Text, "MD5");
int? Status = null;
dc.Users_Validate(txtUserId.Text, EncPwd, ref Status);
if (Status == -3) {
  lblStatus.Text = "You failed all the 3 attempts to login and your account is locked.";
}
else if (Status == -2) {
  lblStatus.Text = "Your account is in locked state, contact customer support.";
}
else if (Status == -1) {
  lblStatus.Text = "No user exists with the given User Id, recheck and try again.";
}
else if (Status == 0) {
  FormsAuthentication.RedirectFromLoginPage(txtUserId.Text, cbRemember.Checked);
}
else {
  lblStatus.Text = Status + " attempt(s) failed, maximum is 3 only.";
}
```

Code under Reset Button Click Event:

```
txtUserId.Text = txtPwd.Text = lblStatus.Text = "";
txtUserId.Focus();
```

Write the below code under " " tag of "ForgotPassword.aspx":

```
<h1 style="text-align: center; background-color: greenyellow; color: orangered">Naresh I Technologies</h1>
<table align="center">
  <caption>Forgot Your Password?</caption>
  <tr><td>Enter your User Id to receive a reset password link.</td></tr>
  <tr>
    <td>Enter User Id:
      <asp:TextBox ID="txtUserId" runat="server" Width="150px" />
      <asp:RequiredFieldValidator ID="rfvUserId" runat="server" ControlToValidate="txtUserId" Display="Dynamic"
                  ErrorMessage="Can't leave the field empty." ForeColor="Red" />
    </td>
  </tr>
  <tr><td align="center"><asp:Button ID="btnSubmit" runat="server" Text="Submit" /></td></tr>
  <tr><td><asp:Label ID="lblStatus" runat="server" ForeColor="Red" /></td></tr>
</table>
```

Write the below code under "ForgotPassword.aspx.cs" file:

```
using System.Net;
using System.Text;
using System.Net.Mail;
```

Code under Page_Load Event:

```
if(!IsPostBack) {
  txtUserId.Focus();
}
```

Code under Submit Button Click Event:

```
ASPDBDataContext dc = new ASPDBDataContext();
Guid? RequestId = null;
string Name = null, Email = null;
dc.ResetPasswordRequest_Insert(txtUserId.Text, ref Name, ref Email, ref RequestId);
if (Name == null || Email == null || RequestId == null) {
  lblStatus.Text = "Could not send a reset password mail, contact customer support.";
}
else {
  SendMail(Name, Email, RequestId.ToString());
  lblStatus.Text = "Reset password request mail is sent to your registered email.";
}
```

```
private void SendMail(string Name, string Email, string RequestId)
{
  StringBuilder mailBody = new StringBuilder();
  mailBody.Append("Hello " + Name + "<br /><br />");
  mailBody.Append("Click on the link below to reset your password: <br />");

  mailBody.Append(
    "http://localhost/SecuritySiteFormsWithDB/NoRestrictions/ResetPassword.aspx?RequestId=" + RequestId);
  mailBody.Append("<br /><br />");
  mailBody.Append("<font color='red'>Note: Reset password link is valid only for 15 minutes.</font>");
  mailBody.Append("<br /><br />");
  mailBody.Append("Regards");
  mailBody.Append("<br /><br />");
  mailBody.Append("Customer Support.");

  MailMessage MailMsg = new MailMessage("Nithelpcenter1234@gmail.com", Email);
  MailMsg.IsBodyHtml = true;
  MailMsg.Subject = "Reset Password Link";
  MailMsg.Body = mailBody.ToString();

  SmtpClient MailSender = new SmtpClient("smtp.gmail.com", 587);
  MailSender.Credentials = new NetworkCredential("Nithelpcenter1234@gmail.com", "Hello123$$");
  MailSender.EnableSsl = true;
  MailSender.Send(MailMsg);
}
```

Write the below code under " " tag of "ResetPassword.aspx":

```
<h1 style="text-align: center; background-color: greenyellow; color: orangered">Naresh I Technologies</h1>
<table align="center">
  <caption>Reset Password</caption>
  <tr>
    <td align="right">Enter New Password:</td>
    <td><asp:TextBox ID="txtPwd" runat="server" TextMode="Password" Width="150px" /></td>
    <td><asp:RequiredFieldValidator ID="rfvPwd" runat="server" ControlToValidate="txtPwd" Display="Dynamic"
        ErrorMessage="Can't leave the field empty." ForeColor="Red" /></td>
  </tr>
  <tr>
    <td>Confirm New Password:</td>
    <td><asp:TextBox ID="txtCPwd" runat="server" TextMode="Password" Width="150px" /></td>
    <td><asp:CompareValidator ID="cvCPwd" runat="server" Display="Dynamic" ControlToCompare="txtPwd"
        ControlToValidate="txtCPwd" ErrorMessage="Should match with password." ForeColor="Red" /></td>
  </tr>
  <tr><td colspan="3" align="center"><asp:Button ID="btnSave" runat="server" Text="Save" /></td></tr>
  <tr><td colspan="3"><asp:Label ID="lblStatus" runat="server" ForeColor="Red" /></td></tr>
</table>
```

Write the below code under "ResetPassword.aspx.cs" file:

```
using System.Web.Security;
```

Declarations:

```
ASPDBDataContext dc = new ASPDBDataContext();
```

Code under Page Load Event:

```
string RequestId = Request.QueryString["RequestId"];
int? Status = null;
dc.ResetPasswordLink_IsValid(Guid.Parse(RequestId), ref Status);
if (Status == 0 || Status == 2) {
  Response.Redirect("ResetPasswordFailed.aspx?Status=" + Status);
}
```

Code under Save Button Click Event:

```
string RequestId = Request.QueryString["RequestId"];
string EncPwd = FormsAuthentication.HashPasswordForStoringInConfigFile(txtPwd.Text, "MD5");
int? Status = null;
dc.Users_ResetPassword(Guid.Parse(RequestId), EncPwd, ref Status);
if (Status == 1)
  lblStatus.Text = "Your password has been reset successfully.";
else
  lblStatus.Text = "Failed reseting the password, please contact customer support.";
```

Write the below code under Page Load Event of "ResetPasswordFailed.aspx.cs" file:

```
int Status = int.Parse(Request.QueryString["Status"]);
if (Status == 0)
  Response.Write("<h3>The reset password link is expired.");
else if(Status == 2)
  Response.Write("<h3>The reset password link is already consumed.");
```

Caching

One of the most important factors in building high-performance, scalable Web applications is the ability to store items, whether data objects or pages, or parts of a page, in memory the initial time they are requested. You can cache, or store, these items on the Web Server or other software in the request stream, such as the Proxy Server or Browser. This allows you to avoid recreating information that satisfied a previous request, particularly information that demands significant processor time or other resources. ASP.NET caching allows you to use a technique to store page output or application data across HTTP Requests and reuse it.

An application can often increase performance by storing data in memory that is accessed frequently and that requires significant processing time to create. For example, if your application processes large amounts of data using complex logic and then returns the data as a report accessed frequently by users, it is efficient to avoid re-creating the report every time that a user requests it. Similarly, if your application includes a page that processes complex data but that is updated only in-frequently, it is in-efficient for the server to re-create that page on every request. To help us increase application performance in these situations, ASP.NET provides caching which is divided into 2 categories like:

1. Output Caching
2. Data Caching

Output Caching: Output caching allows you to store dynamic page and user control responses on any HTTP cache-capable device in the output stream, from the originating server to the requesting browser. On subsequent requests, the page or user control code is not executed; the cached output is used to satisfy the request. To enable Output Caching we need to use "OutputCache Directive" either in a WebForm or a WebUserControl as following:

```
<%@ OutputCache Location="None/Any/Client/Server/Downstream" Duration="<time in seconds>"
    VaryByParam="<none/value>" %>
```

- Location is to specify where the cache must be stored, and the default is Any.
- Duration is to specify the time output should be stored in cache memory, in seconds.
- VaryByParam is to specify the parameter based on which the cached output should vary.

Create a new "ASP.Net Web Application" project naming it as "CachingSite". Open "Web.config" file and write the "Connection String" under "<configuration>" tag for connecting to SQL Server Database:

```
<connectionStrings>
  <add name="ConStr" providerName="System.Data.SqlClient"
                        connectionString="User Id=Sa;Password=123;Database=ASPDB;Data Source=Server"  />
</connectionStrings>
```

Add a new class in the project naming it as "ReadCS.cs" and write the below code in it by importing the namespace "System.Configuration".

```
public class ReadCS {
  public static string ASPDB {
    get { return ConfigurationManager.ConnectionStrings["ConStr"].ConnectionString; }
  }
}
```

Add a Web Form in the Project naming it as "OutputCache.aspx" and write the below code under " " tag:

```
<asp:GridView ID="GridView1" runat="server"></asp:GridView>
Client Machine Date & Time: <script>document.write(Date());</script>
```

Write the below code under "OutputCache.aspx.cs" file:

```
using System.Data;
using System.Data.SqlClient;
```

Code under Page Load Event:

```
SqlDataAdapter da = new SqlDataAdapter("Select * from Customer Where Status=1", ReadCS.ASPDB);
DataSet ds = new DataSet();
da.Fill(ds, "Customer");
GridView1.DataSource = ds;
GridView1.DataBind();
Response.Write("Server Machine Date & Time: " + DateTime.Now.ToString());
```

Now if you run the Web Form and watch the output the Server Time value and Browser Time value will be same. Now refresh the page and watch then also both those values will be same, whereas if we enable caching then the Server Time and Browser Time values will be different, to test that go to "OutputCache.aspx" file and write the following code below the Page Directive:

```
<%@ OutputCache Location="Server" Duration="30" VaryByParam="none" %>
```

VaryByParam attributeof Caching: add a new Web Form under the project naming it as "MasterDetail.aspx" and write the below code under its " " tag:

```
<table>
  <tr><td align="center"><asp:DropDownList ID="ddlDept" runat="server" AutoPostBack="true" /></td></tr>
  <tr><td><asp:GridView ID="gvEmp" runat="server" /></td></tr>
</table>
Client Machine Date & Time:<script>document.write(Date());</script>
```

Write the below code under "MasterDetail.aspx.cs" file:

```
using System.Data;
using System.Data.SqlClient;
```

Declarations:

```
DataSet ds;
SqlDataAdapter da;
SqlConnection con;
```

Code under Page Load Event:

```
con = new SqlConnection(ReadCS.ASPDB);
if (!IsPostBack) {
  da = new SqlDataAdapter("Select * From Department", con);
  ds = new DataSet();
  da.Fill(ds, "Department");

  da.SelectCommand.CommandText = "Select * From Employee";
  da.Fill(ds, "Employee");
  ddlDept.DataSource = ds.Tables["Department"];
  ddlDept.DataTextField = "Dname";
  ddlDept.DataValueField = "Did";
  ddlDept.DataBind();
  ddlDept.Items.Insert(0, "All");

  gvEmp.DataSource = ds.Tables["Employee"];
  gvEmp.DataBind();
}
```

Code under DropDownList SelectedIndexChanged Event:

```
if (ddlDept.SelectedIndex == 0) {
  da = new SqlDataAdapter("Select * From Employee", con);
}
else {
  da = new SqlDataAdapter("Select * From Employee Where Did=" + ddlDept.SelectedValue, con);
}
ds = new DataSet();
da.Fill(ds, "Employee");
gvEmp.DataSource = ds.Tables["Employee"];
gvEmp.DataBind();
Response.Write("Data Cached at: " + DateTime.Now.ToLongTimeString());
```

Now turn on caching under the above Web Form by using the OutputCache directive as following:

```
<%@ OutputCache Location="Server" Duration="300" VaryByParam="none" %>
```

Now run the page and select a Department Name from DropDownList control and data gets cached at that point of time, but when we change the selection of Name it will still display the old, cached output only because the cached output duration is 300 seconds, but we require the output to be changing based on the Department Name we select and to achieve that we need to use the "VaryByParam" attribute of "OuputCache" directive which should be as following:

```
<%@ OutputCache Location="Server" Duration="30" VaryByParam="ddlDept" %>
```

Now run the page again and watch the difference in output where we will notice the data being cached based on the value selected under the DropDownList control.

Note: If in any situation if we want the output should be varying based on multiple parameters, we can even specify a semicolon separated list of values under "VaryByParam".

Fragment Caching: it is an approach of caching a part of a webpage which can be achieved through User Control i.e., if at all we want to cache a particular area in a webpage the data we wanted to cache must be first in the form of a User Control with caching enabled and then that control can be placed on the web form, so that the controls output will be cached but not the remaining content of the webpage. To test this, add a "Web Form User Control" in the project naming it as "CustomerCtrl.ascx" and write the below code below the "Control Directive" present over there:

```
<table style="background-color:bisque">
  <tr><td><asp:Label ID="lblControlHeader" runat="server" /></td></tr>
  <tr><td><asp:GridView ID="GridView1" runat="server" /></td></tr>
  <tr><td><asp:Label ID="lblControlFooter" runat="server" /></td></tr>
</table>
```

Write the below code under "CustomerCtrl.ascx.cs" file:

```
using System.Data;
using System.Data.SqlClient;
```

Code under Page Load Event:

```
SqlDataAdapter da = new SqlDataAdapter("Select * From Customer", ReadCS.ASPDB);
DataSet ds = new DataSet();
da.Fill(ds, "Customer");
GridView1.DataSource = ds.Tables[0];
GridView1.DataBind();
lblControlHeader.Text = "Control output cached at: " + DateTime.Now.ToLongTimeString();
lblControlFooter.Text = "Control output cached at: " + DateTime.Now.ToLongTimeString();
```

Now to consume the control add a Web Form under the project naming it as "CustomerForm.aspx" and write the following code below the "Page Directive":

```
<%@ Register Src="~/CustomerControl.ascx" TagPrefix="NIT" TagName="CustomerControl" %>
```

Now write below code under the Web Form's " " tag:

```
<table>
  <tr><td><asp:Label ID="lblPageHeader" runat="server" /></td></tr>
  <tr><td><NIT:CustomerControl ID="CustomerControl1" runat="server" /></td></tr>
  <tr><td><asp:Label ID="lblPageFooter" runat="server" /></td></tr>
</table>
```

Now write the below code under Page Load Event:

```
lblPageHeader.Text = "Form output generated at: " + DateTime.Now.ToLongTimeString();
```

```
lblPageFooter.Text = "Form output generated at: " + DateTime.Now.ToLongTimeString();
```

Now when we run the Web Form, we will notice the Form Output and Control Output generated time will be same because we have not used caching, to use caching go to "CustomerControl.ascx" and write the following code below the "Control Directive":

```
<%@ OutputCache Duration="30" VaryByParam="none" %>
```

Now run the webform again to the see the difference in output and we will notice the control output cached but not the page output because caching is applied to control only and we call this as Fragment Caching.

VaryByControl in Caching: earlier we used an attribute known as "VaryByParam" in caching to differentiate the output generated based on the value selected in DropDownList control whereas if at all we come across the same situation in case of a "User Control" then we need to use "VaryByControl" attribute in place of "VaryByParam" attribute. To check this, add a new "Web Form User Control" under the project naming it as "MasterDetailControl.ascx" and write the following code below the "Control Directive" present over there:

```
<table style="background-color: bisque">
  <tr><td><asp:Label ID="lblControlHeader" runat="server" /></td></tr>
  <tr><td align="center"><asp:DropDownList ID="ddlDept" runat="server" AutoPostBack="True" /></td></tr>
  <tr><td><asp:GridView ID="gvEmp" runat="server" /></td></tr>
  <tr><td><asp:Label ID="lblControlFooter" runat="server" /></td></tr>
</table>
```

Write the below code under "MasterDetailControl.ascx.cs" file:

```
using System.Data;
using System.Data.SqlClient;
```

Declarations:

```
DataSet ds;
SqlDataAdapter da;
SqlConnection con;
```

Code under Page Load Event:

```
con = new SqlConnection(ReadCS.ASPDB);
if (!IsPostBack) {
  da = new SqlDataAdapter("Select * From Department", con);
  ds = new DataSet();
  da.Fill(ds, "Department");

  da.SelectCommand.CommandText = "Select * From Employee";
  da.Fill(ds, "Employee");

  ddlDept.DataSource = ds.Tables["Department"];
  ddlDept.DataTextField = "Dname";
  ddlDept.DataValueField = "Did";
  ddlDept.DataBind();
  ddlDept.Items.Insert(0, "All");
```

```
  gvEmp.DataSource = ds.Tables["Employee"];
  gvEmp.DataBind();
}
```

Code under DropDown ListSelectedIndexChanged Event:

```
if (ddlDept.SelectedIndex == 0) {
  da = new SqlDataAdapter("Select * From Employee", con);
}
else  {
  da = new SqlDataAdapter("Select * From Employee Where Did=" + ddlDept.SelectedValue, con);
}
ds = new DataSet();
da.Fill(ds, "Employee");
gvEmp.DataSource = ds.Tables["Employee"];
gvEmp.DataBind();
lblControlHeader.Text = "Control output cached at: " + DateTime.Now.ToLongTimeString();
lblControlFooter.Text = "Control output cached at: " + DateTime.Now.ToLongTimeString();
```

Now add a Web Form under the project naming it as "MasterDetailForm.aspx" and write the following code below "Page directive":

```
<%@ Register Src="~/MasterDetailControl.ascx" TagName="MasterDetailControl" TagPrefix="NIT" %>
```

Now write the following code under " " tag of the Web Form:

```
<table>
  <tr><td><asp:Label ID="lblPageHeader" runat="server" /></td></tr>
  <tr><td><NIT:MasterDetailControl runat="server" ID="MasterDetailControl1" /></td></tr>
  <tr><td><asp:Label ID="lblPageFooter" runat="server" /></td></tr>
</table>
```

Now write the below code under Page Load Event:

```
lblPageHeader.Text = "Form output generated at: " + DateTime.Now.ToLongTimeString();
lblPageFooter.Text = "Form output generated at: " + DateTime.Now.ToLongTimeString();
```

Now run the Web Form and we will not notice any difference in the Control output time and Form output time because we did not enable caching, so to enable caching, go to "MasterDetailControl.ascx" and write the following code below the "Control directive":

```
<%@ OutputCache Duration="60" VaryByControl="ddlDept" %>
```

Data Caching: it's a process of storing an amount of data into cache memory under a Web Form, which is very similar like storing data in "View State" or "Session State" or "Application State".

Syntax of storing value into cache:

Cache[string key] = value (accepts value of type object)

Syntax of accessing values from cache:

Object value = Cache[string key]

Note: apart from the above we can also store the data by using "Cache.Add" and "Cache.Insert" methods also.

Storing data in Cache is very similar to storing data in "Application State" i.e., both are "multi-user global data", whereas the difference between Application State and Cache are:

1. Application State is not Thread Safe, whereas Cache Memory is Thread Safe.
2. We can access Application State data in Global.asax file, whereas we can't access Cache Memory in Global.asax.
3. In Application State the data stays for a long time i.e., until the server restarts or the worker process is recycled, whereas data in Cache Memory will not stay for long time i.e., server can remove the values in Cache Memory at any point of time.
4. In case of Application State, data will not have any dependencies, whereas in case of Cache Memory, data can have 3 types of dependencies:
    a) Time Based Dependency.
    b) File Based Dependency.
    c) SQL Based Dependency.

Time Based Dependency: in this case the values in Cache Memory will be removed based on a time i.e., when the time expires the data in the Cache Memory is deleted. This is again divided in to 2 types:

1. **Absolute Expiration:** In this case the data in the Cache Memory stays for a specified time which is calculated from the first time it was stored in the Cache and once the time elapses data in Cache Memory is removed.
2. **Sliding Expiration:** In this case the data in the Cache Memory stays for a specified time which is calculated from the last visit and once the time elapses data in Cache Memory is removed.

To test this, add a new Web Form under the project naming it as "DataCachingTimeBased.aspx", place a GridView control on it and write the below code in "DataCachingTimeBased.aspx.cs" file.

```
using System.Data;
using System.Web.Caching;
using System.Data.SqlClient;
```

Declarations:

```
DataSet ds;
```

Code under Page_Load:

```
if (Cache["CustomerDS"] == null) {
  SqlDataAdapter da = new SqlDataAdapter("Select * From Customer", ReadCS.ASPDB);
  ds = new DataSet();
  da.Fill(ds, "Customer");
  Cache.Insert("CustomerDS", ds);
  Response.Write("Data loaded from database server.");
}
else {
  ds = (DataSet)Cache["CustomerDS"];
  Response.Write("Data loaded from cache memory");
}
GridView1.DataSource = ds;
GridView1.DataBind();
```

In the above case we haven't applied a dependency on the cache, so the values get stored in the Cache Memory until the Server explicitly removes it, whereas we can apply any of the 3 types of dependencies we have discussed above. Now let us implement Time Based Dependency for the above program:

Implementing Absolute Expiration: to implement this re-write the "Cache.Insert" method code in the above program as following:

```
Cache.Insert("CustomerDS", ds, null, DateTime.UtcNow.AddSeconds(30), Cache.NoSlidingExpiration);
```

In the above case because we have applied Absolute Expiration to Cache, data stays in the Cache Memory for 30 seconds from the point it was stored in the cache.

Implementing Sliding Expiration: to implement this re-write the "Cache.Insert" method code in the above program as following:

```
Cache.Insert("CustomerDS", ds, null, Cache.NoAbsoluteExpiration, TimeSpan.FromSeconds(30));
```

In the above case because we have applied Sliding Expiration to Cache, data stays in the Cache Memory for 30 seconds from the last visit to the page.

File Based Dependency: in this case the data in the Cache is dependent on some File and whenever the data in the file changes automatically the cache data gets flushed out. To use file based dependency we need to monitor the file present on the hard disk by using the class "CacheDependency" which is present under the namespace "System.Web.Caching" i.e., we need to create the instance of "CacheDependency" class by passing "Filename" which we want to monitor as a parameter to the Constructor and then "CacheDependency" class instance should be passed as a parameter to "Cache.Insert" method, so that whenever the data in file changes immediately it gets flushed out from the cache memory. To test this, add an XML File under the project naming it as "Products.xml" and write the below code in the file:

```
<Products>
  <Product>
    <Id>101</Id>
    <Name>Television</Name>
  </Product>
  <Product>
    <Id>102</Id>
    <Name>Microwave</Name>
  </Product>
  <Product>
    <Id>103</Id>
    <Name>Washing Machine</Name>
  </Product>
  <Product>
    <Id>104</Id>
    <Name>Refrigerator</Name>
  </Product>
  <Product>
    <Id>105</Id>
    <Name>Home Theatre</Name>
```

```
  </Product>
  <Product>
    <Id>106</Id>
    <Name>Mixer Grinder</Name>
  </Product>
  <Product>
    <Id>107</Id>
    <Name>Blueray Player</Name>
  </Product>
  <Product>
    <Id>108</Id>
    <Name>Split A/C</Name>
  </Product>
  <Product>
    <Id>109</Id>
    <Name>Air Cooler</Name>
  </Product>
  <Product>
    <Id>110</Id>
    <Name>Window A/C</Name>
  </Product>
</Products>
```

Now add a new Web Form under the project naming it as "DataCachingFileBased.aspx", place a "GridView" control on it and write the below code under "DataCachingFileBased.aspx.cs" file.

```
using System.Data;
using System.Web.Caching;
```

Code Under Page Load Event:

```
DataSet ds;
if (Cache["ProductDS"] == null) {
  ds = new DataSet();
  ds.ReadXml(Server.MapPath("~/Products.xml"));
  CacheDependency cd = new CacheDependency(Server.MapPath("~/Products.xml"));
  Cache.Insert("ProductDS", ds, cd);
  Response.Write("Data loaded from XML File.");
}
else {
  ds = (DataSet)Cache["ProductDS"];
  Response.Write("Data loaded from Cache.");
}
GridView1.DataSource = ds.Tables[0];
GridView1.DataBind();
```

Run the above Web Form and watch the output and in this case data in Cache Memory will be staying in Cache until the file is modified and once the file is modified it will immediately flush the data from cache.

SQL Based Dependency: Flushing out data from the cache memory based on the changes made in the database table is known as "SQL Based Dependency" i.e., whenever the data in the table changes, automatically the data in the Cache is flushed out. To work with SQL Based Dependency we need to first perform 3 actions:

1. We need to enable SQL Cache Dependency on the database and table explicitly by using "aspnet_regsql" tool.
2. We need to specify the Cache Details under the config file.
3. We need to create the SQLCacheDependency class instance by specifying the Database Name and Table Name which should be passed as a parameter to the Insert method of Cache.

Step 1: open Visual Studio Developer Command Prompt and enable Cache Dependency to the Database and Table as following:

Enabling Cache Dependency for Database and Table:

SQL Authentication: aspnet_regsql -S Server -U Sa -P 123 -d ASPDB -ed -t Customer -et
Windows Authentication: aspnet_regsql -S Server -E -d ASPDB -ed -t Customer -et

Note: we can also disable the enabled Cache Dependency on Database with the following statement:

SQL Authentication: aspnet_regsql -S Server -U Sa -P 123 -d ASPDB -dd
Windows Authentication: aspnet_regsql -S Server -E -d ASPDB -dd

Step 2: open "Web.config" file and specify the Cache details under "<system.web>" tag as following:

```
<caching>
  <sqlCacheDependency enabled="true">
    <databases>
      <add name="ASPDB" connectionStringName="ConStr" pollTime="2000"  />
    </databases>
  </sqlCacheDependency>
</caching>
```

Note: pollTime is an attribute to specify when IIS must contact the Database Engine for any change notifications and the default pollTime is 500 milliseconds.

Now add a new Web Form under the project naming it as "DataCachingSQLBased.aspx", place a "GridView" control on it and write the below code under "DataCachingSQLBased.aspx.cs" file.

```
using System.Data;
using System.Web.Caching;
using System.Data.SqlClient;
```

Code under Page Load Event:

```
DataSet ds;
if (Cache["CustomerDS"] == null) {
  SqlDataAdapter da = new SqlDataAdapter("Select * From Customer", ReadCS.ASPDB);
  ds = new DataSet();
  da.Fill(ds, "Customer");
```

```
  SqlCacheDependency cd = new SqlCacheDependency("ASPDB", "Customer");
  Cache.Insert("CustomerDS", ds, cd);
  Response.Write("Data loaded from database server.");
}
else {
  ds = (DataSet)Cache["CustomerDS"];
  Response.Write("Data loaded from cache memory");
}
GridView1.DataSource = ds.Tables[0];
GridView1.DataBind();
```

Now any changes that are made in the Database to the table will flush the Data in cache and reloads it when the next request comes to the page.

Note: we can also explicitly remove values from Cache Memory by calling "Cache.Remove" method by passing the "Key Name" string as a parameter to the method.

E.g.: Cache.Remove(string key)

XML Web Services

A "Web Service" is a web application which is basically a class consisting of methods called "Web Methods" that could be used by other applications. It means that we can create a web service in any language, such as Java or other languages and that web service can be used in a .NET based application and a .NET web service in any other application to exchange the information. It follows code-behind architecture such as the ASP.NET Web Forms, although it does not have a user interface.

Web Method: Methods in web services attached or decorated with [WebMethod] attribute is known as Web Methods, which indicates that we want the method exposed as part of the XML Web service.

The WebMethod attribute has the following properties:

1. BufferResponse
2. CacheDuration
3. Description
4. EnableSession
5. MessageName

BufferResponse: This property of Web Method attribute enables buffering of responses for an XML Web service method. When set to true [default], ASP.NET buffers the entire response before sending it down to the client. The buffering is very efficient and helps improve performance by minimizing communication between the worker process and the IIS process. When set to false, ASP.NET buffers the response in chunks of 16 KB. Typically, you would set this property to false only if you did not want the entire contents of the response in memory at once. For example, you are writing back a collection that is streaming its items out of a database. Unless otherwise specified, the default value is true.

CacheDuration: This property of the Web Method attribute enables caching of the results for an XML Web service method. ASP.NET will cache the results for each unique parameter set. The value of this property specifies how

many seconds ASP.NET should cache the results. A value of zero disables the caching of results. Unless otherwise specified, the default value is zero.

Description: This property of the Web Method attribute supplies a description for an XML Web service method that will appear on the Service help page.

EnableSession: This property of the Web Method attribute enables session state for an XML Web service method. Once enabled, the XML Web service can access the session state collection directly, provided the Web Service class inherits from "System.Web.Services.WebService" class. Unless otherwise specified, the default value is false.

MessageName: This property of the WebMethod attribute enables the XML Web service to uniquely identify overloaded methods using an alias. Unless otherwise specified, the default value is method name. When specified the MessageName, the resulting SOAP messages will reflect this name instead of the actual method name.

SOAP and Web Services: SOAP (simple object access protocol) is an XML-based protocol for exchanging information between computers. In short, SOAP is the way by which method calls translate into XML format and sent via HTTP. SOAP is a standard XML based protocol that communicated over HTTP. We can think of SOAP as message format for sending messages between applications using XML. It is independent of technology, language, and platform. It doesn't mean that we must write XML and SOAP specific things ourselves to facilitate these communications, ASP.NET will take care of doing the low-level SOAP and XML work for us.

WSDL: It stands for Web Service Description Language, is a standard format for describing a web service by which a web service can tell clients what input it accepts and which results it will return. WSDL contains every detail regarding using Web Service and its Method provided by Web Service and URLs from which those methods can be accessed, and Data Types used. WSDL definition describes how to access a web service and what operations it will perform. Once we implement an ASP.NET Web Service, Visual Studio automatically generates Web Services Description Language (WSDL) file for each Web Service on that application. When you type the URL of a Web service appended by the "?wsdl" parameter in a Web browser, the ASP.NET application returns the WSDL file, which contains the WSDL binding definition for the Web service.

Developing a Web Service: create a new "ASP.NET Web Application" Project naming it as "WebServiceProject" => Choose Empty, but don't select "Web Forms" or any other CheckBox and click on the "Create" button. Host the project under IIS Web Server. Now right click on the project in Solution Explorer => Select Add => New Item and choose "Web Service (ASMX)" which will add an item with an extension of ".asmx" name it as "FirstService.asmx" and click ok and under the file we find a class "FirstService" inheriting from "System.Web.Services.WebService" and this class uses an attribute known as [WebService] to tell this is a Web Service which will be as following:

```
[WebService(Namespace = "http://tempuri.org/")]
```

Namespace is to identify our web services uniquely after publishing them, so generally we use our company name as the namespace, so let us change this as following:

```
[WebService(Namespace = "http://nareshit.com/")]
```

Under the class we find a method "HelloWorld" which uses an attribute [WebMethod] to specify that the method is a Web Method and if this attribute is not present the method can't be accessed by the clients and the method looks as following:

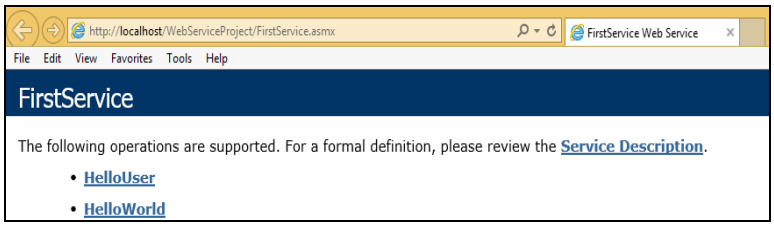
```
[WebMethod]
public string HelloWorld()
{
```

```
  return "Hello World";
}
```

Under this method let us add another method "HelloUser" and that method also should have the [WebMethod] attribute as following:

```
[WebMethod]
public string HelloUser(string Name)
{
  return "Hello " + Name;
}
```

Now if the run the class by hitting F5 it will display as following under the browser:



In the above "HelloUser" and "HelloWorld" methods are displayed as Hyper Links and when we click on a method it will display a window with "Invoke" button and if the method requires any Input, it will also provide "Textbox's" for entering the Input values, which we can enter and click on the "Invoke" button which displays the result in a new window.

Note: if we want to view the "wsdl" code of this service, add "?wsdl" to the URL and then hit enter, as following:

http://localhost/WebServiceProject/FirstService.asmx?wsdl

Consuming the Web Service: to consume the Web Service we have developed open a new ASP.NET Web Application project choosing "Web Forms", name it as "WebServiceConsumer" and click on the "Create" button. Host the application, IIS Web Server.

Open solution explorer, right click on the "References" option under project and select "Add Service Reference" which opens a window and in that enter the following URL in the TextBox: http://localhost/WebServiceProject/FirstService.asmx and click on "Go" button which displays our "FirstService" below. In the bottom of the window, we will find a namespace option with a value "ServiceReference1", either leave the same or change to any meaningful name like "TestFS" and click on "Ok" button which will generate a proxy class for consuming the Web Methods and that will be under a new folder that is add under project with the name "Service References".

Note: To watch the Proxy class, open Visual Studio => select the project "WebServiceConsumer" and on the top we find an option "Show All Files" select it => expand "Services References" node => expand "TestFS" node => expand "Reference.svcmap" node and double click on "Reference.cs" file which will display the contents of that file and in that we find a class with the name "FirstServiceSoapClient" which is our proxy class using which we can call the Web Methods.

Now add a new WebForm under the project naming it as "WebForm1.aspx" and write the following code under its " " tag:

```
<asp:Button ID="Button1" runat="server" Text="Call Hello World" />
<asp:Label ID="Label1" runat="server" ForeColor="Red" /><br />
Enter Name: <asp:TextBox ID="txtName" runat="server" />
<asp:Button ID="Button2" runat="server" Text="Call Hello User" />
<asp:Label ID="Label2" runat="server" ForeColor="Red" />
```

Now write the below code under "WebForm1.aspx.cs" file:

```
using WebServiceConsumer.TestFS;
```

Declarations:

```
FirstServiceSoapClient obj = new FirstServiceSoapClient();
```

Code under "Call Hello World" Button Click Event:

```
Label1.Text = obj.HelloWorld();
```

Code under "Call Hello User" Button Click Event:

```
Label2.Text = obj.HelloUser(txtName.Text);
```

Creating a new Web Service to understand the properties of WebMethod attribute: add a new Service under the "WebServiceProject" naming it as "SecondService.asmx" and change the "WebService Namespace" values as http://nareshit.com/ and below the "WebService" attribute we find another attribute "WebServiceBinding" as following: "[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]" which should be changed as following: "[WebServiceBinding(ConformsTo = WsiProfiles.None)]" and then write the below code under "SecondService" class deleting the existing "HelloWorld" method:

```
[WebMethod(Description = "This method wishes a  user.", MessageName = "SayHello1")]
public string SayHello() {
  return "Hello Mr./Ms. have a nice day.";
}
[WebMethod(Description = "This method wishes a user with his/her name.", MessageName = "SayHello2")]
public string SayHello(string name) {
  return "Hello Mr./Ms. " + name + " have a nice day.";
}
[WebMethod(Description = "This method adds 2 integer values and will cache the output.", CacheDuration = 30)]
public string AddNums(int a, int b) {
  int c = a + b;
  return "Output generated at: " + DateTime.Now.ToLongTimeString() + " and the value is: " + c;
}
[WebMethod(Description = "This method increments a number.", EnableSession = true)]
public string IncrementValue() {
  int Count = 0;
  if (Session["Counter"] == null)
  {
    Count = 1;
```

```
  }
  else
  {
    Count = (int)Session["Counter"] + 1;
  }
  Session["Counter"] = Count;
  return "Count value is: " + Count;
}
```

Run the class to test all the methods and now to consume those methods add a new Web Form under "WebServiceConsumer" project naming it as "WebForm2.aspx" and write the below code under its " " tag:

```
<asp:Button ID="Button1" runat="server" Text="Call SayHello()" Width="200px" /><br />
<asp:Label ID="Label1" runat="server" ForeColor="Red" /><br />
<asp:Button ID="Button2" runat="server" Text="Call SayHello(string)" Width="200px" />
<asp:TextBox ID="txtName" runat="server" /><br />
<asp:Label ID="Label2" runat="server" ForeColor="Red" /><br />
<asp:Button ID="Button3" runat="server" Text="Call AddNums(int, int)" Width="200px" />
<asp:TextBox ID="txtNum1" runat="server" />
<asp:TextBox ID="txtNum2" runat="server" /><br />
<asp:Label ID="Label3" runat="server" ForeColor="Red" /><br />
<asp:Button ID="Button4" runat="server" Text="Call IncrementValue()" Width="200px" /><br />
<asp:Label ID="Label4" runat="server" ForeColor="Red" />
```

Add "WebReference" to the "WebServiceConsumer" Project target the "SecondService" class by the using following URL http://localhost/WebServiceProject/SecondService.asmx and name the namespace as "TestSS" which creates a proxy class with the name "SecondServiceSoapClient".

Now write the below code under "WebForm2.aspx.cs" file:

```
using WebServiceConsumer.TestSS;
```

Declarations:

```
SecondServiceSoapClient obj = new SecondServiceSoapClient();
```

Code under Button1 Click Event:

```
Label1.Text = obj.SayHello();
```

Code under Button2 Click Event:

```
Label2.Text = obj.SayHello1(txtName.Text);
```

Code under Button3 Click Event:

```
Label3.Text = obj.AddNums(int.Parse(txtNum1.Text), int.Parse(txtNum2.Text));
```

Code under Button4 Click Event:

```
Label4.Text = obj.IncrementValue();
```

Note: in the above case even if the "IncrementValue" method is enabled with Session still it will not increment the value because the "Session Id" is not submitted to the "Web Service" by our application so to resolve the problem open the "Web.config" file and there we find "<system.serviceModel>" tag and under that we find "<binding name="SecondServiceSoap" />" and to this add "allowCookies="true"" which should now look as following:

```
<binding name="SecondServiceSoap" allowCookies="true" />
```

Creating a Web Service performing Database Operations: add a new Service under the "WebServiceProject" naming it as "DBService.asmx", change the "WebService Namespace" values as http://nareshit.com/ and write the below code under the class by deleting the existing "HelloWorld" method:

```
using System.Data;
using System.Data.SqlClient;
```

```
[WebMethod(Description = "Returns all the table names in the given Database")]
public DataSet GetTableNames(string DBName) {
  SqlConnection con = new SqlConnection("User Id=Sa;Password=123;Data Source=Server;Database=" + DBName);
  SqlDataAdapter da = new SqlDataAdapter("Select Name From SysObjects Where xtype='U'", con);
  DataSet ds = new DataSet();
  da.Fill(ds, "SysObjects");
  return ds;
}
[WebMethod(BufferResponse = false, Description = "Returns data of given table.")]
public DataSet GetTableData(string TableName, string DBName) {
  SqlConnection con = new SqlConnection("User Id=Sa;Password=123;Data Source=Server;Database=" + DBName);
  SqlDataAdapter da = new SqlDataAdapter("Select * From " + TableName, con);
  DataSet ds = new DataSet();
  da.Fill(ds, TableName);
  return ds;
}
```

Run the class to test all the methods and now to consume those methods add a new Web Form under the "WebServiceConsumer" project naming it as "WebForm3.aspx" and write the following code under its " " tag:

```
<table align="center">
  <tr>
    <td align="right">DB Name:</td>
    <td>
      <asp:TextBox ID="txtDBName" runat="server" Width="150px" />
      <asp:Button ID="Button1" runat="server" Text="Get Tables" />
    </td>
  </tr>
  <tr>
    <td align="right">Tables:</td>
    <td><asp:DropDownList ID="ddlTables" runat="server" AutoPostBack="True" Width="155px" /></td>
  </tr>
  <tr><td colspan="2" align="center"><asp:GridView ID="GridView1" runat="server" /></td></tr>
</table>
```

Add "WebReference" to the "WebServiceConsumer" Project targeting "DBService" class by using this URL http://localhost/WebServiceProject/DBService.asmx and name the namespace as "TestDBS" which creates a proxy class with the name "DBServiceSoapClient".

Now write the below code under "WebForm3.aspx.cs" file:

```
using WebServiceConsumer.TestDBS;
```

Declarations:

```
DBServiceSoapClient obj = new DBServiceSoapClient();
```

Code under Get Tables Button Click Event:

```
DataSet ds = obj.GetTableNames(txtDBName.Text);
ddlTables.DataSource = ds;
ddlTables.DataTextField = "Name";
ddlTables.DataValueField = "Name";
ddlTables.DataBind();
ddlTables.Items.Insert(0, "-Select Table-");
```

Code under DropDownList SelectedIndexChanged Event:

```
if (ddlTables.SelectedIndex > 0) {
  DataSet ds = obj.GetTableData(ddlTables.SelectedValue, txtDBName.Text);
  GridView1.DataSource = ds;
}
else {
  GridView1.DataSource = null;
}
GridView1.DataBind();
```