

Contents

ASP.Net 4.5 Architecture	3
NET Framework	3
ASP.NET Framework	3
Asp.NET Site	3
Web Forms	3
MVC	3
Web Pages	4
SPA (Single Page Application)	4
Asp.NET Services	4
IIS	4
Web Form Controls	6
Master Page	9
Aspt.Net Page Life Cycle	16
Request, Response and Server Objects	17
Server Object	17
Properties and Methods of the Server object	17
Request Object	18
Properties and Methods of the Request Object	19
Response Object	20
Properties and Methods of the Response Object	21
Example	23
Get & Post Method	24
GET method:	24
Post Method:	25
State Management	26
View Sate	26
Session	27
Application State	28
Cookies	29
Page Navigation	30

User Controls	33
Custom Controls	37
Authentication and Authorization	42
Detecting authentication and authorization: - The principal and identity objects	43
Windows Authentication	45
Basic Authentication	50
Base64 is an encoding mechanism and not encryption	52
Digest Authentication	52
Comparison of Basic, digest and windows authentication	56
Forms Authentication	57
Forms authentication using 'web.config' as a data store	58
Passport Authentication	65

ASP.Net 4.5 Architecture

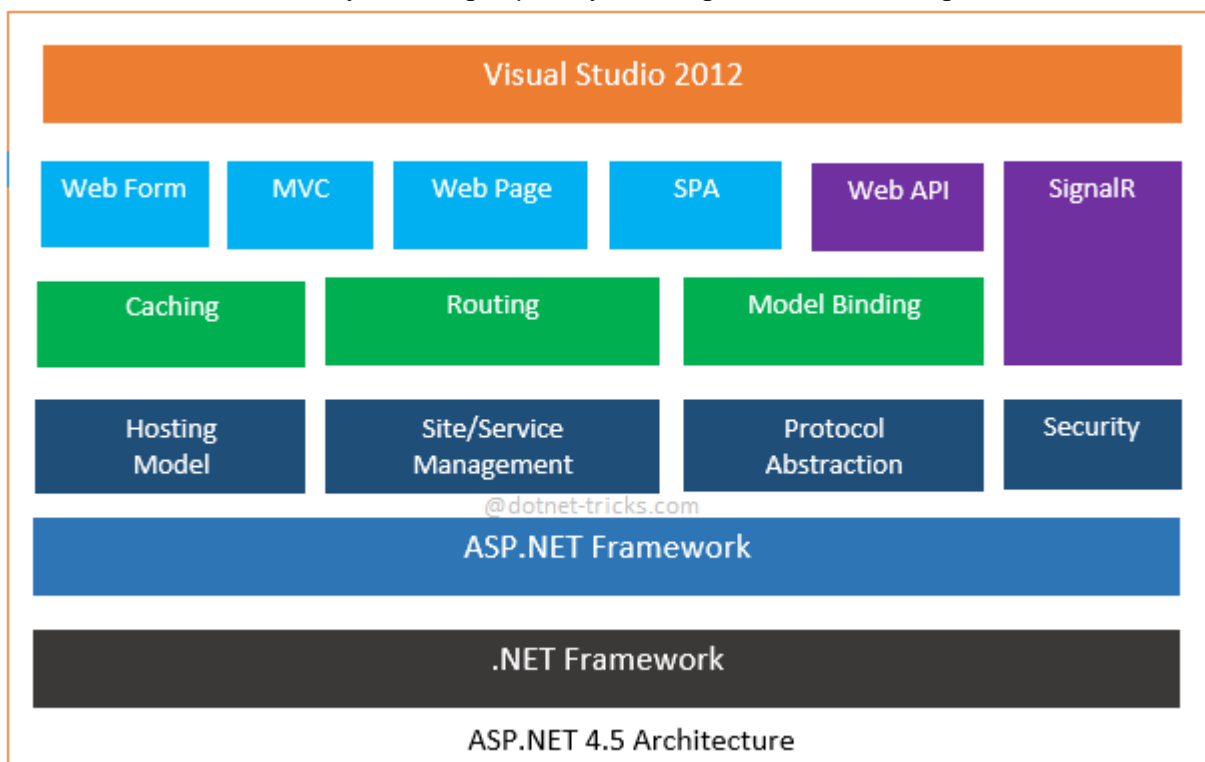
NET Framework

.Net framework is an integrated component of windows operating system that supports development and execution of next generation applications, Windows store apps and services.

ASP.NET Framework

ASP.Net Framework is used to create dynamic website, web application and web services. It is built on the top of .NET Framework.

Asp.NET Framework provides you various capabilities like Hosting Model, Site/Service Management, Protocol Abstraction, Security, Caching capability, Routing and Model Binding etc.



Mainly, Asp.Net can be divided into two parts - Asp.Net Sites and Asp.Net Services.

Asp.NET Site

There are following flavours of Asp.NET Site -

Web Forms

This is the traditional event driven development model. It has drag and drop server controls, server events and state management techniques.

MVC

This is a lightweight and MVC (Model, View, and Controller) pattern based development model. It provides full control over mark-up and support many features that allow fast & agile development. This is best for developing lightweight, interactive and device oriented (i.e. compatible to smart phones, iPhone, tablet, laptop etc.) web application with latest web standards.

Web Pages

This is also a lightweight and Razor syntax based development model. It has built-in template and helpers also provide full control over mark-up. It is best for developing beautiful web application with latest web standards. You can also use WebMatrix which is a free tool and has built-in template; for developing Asp.Net Web Page.

SPA (Single Page Application)

SPA stands for Single Page Application which helps you to build web applications that include significant client-side interactions using HTML5, CSS3 and JavaScript. It is best to make highly interactive single page dashboard web applications.

Asp.NET Services

There are two ways to make Asp.Net Services as given below –

Web API

Asp.Net Web API is a framework for building HTTP services that can be consume by a broad range of clients including browsers, mobiles, iPhone and tablets.

SignalR

ASP.NET SignalR is a library that simplifies the process of adding real-time web functionality to applications. Real-time web functionality is the ability to have server code push content to connected clients instantly as it becomes available, rather than having the server wait for a client to request new data.

IIS

Introduction

When request come from client to the server a lot of operation is performed before sending response to the client. This is all about how IIS Process the request. Here I am not going to describe the Page Life Cycle and there events, this article is all about the operation of IIS Level.

What is Web Server?

When we run our ASP.NET Web Application from visual studio IDE, Visual Studio Integrated ASP.NET Engine is responsible to execute all kind of asp.net requests and responses. The process name is “WebDev.WebServer.Exe” which actually takes care of all request and response of a web application which is running from Visual Studio IDE.

Now, the name “Web Server” comes into picture when we want to host the application on a centralized location and wanted to access from many locations. Web server is responsible for handle all the requests that are coming from clients, process them and provide the responses.

What is IIS ?

IIS (Internet Information Server or Internet Information Service) is one of the most powerful web servers from Microsoft that is used to host your ASP.NET Web application. IIS has its own ASP.NET Process Engine to handle the ASP.NET request. So, when a request comes from client to server, IIS takes that request and process it and send response back to clients.

Worker Process: Worker Process (w3wp.exe) runs the ASP.Net application in IIS. This process is responsible to manage all the request and response that are coming from client system. All the ASP.Net functionality runs under the scope of worker process. When a request comes to the server from a client

worker process is responsible to generate the request and response. In a single word we can say worker process is the heart of ASP.NET Web Application which runs on IIS.

Application Pool: Application pool is the container of worker process. Application pools are used to separate sets of IIS worker processes that share the same configuration. Application pools enable a better security, reliability, and availability for any web application. The worker process serves as the process boundary that separates each application pool so that when one worker process or application is having an issue or recycles, other applications or worker processes are not affected. This makes sure that a particular web application doesn't not impact other web application as they are configured into different application pools.

Application Pool with multiple worker process is called "Web Garden".

Now let's have looked how IIS process the request when a new request comes up from client.

If we look into the IIS 6.0 Architecture, we can divided them into Two Layer

1. Kernel Mode
2. User Mode

Now, Kernel mode is introduced with IIS 6.0, which contains the HTTP.SYS. So whenever a request comes from Client to Server, it will hit HTTP.SYS First.

Now, HTTP.SYS is Responsible for pass the request to particular Application pool. Now here is one question, How HTTP.SYS comes to know where to send the request? This is not a random pickup. Whenever we create a new Application Pool, the ID of the Application Pool is being generated and it's registered with the HTTP.SYS. So whenever HTTP.SYS Received the request from any web application, it checks for the Application Pool and based on the application pool it send the request.

So, this was the first steps of IIS Request Processing.

Till now, Client Requested for some information and request came to the Kernel level of IIS means at HTTP.SYS. HTTP.SYS has been identified the name of the application pool where to send. Now, let's see how this request moves from HTTP.SYS to Application Pool.

In User Level of IIS, we have Web Admin Services (WAS) which takes the request from HTTP.SYS and pass it to the respective application pool.

When Application pool receives the request, it simply passes the request to worker process (w3wp.exe). The worker process "w3wp.exe" looks up the URL of the request in order to load the correct ISAPI extension. ISAPI extensions are the IIS way to handle requests for different resources. Once ASP.NET is installed, it installs its own ISAPI extension (aspnet_isapi.dll) and adds the mapping into IIS.

Note: Sometimes if we install IIS after installing asp.net, we need to register the extension with IIS using aspnet_regiis command.

When Worker process loads the aspnet_isapi.dll, it starts an HTTPRuntime, which is the entry point of an application. HTTPRuntime is a class which calls the ProcessRequest method to start Processing.

When this methods called, a new instance of HttpContext is been created. This is accessible using HttpContext.Current Properties. This object still remains alive during life time of object request. Using HttpContext.Current we can access some other objects like Request, Response, and Session etc.

After that HttpRuntime load an HttpApplication object with the help of HttpApplicationFactory class... Each and every request should pass through the corresponding HTTPModule to reach to HTTPHandler, this list of module are configured by the HttpApplication.

Now, the concept comes called "HTTPPipeline". It is called a pipeline because it contains a set of HttpModules (For Both Web.config and Machine.config level) that intercept the request on its way to the HttpHandler. HttpModules are classes that have access to the incoming request. We can also create our own HTTPModule if we need to handle anything during upcoming request and response.

HTTP Handlers are the endpoints in the HTTP pipeline. All requests that are passing through the HTTPModule should reach to HTTPHandler. Then HTTP Handler generates the output for the requested resource. So, when we requesting for any aspx web pages, it returns the corresponding HTML output.

The entire request now passes from HTTPModule to respective HTTPHandler then method and the ASP.NET Page life cycle starts. This ends the IIS Request processing and starts the ASP.NET Page Lifecycle.

Conclusion

When client request for some information from a web server, request first reaches to HTTP.SYS of IIS. HTTP.SYS then sends the request to respective Application Pool. Application Pool then forward the request to worker process to load the ISAPI Extension which will create an HTTPRuntime Object to Process the request via HTTPModule and HttpHandler. After that the ASP.NET Page Lifecycle events start.

This was just overview of IIS Request Processing to let Beginner's know how the request gets processed in backend. If you want to learn in details please check the link for Reference and further Study section.

Web Form Controls

There are some Web Forms Controls which is given below with Real Time Example:-

1. Label Control
2. Textbox control
3. Button control
4. Literal control
5. Placeholder control
6. Hidden Field control
7. File Upload control
8. Image control
9. Image Button control
10. Image Map control

Label Control

The Label Control is basically used to display the Information (Text) on Web Forms. Any end User cannot Edit (change) the Label Information.

Properties of Label Control:-

There are some important properties of Label controls.

- **Text**:-It is used to change the Label control information (text).
- **Font**:- It is used to sets the font of the Label text.
- **ForeColor**:- It is used to sets the text color in the Label.
- **Height**:- It is used to Specify the height of the Label Control.
- **BackColor**:-It is used to Sets the Background color of the Label control.
- **BorderWidth**:-It is used to Sets the Border color of the Label control.
- **BorderStyle**:-It is used to Sets the Style of the Label control.
- **AccessKey**:-It is used to Navigate the Web Server control.

Example: You can assign the value

```
Label2.Text = "You can easily display any information inside it";
```

TextBox control is used to input the Data(text).Any end user can easily enter the text in this control.

Properties of TextBox Control:-

There are some important properties of TextBox controls.

- **Text**:-It is used to sets the text in TextBox Control.
- **TextMode**:- It is used to sets the mode of the TextBox Control as Single Mode,Multiline or Password.
- **Rows**:-It is used to sets the number of rows display in a TextBox(Multiline).
- **MaxLength**:->It is used to sets the maximum number of character allowed in a TextBox control.
- **ReadOnly**:-It is used to read the contents by the end user but can not change it.
- **Columns**:-It is used to sets the width of character in TextBox.

- **AutoCompleteType**:-It is used to sets a value that indicates the Auto Complete behavior of the TextBox control.
- **AutoPostBack**:-It is used to handle the event when the TextBox control lose focus.
- **CauseValidation**:-It is used to set a value that validate the TextBox control(client,server).
- **TextChanged**:-It is an event.It occurs when the end user change the text of the TextBox control.

Button Control:-> The Button control is used to create an Event and send request to the web server.

Properties of Button Control:-

There are some important properties of Button controls.

- **Text**:- It is used to sets the text to be displayed on the Button Control.
- **Click**:-It is an Event that occurs when the Button is clicked.
- **Command**:- It is also an Event that occurs when the Button is clicked.It is used whenever we are using multiple Button on one page.
- **CommandName**:-It is used to sets the command Name associated with the Button control that is passed to the command event.
- **OnClientClick**:-It is used to sets the client side script that executes when a Button click event is fired.
- **CauseValidation**:- It is used to sets the validation that is perform or not when we click the Button control.

Example:

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

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        String str = TextBox1.Text;
        Label2.Text = str;
    }
}
```

Literal Control

It is similar to Label control but there are some differences and similarities which is given below.

- Literal and Label control both are used to display the text(information) on the web form.
- On Browser side Label control is converted to HTML '**Span**' tag but Literal control is not converted in any HTML tag.
- We can provide formatting to the Label control but not to the Literal control.
- If we want to show some HTML code and java script code the we used mainly Literal control ,not Label control.Encoding Mode is used with Literal control only.

Properties of Literal Control:-

There are some important properties of Literal controls.

- **Mode**:- IT IS A Literal control property called Mode which can be changed as Transform,Encode and Pass Through.
- **Text**:-It is used to sets the caption displayed in the Literal.

Example:

```
protected void Button1_Click(object sender, EventArgs e)
{
    Literal1.Text = "Hello Asp.net Website I am Literal Control";
    Label2.Text = "Hello Asp.net Website I am Label Control";
}
```

Placeholder Control

It is known as container control in Asp.Net. It is mainly used to store the value of other controls. You can Add controls to a Web Form by using the **Controls.Add()** Method.

Example:

```
protected void Button1_Click(object sender, EventArgs e)
{
    Placeholder1.Controls.Add(TextBox1);
}
```

HiddenField Control

It is used to display the value stored in the HiddenField control in the Label control. It stores the information in the form of **Strings**.

Example:

```
protected void Button1_Click(object sender, EventArgs e)
{
    HiddenField1.Value = TextBox1.Text;
    Label2.Text = HiddenField1.Value;
}
```

FileUpload Control

It is basically used to Upload the File on the server using **SaveAs()** method on the click Event. When user browse a File and after browse click the Upload button then File is automatically Upload on the Server.

Example:

```
protected void Button1_Click(object sender, EventArgs e)
{
    FileUpload1.SaveAs("I:" + FileUpload1.FileName);
    Label2.Text = "File" + " " + FileUpload1.FileName + " "+"is uploaded successfully";
}
```

Image Control

It is used to display an image on a web page.

Properties of Image control:-

There are some important properties of Image control controls.

- **Alternate Text**:- It is used to display alternate text when the image is not present.
- **ImageAlign**:- It is used to sets the alignment of image control on the web page.
- **ImageUrl**:- It is used to sets the path to an image to display in the image control.
- **Font**:- It is used to increase or decrease font of the image control text.
- **DescriptionUrl**:- It is used to set the location to a detailed description for the image.

ImageButton Control

It is used to display the image on Button instead of text. Properties of ImageButton is same as Button property.

ImageMap Control

It is used to provide various links(hotspots) to navigate to other web page,depending on the place where the user clicks.

Properties of ImageMap Control:-

There are some important properties of ImageMap control.

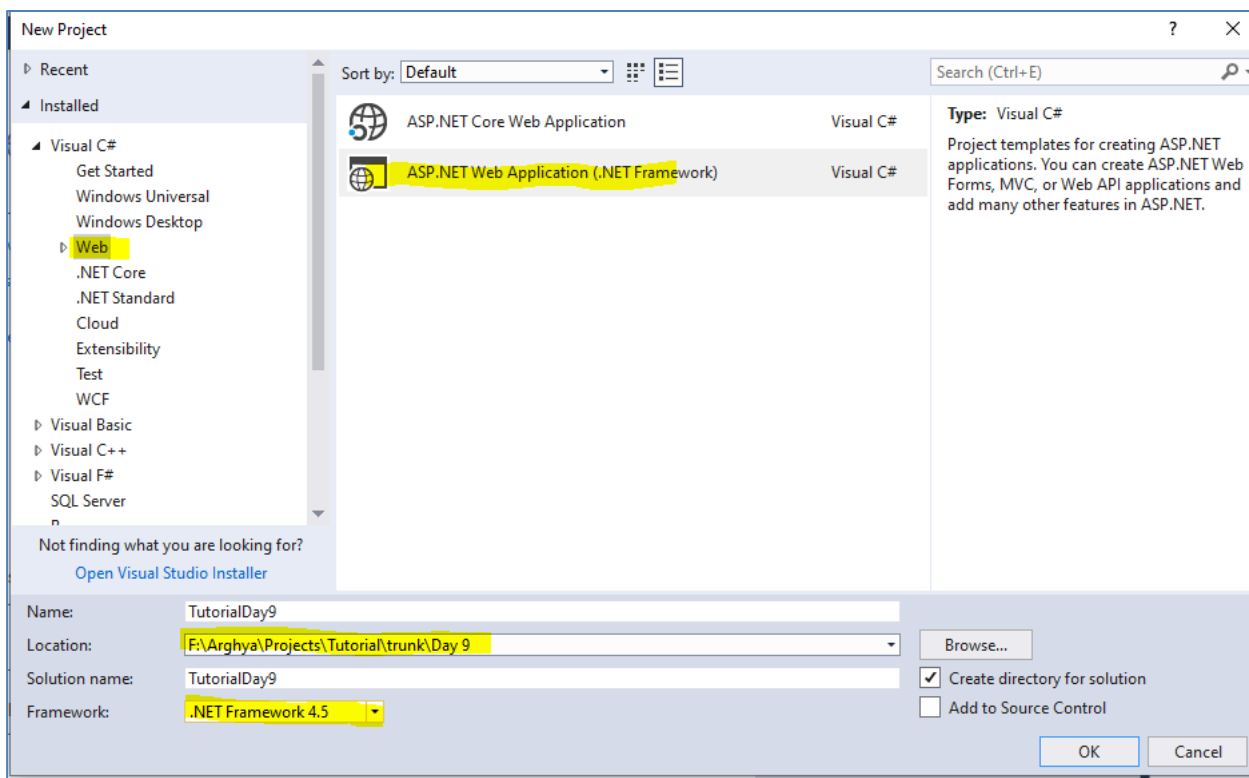
- **Hotspots:-**It obtains a group of Hotspot Object that is defined in a ImageMap.
- **Target:-**It is used to sets target window to show the web page when the ImageMap control clicked.
- **Enabled:-** It is used to sets a value indicating whether the control can respond to the user interaction.

Master Page

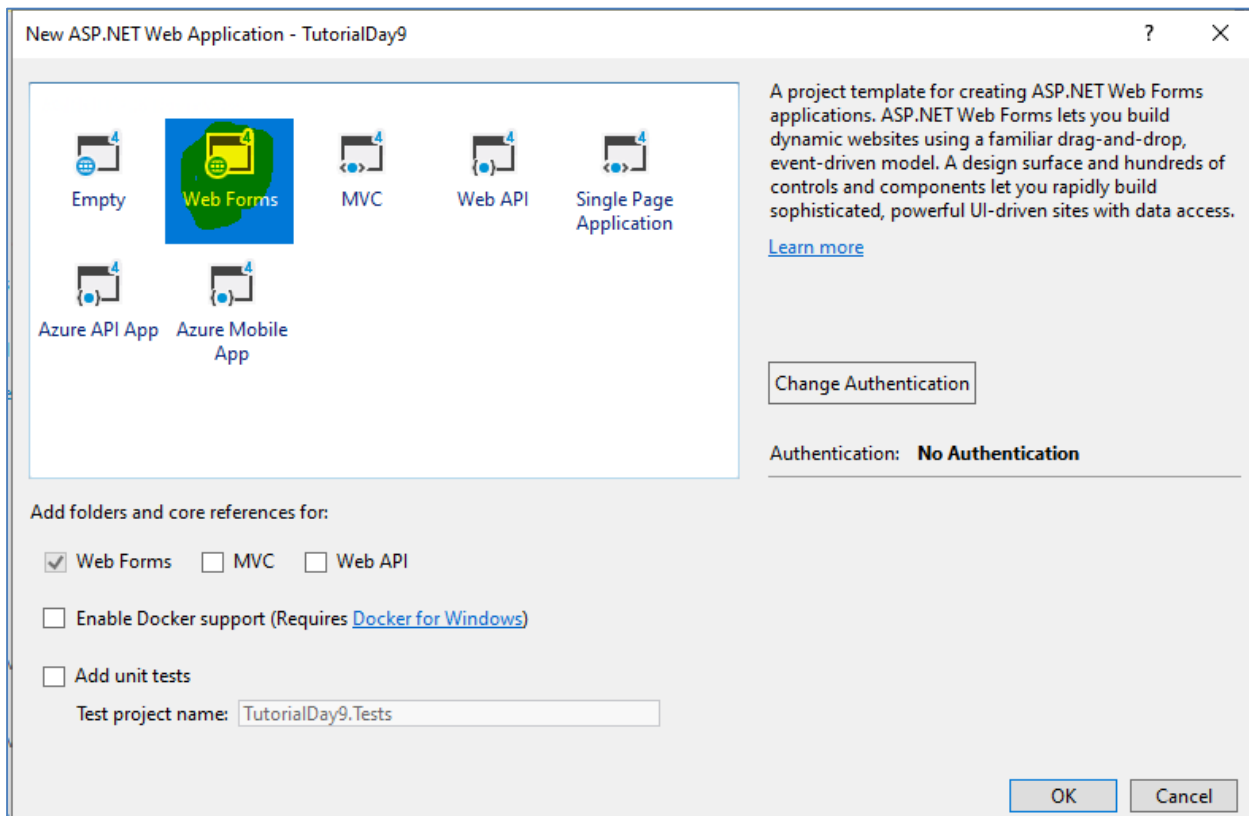
Master page provides the layout and functionality to the other pages. Creating a master page in ASP.NET is very easy. Let's start creating master page step by step.

Step 1: Open new project in visual studio

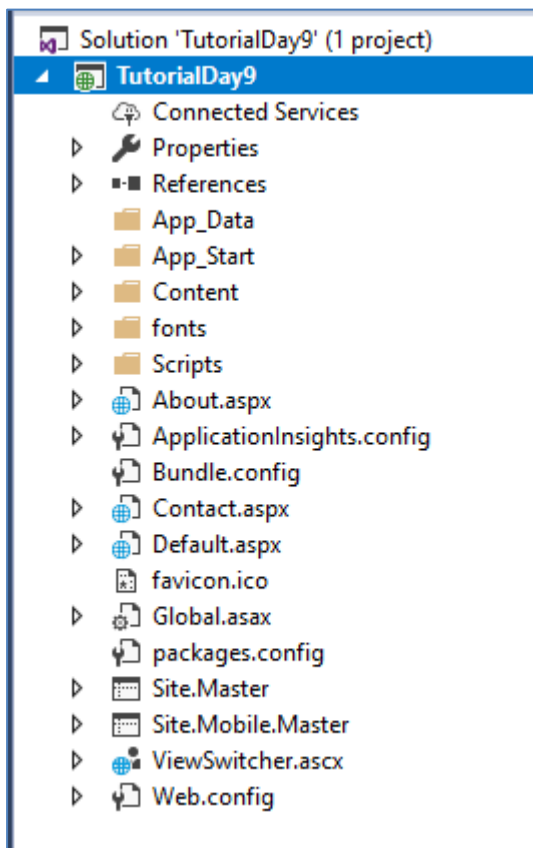
New project->Installed->Web->ASP.NET Web Application (shown in the picture),



After clicking OK button in the Window, select Empty (shown in the picture),



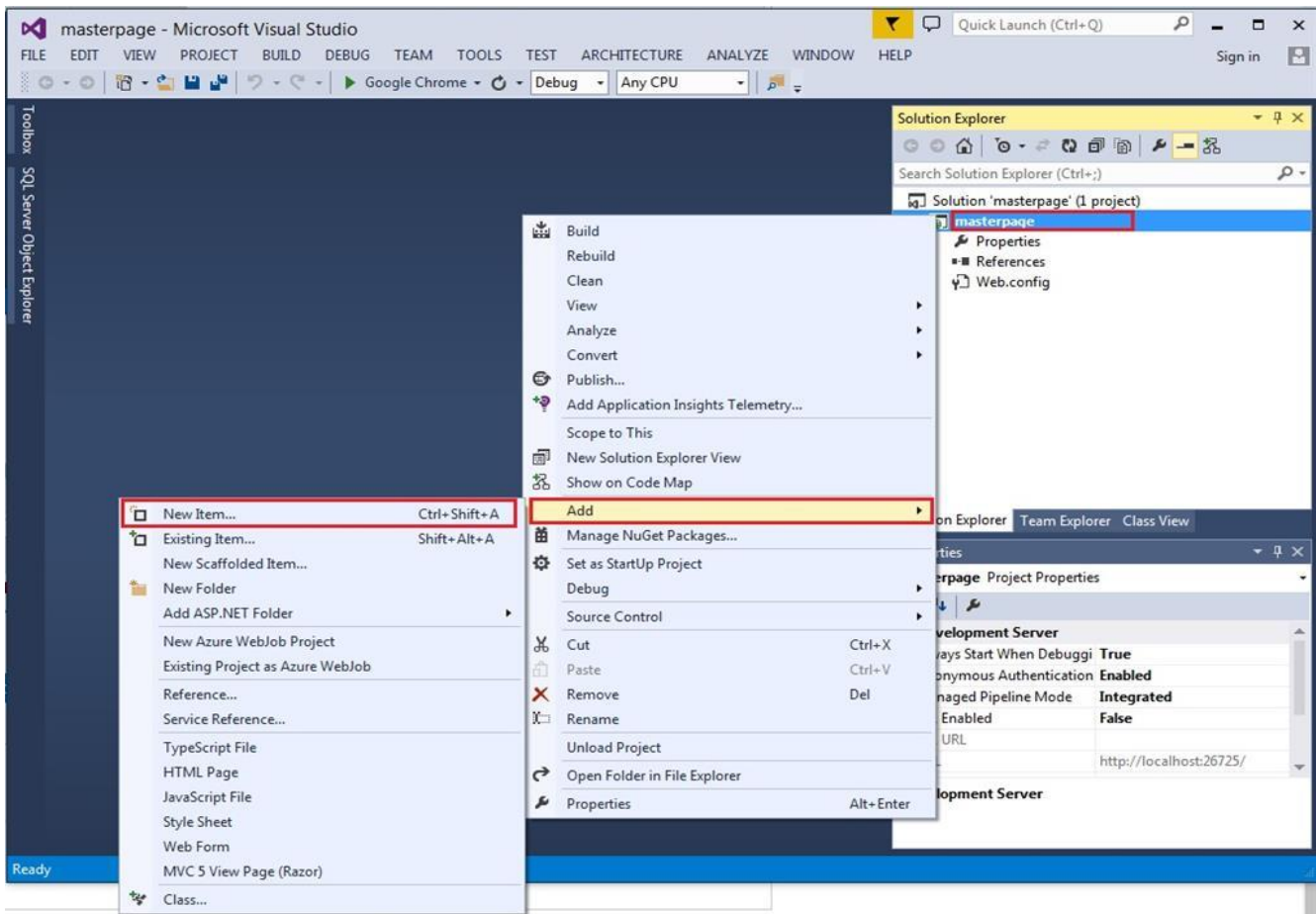
After clicking OK button, project opens (shown in the picture),



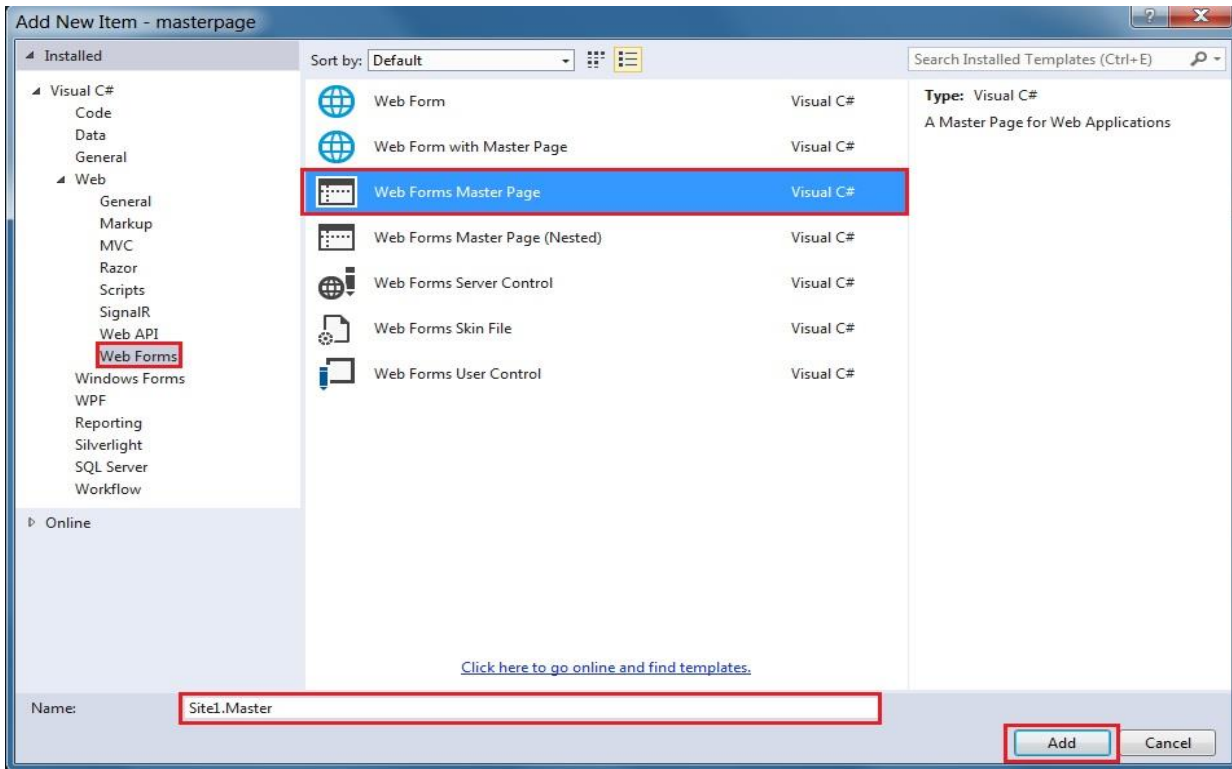
Step 2: Add new file in to our project.

Add the master page into our project.

Right click Project->Add->New item (shown in the picture),

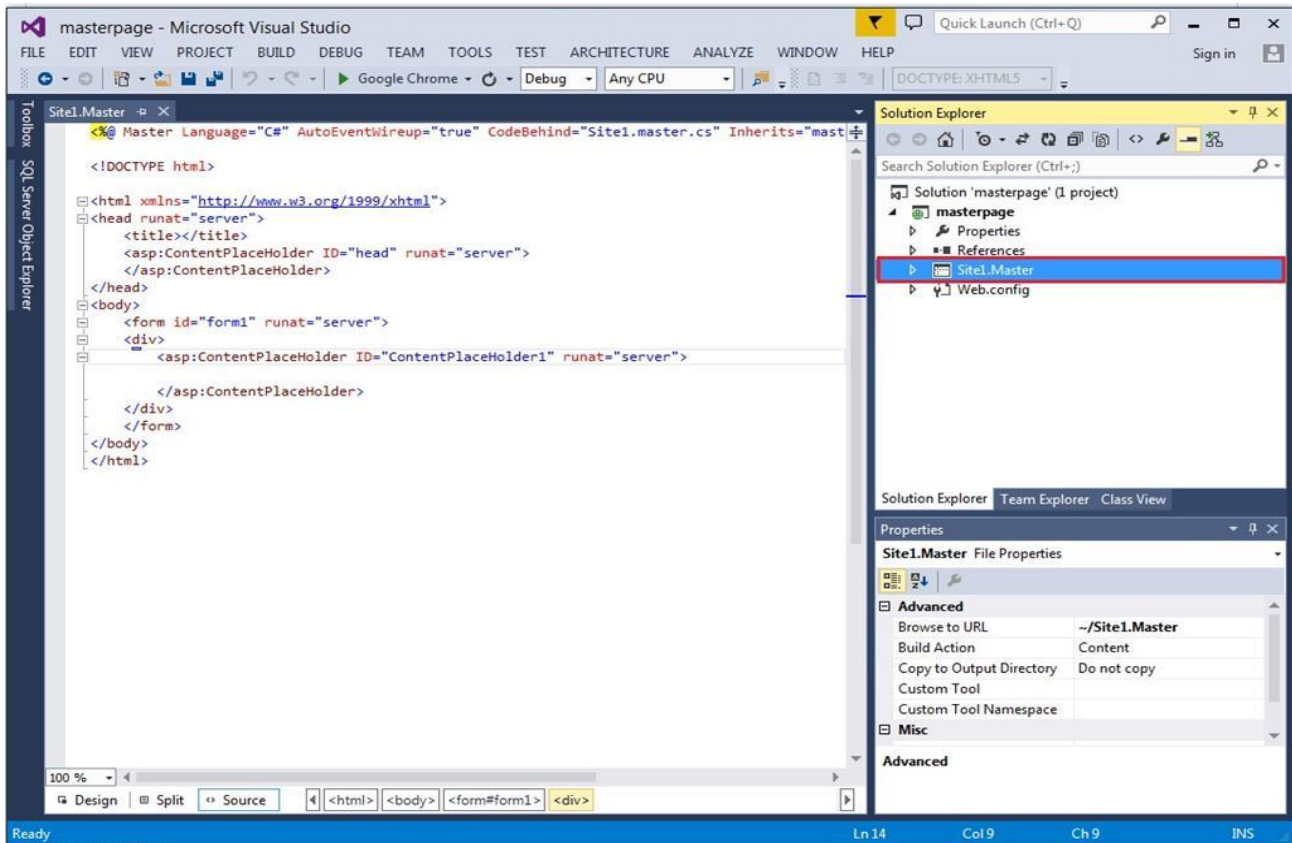


After clicking on new item, Window will open, select Web Form->Web Forms Master Page (shown in the picture),



After clicking the add button, master page 'site1.master' adds to our project.

Click on site1.master into Solution Explorer (shown in the picture),



Step 3: Design the master page, using HTML.

HTML code of my master page is,

```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site1.master.cs" Inherits="masterpage.Site1" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>c# corner</title>
    <link href="css/my.css" rel="stylesheet" />
    <asp:ContentPlaceholder ID="head" runat="server">
    </asp:ContentPlaceholder>
</head>
<body>
    <!DOCTYPE html>
<html>
<head>
    <title>my layout</title>
    <link rel="stylesheet" type="text/css" href="my.css">
</head>
<body>
<header id="header">
<h1>c# corner</h1>
</header>
<nav id="nav">
    <ul>
        <li><a href="home.aspx">Home</a></li>
```

```

        <li><a href="#">About</a></li>
        <li><a href="#">Article</a></li>
        <li><a href="#">Contact</a></li>
    </ul>
</nav>
<aside id="side">
    <h1>news</h1>
    <a href="#"><p>creating html website</p></a>
    <a href="#"><p>learn css</p></a>
    <a href="#">learn c#</a>
</aside>

    <div id="con">
        <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">

        </asp:ContentPlaceHolder>
    </div>

<footer id="footer">
    copyright @c# corner
</footer>
</body>
</html>
    <form id="form1" runat="server">

    </form>
</body>
</html>

```

CSS Code

```

#header{
    color: #247BA0;
    text-align: center;
    font-size: 20px;
}
#nav{
    background-color: #FF1654;
    padding: 5px;
}
ul{
    list-style-type: none;
}
li a {
    color: #F1FAEE;
    font-size: 30px;
    column-width: 5%;
}
li
{
    display: inline;
    padding-left: 2px;
    column-width: 20px;
}

```

```

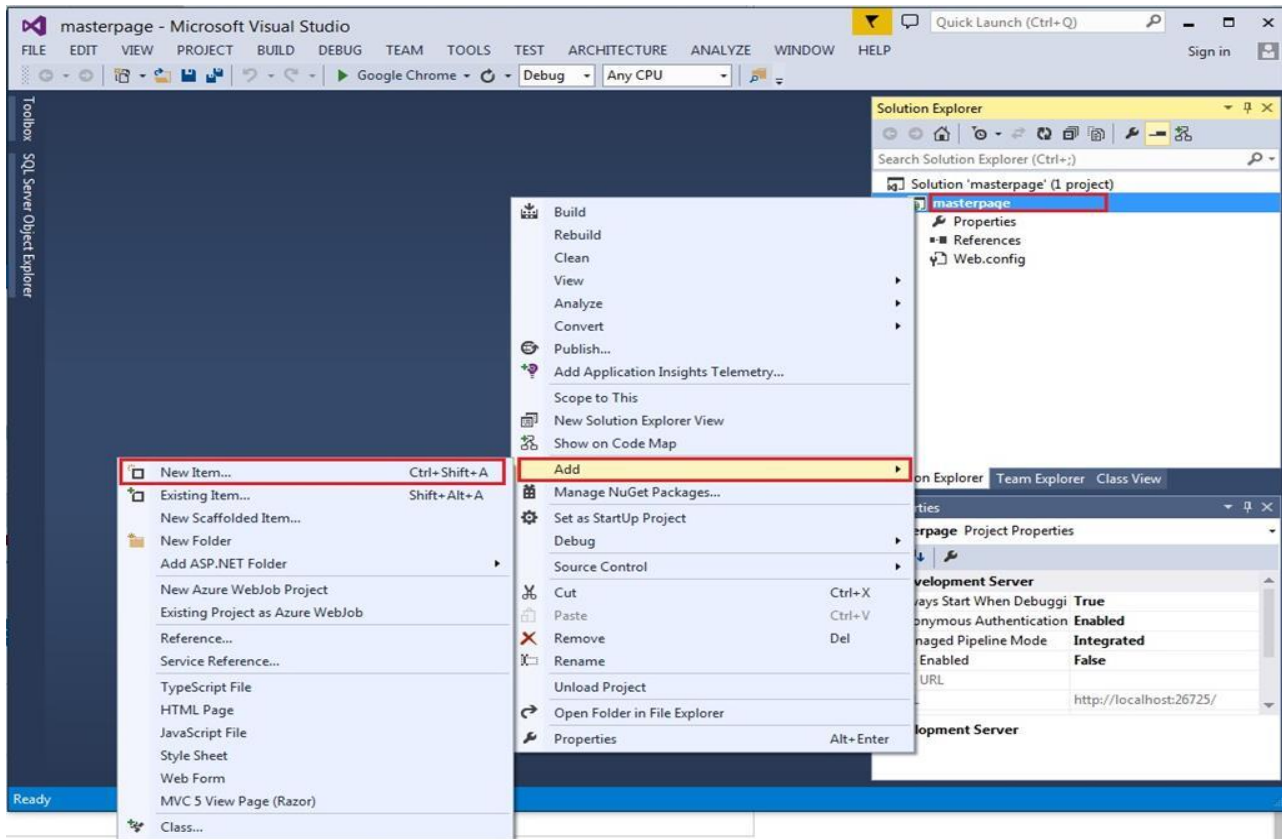
    }
    a{
text-decoration: none;
margin-left:20px
    }
    li a:hover{
background-color: #F3FFBD;
color: #FF1654;
padding:1%;
    }
    #side{
text-align: center;
float: right;
width: 15%;
padding-bottom: 79%;
background-color: #F1FAEE;
    }
    #article{
background-color: #EEF5DB;
padding: 10px;
padding-bottom: 75%;
    }
    #footer{
background-color: #C7EFCF;
text-align:center;
padding-bottom: 5%;
font-size: 20px;
    }
}
#con{
border:double;
border-color:burlywood;
}

```

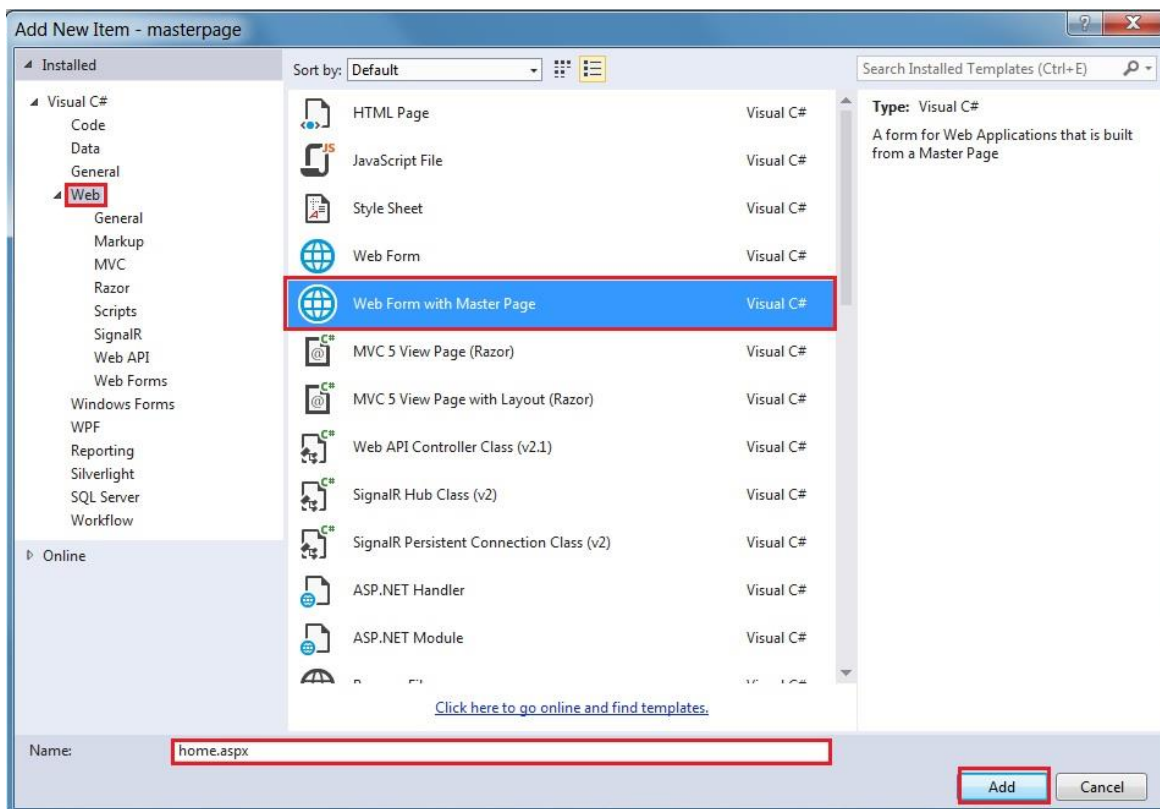
Our master page is designed. Move to the next step.

Step 4: Add web form in to our project.

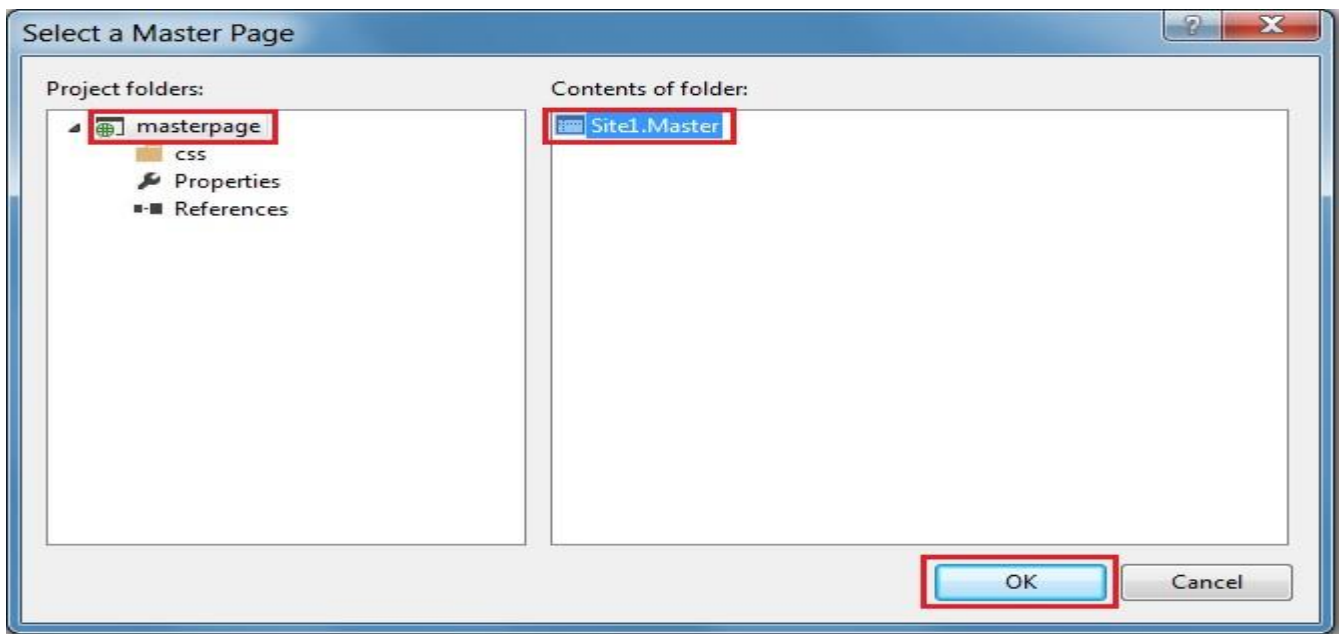
Right click on the project->Add->New item (shown in the picture),



Select Web form with the master page.



After clicking on that, add the button Window, open the selected masterpage->site1.master and click OK.



Now, design our homepage.

Here, we write home page only,

Home.aspx

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Site1.Master" AutoEventWireup="true" CodeBehind="home.aspx.cs" Inherits="masterpage.home" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <h1>Home page</h1>
</asp:Content>
```

Finally, our Master page is created; build and run the project.

Aspt.Net Page Life Cycle

In Asp.Net page life cycle has below events

Prenit:

- Here you can check for the **IsPostBack** property to determine whether this is the first time the page is being processed.
- Create or recreate dynamic controls.
- Set master page dynamically.
- Set the **Theme** property dynamically.
- Read or set profile property values.
- If Request is postback:
- The values of the controls have not yet been restored from view state.
- If you set control property at this stage, its value might be overwritten in the next event.

Init:

- In the **Init** event of the individual controls occurs first, later the **Init** event of the Page takes place.
- This event is used to initialize control properties.

InitComplete:

- Tracking of the ViewState is turned on in this event.

- Any changes made to the ViewState in this event are persisted even after the next postback.

PreLoad:

- This event processes the postback data that is included with the request.

Load:

- In this event the **Page** object calls the **OnLoad** method on the **Page** object itself, later the **OnLoad** method of the controls is called.
- Thus **Load** event of the individual controls occurs after the **Load** event of the page.

ControlEvents:

- This event is used to handle specific control events such as a **Button** control's **Click** event or a **TextBox** control's **TextChanged** event.
- In case of postback:
- If the page contains validator controls, the **Page.IsValid** property and the validation of the controls takes place before the firing of individual control events.

LoadComplete:

- This event occurs after the event handling stage.
- This event is used for tasks such as loading all other controls on the page.

PreRender:

- In this event the **PreRender** event of the page is called first and later for the child control.
- Usage:
- This method is used to make final changes to the controls on the page like assigning the **DataSourceId** and calling the **DataBind** method.

PreRenderComplete:

- This event is raised after each control's **PreRender** property is completed.

SaveStateComplete:

- This is raised after the control state and view state have been saved for the page and for all controls.

RenderComplete:

- The page object calls this method on each control which is present on the page.
- This method writes the control's markup to send it to the browser.

Unload:

- This event is raised for each control and then for the **Page** object.
- Usage:
- Use this event in controls for final cleanup work, such as closing open database connections, closing open files, etc.

Request, Response and Server Objects

Server Object

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

Properties and Methods of the **Server** object

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

The following table provides a list of the properties:

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

The following table provides a list of some important methods:

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

Request Object

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

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

Properties and Methods of the Request Object

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

Property	Description
AcceptTypes	Gets a string array of client-supported MIME accept types.
ApplicationPath	Gets the ASP.NET application's virtual application root path on the server.
Browser	Gets or sets information about the requesting client's browser capabilities.
ContentEncoding	Gets or sets the character set of the entity-body.
ContentLength	Specifies the length, in bytes, of content sent by the client.
ContentType	Gets or sets the MIME content type of the incoming request.
Cookies	Gets a collection of cookies sent by the client.
FilePath	Gets the virtual path of the current request.
Files	Gets the collection of files uploaded by the client, in multipart MIME format.
Form	Gets a collection of form variables.
Headers	Gets a collection of HTTP headers.
HttpMethod	Gets the HTTP data transfer method (such as GET, POST, or HEAD) used by the client.
InputStream	Gets the contents of the incoming HTTP entity body.
IsSecureConnection	Gets a value indicating whether the HTTP connection uses secure sockets (that is, HTTPS).
QueryString	Gets the collection of HTTP query string variables.
RawUrl	Gets the raw URL of the current request.
RequestType	Gets or sets the HTTP data transfer method (GET or POST) used by the client.
ServerVariables	Gets a collection of Web server variables.
TotalBytes	Gets the number of bytes in the current input stream.

Url	Gets information about the URL of the current request.
UrlReferrer	Gets information about the URL of the client's previous request that is linked to the current URL.
UserAgent	Gets the raw user agent string of the client browser.
UserHostAddress	Gets the IP host address of the remote client.
UserHostName	Gets the DNS name of the remote client.
UserLanguages	Gets a sorted string array of client language preferences.

The following table provides a list of some important methods:

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

Response Object

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

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

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

Properties and Methods of the Response Object

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

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

The following table provides a list of some important methods:

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

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

Example

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

The content file:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
    Inherits="server_side._Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

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

    <head runat="server">
        <title>Untitled Page</title>
    </head>

    <body>
        <form id="form1" runat="server">
            <div>

                Enter your name:
                <br />
                <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
                <asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="Submit" />
                <br />
                <asp:Label ID="Label1" runat="server"/>

            </div>
        </form>
    </body>

</html>
```

The code behind Button1_Click:

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

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

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

Get & Post Method

The Get/Post methods are used to send client information to web server. The methods are specified in inside form element using method attribute.

Syntax:

```
<form method="get|post">
```

The method attribute uses either get or post. Default is GET method.

GET method:

This method appends form data to page request URL with key value pairs called as query string. Page request URL and the form data information (query string) are separated by the ? Character.

- Get method is restricted to send up to 1024 characters only.
- Never use Get method to send sensitive information like password, login information to web server.
- Form data support only ASCII characters. Cannot used to send binary information like word document, text file and image data to server.
- We can access query string in ASP.NET using Request.QueryString["key1"]

Example: Usage of Get method In ASP.NET/HTML:

In the below example we will see how GET method passed data from login.html (client) to login.aspx page (server).

Login.html

```
<html>
<body>
  <form method="get" action="Login.aspx">
    User Name: <input id="txtuserName" type="text" name="username" />
    <input id="btnSubmit" type="submit" value="Submit data using GET" />
  </form>
</body>
</html>
```

Output:

User Name:_esspl

Login.aspx

```
<html>
<body>
  <form id="form1" runat="server">
    <div>
      Welcome <b><% Response.Write(Request.QueryString["username"].ToString()); %></b>
    </div>
  </form>
</body>
</html>
```


Output:

Welcome esspl.

Output Explanation: Login.html has username data which is sent using get method to login.aspx (server).

In the output of second screen shot we can see there is key (username) value(esspl) data as query string in the URL which is submitted to login.aspx page and the output(Welcome esspl) is shown in login.aspx page.

Post Method:

- POST method transfers information over HTTP headers.
- Data transfer through HTTP headers so using secure HTTP protocol we can make sure that data is secured.
- No restriction on sending data size via Post and also we can send binary data or ASCII information using POST method.
- We can access form data in ASP.NET using Request.Form["key1"]

Example: Usage of Get method In ASP.NET/HTML:

In the below example we will see how POST method passed data from login.html (client) to login.aspx page (server).

Login.html

```
<html>
<body>
<form method="post" action="Login.aspx">
  User Name: <input id="txtuserName" type="text" name="username" />
  <input id="btnSubmit" type="submit" value="Submit data using POST" />
</form>
</body>
</html>
```

Output:

User Name: esspl

Login.aspx

```
<html>
<body>
  <form id="form1" runat="server">
    <div>
      Welcome <b><% Response.Write(Request.Form["username"].ToString()); %></b>
    </div>
  </form>
</body>
</html>
```

output:

Welcome esspl

Output Explanation: Login.html has username data which is sent using post method to login.aspx (server). The output (Welcome esspl) is shown in login.aspx page.

Usage of Post method:

- If the form data is large then use POST method because GET method cannot handle long URL's
- Form data contains Non-ASCII characters use POST because GET doesn't support it.

State Management

State management means to preserve state of a control, web page, object/data, and user in the application explicitly because all ASP.NET web applications are stateless, i.e., by default, for each page posted to the server, the state of controls is lost. Nowadays all web apps demand a high level of state management from control to application level.

There are two types of state management techniques: client side and server side.

Client side

1. Hidden Field
2. View State
3. Cookies
4. Control State
5. Query Strings

Server side

1. Session
2. Application

View State

View state is another client side state management mechanism provided by ASP.NET to store user's data, i.e., sometimes the user needs to preserve data temporarily after a post back, then the view state is the preferred way for doing it. It stores data in the generated HTML using hidden field not on the server.

View State provides page level state management i.e., as long as the user is on the current page, state is available and the user redirects to the next page and the current page state is lost. View State can store any type of data because it is object type but it is preferable not to store a complex type of data due to the need for serialization and deserialization on each post back. View state is enabled by default for all server side controls of ASP.NET with a property `EnableViewState` set to `true`.

Let us see how ViewState is used with the help of the following example. In the example we try to save the number of postbacks on button click.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (IsPostBack)
    {
        if (ViewState["count"] != null)
        {
            int ViewstateVal = Convert.ToInt32(ViewState["count"]) + 1;
            Label1.Text = ViewstateVal.ToString();
            ViewState["count"] = ViewstateVal.ToString();
        }
        else
        {

```

```

        ViewState["count"] = "1";
    }
}
protected void Button1_Click(object sender, EventArgs e)
{
    Label1.Text=ViewState["count"].ToString();
}

```

Session

Session management is a very strong technique to maintain state. Generally session is used to store user's information and/or uniquely identify a user (or say browser). The server maintains the state of user information by using a session ID. When users makes a request without a session ID, ASP.NET creates a session ID and sends it with every request and response to the same user.

How to get and set value in Session:

```

Session["Count"] = Convert.ToInt32(Session["Count"]) + 1; //Set Value to The Session
Label2.Text = Session["Count"].ToString(); //Get Value from the Session

```

Let us see an example where we save the count of button clicks in a session, and save the “number of redirects to the same page” button click in a query string. Here I have set the expiry to 10 minutes. After starting the application, the application variable exists till the end of the application. A session variable will expire after 10 minutes (if it is idle). A query string contains the value in URL so it won't depend on the user idle time and could be used by the server anytime it is passed with a request

Session Events in ASP.NET

To manage a session, ASP.NET provides two events: session_start and session_end that is written in a special file called Global.asax in the root directory of the project.

Session_Start: The Session_start event is raised every time a new user makes a request without a session ID, i.e., new browser accesses the application, then a session_start event raised. Let's see the Global.asax file.

```

void Session_Start(object sender, EventArgs e)
{
    Session["Count"] = 0; // Code that runs when a new session is started
}

```

Session_End: The Session_End event is raised when session ends either because of a time out expiry or explicitly by using Session.Abandon(). The Session_End event is raised only in the case of In proc mode not in the state server and SQL Server modes.

There are four session storage mechanisms provided by ASP.NET:

- In Proc mode
- State Server mode
- SQL Server mode
- Custom mode

In Process mode: In proc mode is the default mode provided by ASP.NET. In this mode, session values are stored in the web server's memory (in IIS). If there are more than one IIS servers then session values are stored in each server separately on which request has been made. Since the session values are stored in server, whenever server is restarted the session values will be lost.

```
<configuration>
<sessionstate mode="InProc" cookieless="false" timeout="10"

stateConnectionString="tcpip=127.0.0.1:80808"

sqlConnectionString="Data Source=.\SqlDataSource;User ID=userid;Password=password"/>
</configuration>
```

In State Server mode: This mode could store session in the web server but out of the application pool. But usually if this mode is used there will be a separate server for storing sessions, i.e., stateServer. The benefit is that when IIS restarts the session is available. It stores session in a separate Windows service. For State server session mode, we have to configure it explicitly in the web config file and start the aspnet_state service.

```
<configuration><sessionstate mode="stateserver" cookieless="false"

timeout="10" stateConnectionString="tcpip=127.0.0.1:42424"

sqlConnectionString="Data Source=.\SqlDataSource;User ID=userid;Password=password"/> </configuration>
```

In SQL Server mode: Session is stored in a SQL Server database. This kind of session mode is also separate from IIS, i.e., session is available even after restarting the IIS server. This mode is highly secure and reliable but also has a disadvantage that there is overhead from serialization and deserialization of session data. This mode should be used when reliability is more important than performance.

```
<configuration>
<sessionstate mode="sqlserver" cookieless="false" timeout="10"

stateConnectionString="tcpip=127.0.0.1:4 2424"

sqlConnectionString="Data Source=.\SqlDataSource;User ID=userid;Password=password"/>
</configuration>
```

Custom Session mode: Generally we should prefer in proc state server mode or SQL Server mode but if you need to store session data using other than these techniques then ASP.NET provides a custom session mode. This way we have to maintain everything customized even generating session ID, data store, and also security.

Application State

Application state is a server side state management technique. The data stored in application state is common for all users of that particular ASP.NET application and can be accessed anywhere in the application. It is also called application level state management. Data stored in the application should be of small size.

How to get and set a value in the application object:

```
Application["Count"] = Convert.ToInt32(Application["Count"]) + 1; //Set Value to The Application Object
Label1.Text = Application["Count"].ToString(); //Get Value from the Application Object
```

Application events in ASP.NET

There are three types of events in ASP.NET. Application event is written in a special file called Global.asax. This file is not created by default, it is created explicitly by the developer in the root directory. An application can create more than one Global.asax file but only the root one is read by ASP.NET.

Application_start: The Application_Start event is raised when an app domain starts. When the first request is raised to an application then the Application_Start event is raised. Let's see the Global.asax file.

```
void Application_Start(object sender, EventArgs e)
{
    Application["Count"] = 0;
}
```

Application_Error: It is raised when an unhandled exception occurs, and we can manage the exception in this event.

Application_End: The Application_End event is raised just before an application domain ends because of any reason, may IIS server restarting or making some changes in an application cycle.

Cookies

Cookie is a small text file which is created by the client's browser and also stored on the client hard disk by the browser. It does not use server memory. Generally a cookie is used to identify users.

A cookie is a small file that stores user information. Whenever a user makes a request for a page the first time, the server creates a cookie and sends it to the client along with the requested page and the client browser receives that cookie and stores it on the client machine either permanently or temporarily (persistent or non persistence). The next time the user makes a request for the same site, either the same or another page, the browser checks the existence of the cookie for that site in the folder. If the cookie exists it sends a request with the same cookie, else that request is treated as a new request.

Types of Cookies

1. Persistence Cookie: Cookies which you can set an expiry date time are called persistence cookies. Persistence cookies are permanently stored till the time you set.

Let us see how to create persistence cookies. There are two ways, the first one is:

```
Response.Cookies["nameWithPCookies"].Value = "This is A Persistence Cookie";
Response.Cookies["nameWithPCookies"].Expires = DateTime.Now.AddSeconds(10);
```

And the second one is:

```
HttpCookie aCookieValPer = new HttpCookie("Persistence");
```

```
aCookieValPer.Value = "This is A Persistence Cookie";  
aCookieValPer.Expires = DateTime.Now.AddSeconds(10);  
Response.Cookies.Add(aCookieValPer);
```

2. Non-Persistence Cookie: Non persistence cookies are not permanently stored on the user client hard disk folder. It maintains user information as long as the user accesses the same browser. When user closes the browser the cookie will be discarded. Non Persistence cookies are useful for public computers.

Let us see how to create a non persistence cookies. There are two ways, the first one is:

```
Response.Cookies["nameWithNPCookies"].Value = "This is A Non Persistence Cookie";
```

And the second way is:

```
HttpCookie aCookieValNonPer = new HttpCookie("NonPersistence");  
aCookieValNonPer.Value = "This is A Non Persistence Cookie";  
Response.Cookies.Add(aCookieValNonPer);how to create cookie :
```

How to read a cookie:

```
if (Request.Cookies["NonPersistence"] != null)  
Label2.Text = Request.Cookies["NonPersistence"].Value;
```

Page Navigation

Page navigation in a web application is very important to understand since practically several times in a web application project we need to navigate from one page to another and return to the page of the original page as part of the functionality.

Navigation can cause data loss if it not properly handled. We do have many techniques to transfer data from one page to another but every technique has its own importance and benefits.

There are four type of page navigation in asp.net.

- Response.Redirect
- Server.Transfer
- Server.Exceute
- Cross page posting

Response.Redirect

This is one of the navigation techniques in ASP.NET to move from one web form to another.

It redirects a client to a new URL. Specifies the new URL and whether execution of the current page should terminate.

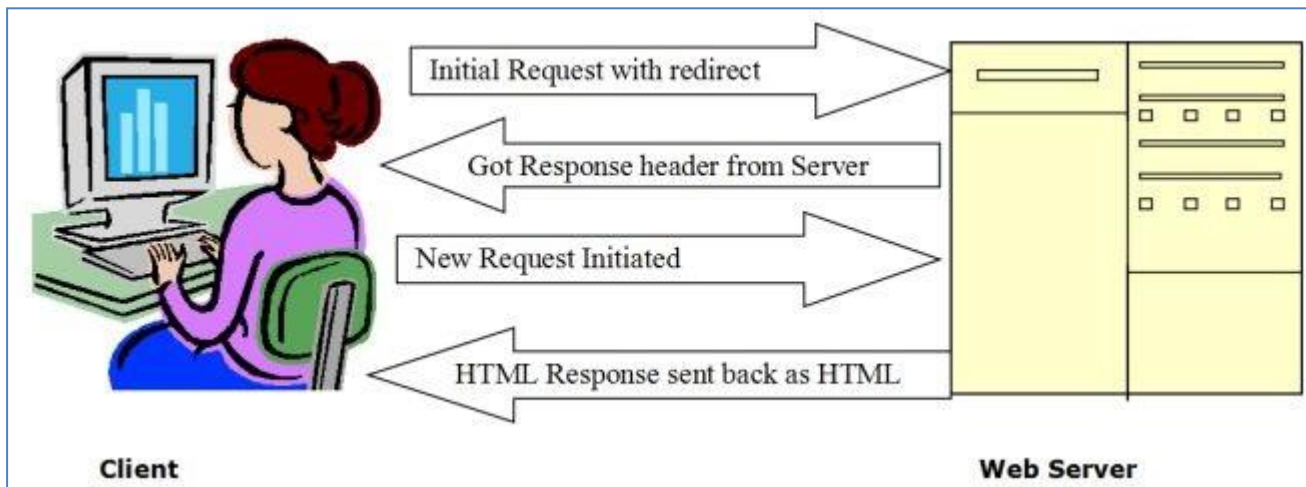
```
Response.Redirect("WebForm1.aspx");
```

Processing

when processing the client request when a web server encounters the code block Response.Redirect with the location URL to be redirected it will send back a response header to the client then the client initiates a

get request of the new page.

In this example we can see an extra round-trip to the web server being done when a `Response.Redirect` is used.



We can navigate to another URL using `Response.Redirect` either on the same server or a different server. Like in the following example we can go to the Google home page hosted on a different server using this technique.

```
Response.Redirect("https://www.google.co.in/?gfe_rd=cr&ei=BpszVKXmIsPN8gfBi4GABg&gws_rd=ssl");
```

It also maintains the history of an URL from where it has got navigated to, meaning we can return using the back button.

Important Sticky

- It redirects to a new redirected URL where it is redirected in the browser.
- It maintains the history and previous page is available with the back button.
- It redirects the user to a web page hosted on the same server or a different server.
- It has an additional round trip to the server that makes it a bit slower.

Server.Transfer

It is a navigation technique in an ASP.NET web application to move from one web form to another on the same server without changing the URL in an address bar.

This is the syntax for using this technique, basically it expects the URL of where you want to navigate to.

```
Server.Transfer("Webform2.aspx");
```

Cross Page Posting

ASP.Net 2.0 provides a feature known as Cross PagePostBack for a web form to post-back to a different web form (other than itself)

How to post to a different page

To set a web form to post back to a different web form, in the source web form, set the `PostBackURL` property of a control that implements `IButtonControl` (eg. `Button`, `ImageButton`, `LinkButton`) to the target web

form. When the user clicks on this button control, the web form is cross-posted to the target web form. No other settings or code is required in the source web form.

Access source page info within the posted page: FindControl Method

The target web form resulting from the cross-page postback provides a non-null PreviousPage property. This property represents the source page and provides reference to the source web form and its controls.

The controls on the source page can be accessed via the FindControl method on the object returned by the PreviousPage property of the target page.

```
protected void Page_Load(object sender, EventArgs e)
{
    ...
    TextBox txtStartDate = (TextBox) PreviousPage.FindControl("txtStartDate ");
    ...
}
```

At this point the target page does not have any knowledge of the source page. The PreviousPage property is of the type Page. For accessing controls using FindControl, the developer has to presume a certain structure in the source web form. This approach using FindControl has a few limitations. FindControl is dependent on the developer to provide the ids of the controls to access. The code will stop working if the control id is changed in the source web form. The FindControl method can retrieve controls only within the current container. If you need to access a control within another control, you need to first get a reference to the parent control.

Access source page info within the posted page: @PreviousPageType Directive

There is another more direct option to get access to the source page controls if the source page is pre-determined. The @PreviousPageType directive can be used in the target page to strongly type the source page. The directive specifies the source page using either the VirtualPath attribute or the TypeName attribute. The PreviousPage property then returns a strongly typed reference to the source page. It allows access to the public properties of the source page.

SourcePage.aspx:

```
<form runat="server" >
...
<asp:textbox runat="server" id="txtFirstName"/>
<asp:textbox runat="server" id="txtLastName"/>
<asp:button runat="server" id="btnViewReport" Text="View Report"PostBackURL=~/targetpage.aspx" />
...
public string FirstName
{
    get { return txtFirstName.Text; }
}
...
```

TargetPage.aspx

```
<%@ PreviousPageType VirtualPath="sourcepage.aspx" %>
```

```
string strFirstName;
strFirstName = PreviousPage.FirstName //Strongly Typed PreviousPage allows direct access to the
public properties of the source page.
```

Access source page info within the posted page: @Reference Directive

A third option to access the source page in a strongly typed fashion from the target page is to include an @Reference directive to the source page in the target page and then cast the PreviousPage property to the type of the source page.

Detect Cross Page PostBacks: IsCrossPagePostBack Property

When the source page cross-posts back to the target page, and the target page accesses the source page, the source page is reloaded in memory and goes through all the life cycle stages except the rendering. This version of the source page object is used by the target page to access information on the source page.

The IsCrossPagePostBack property in the source page indicates if it is being reloaded in memory in response to a PreviousPage reference from a target page.

1. Page A cross postback to Page B
2. Page B accesses the PreviousPage : Page A is reloaded in memory and the IsCrossPostBack property on this object has the value "true".

The *IsCrossPagePostBack* property can be used in the **source** page to prevent un-necessary processing from repeating when the page is being reloaded for the PreviousPage reference.

The *PreviousPage.IsCrossPagePostBack* property can be used to deduce if the current page has been loaded as a result of a cross page postback.

Validations

If the Source Page has any Validator controls, the source page will need to have valid input when it has cross posted to the target page. The target page can contain the validation check for *PreviousPage.IsValid* to catch invalid submissions to the source page.

User Controls

In daily development we use UserControls. Now the question is, what are UserControls and why do we use them.

In Software Development, everybody is talking about reusability. A UserControl also provides for reusability. In other words, we can create and edit in one thing and get results everywhere, wherever you use the item.

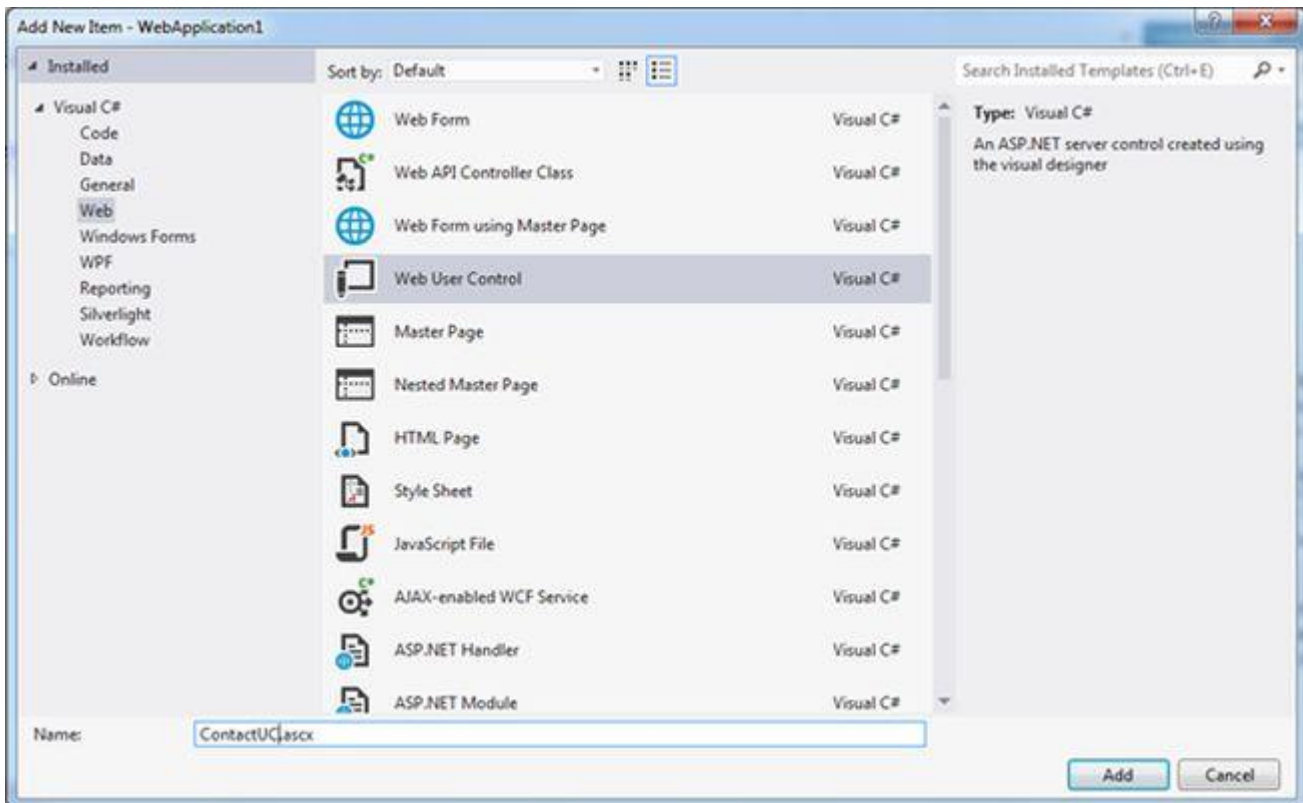
For example we have an e-mail form or a contact us form and I want to use it in 4 different pages in various parts. And if the client requirements change then I just make the changes in one place and automatically the changes will be done in all forms. So for this we can use a UserControl.

How to create a User Control

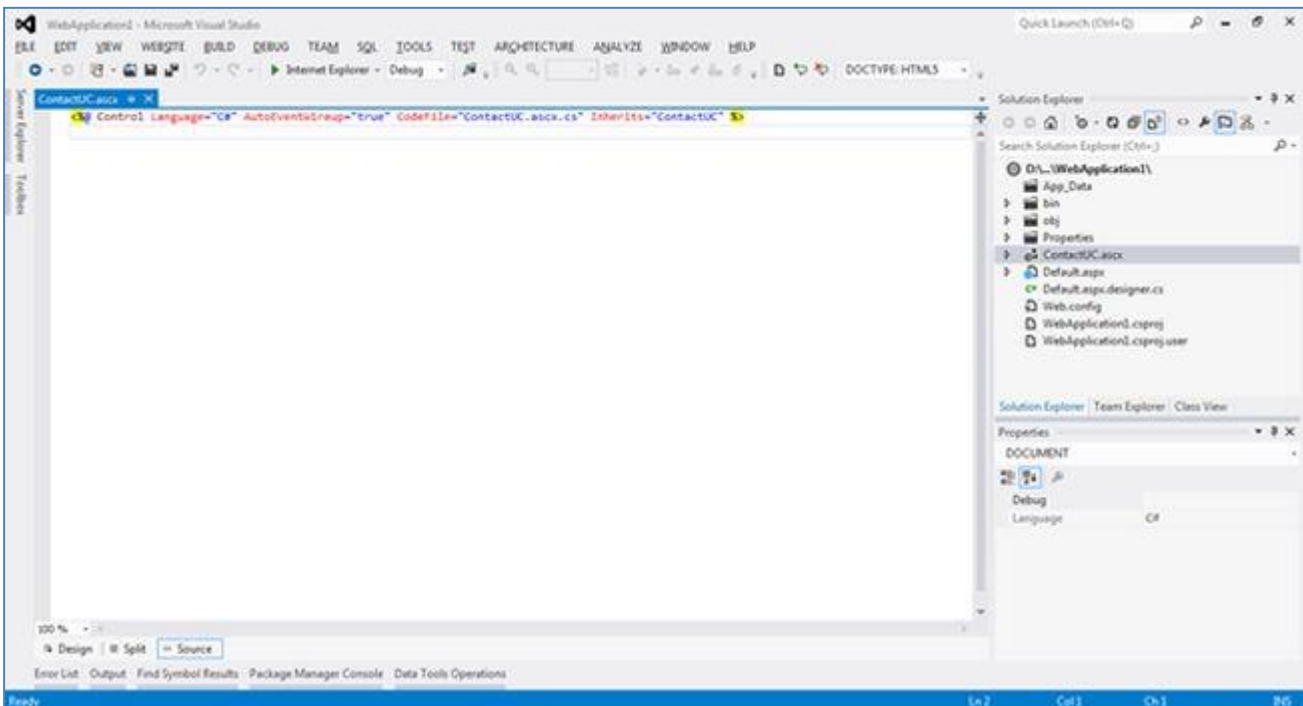
Have a look at the following example:

1. Open Visual Studio.
2. "File" -> "New" -> "Project..." then select ASP.NET Webform Application.
3. Add a new web form.

To create a new UserControl, in Solution Explorer, Add New Item, provide your File Name and click Add.



When you click on the Add Button the following screen will be shown.



ContactUC.ascx

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="ContactUC.ascx.cs"
Inherits="UserControlDemo.ContactUC" %>

<table>
  <tr>
    <td>
```

```

        <fieldset>
            <asp:Label ID="lblMessage" runat="server"></asp:Label>
        </fieldset>
    </td>
</tr>
<tr>
    <td>
        <asp:Label ID="lblName" runat="server" Text="Name"></asp:Label>
    </td>
    <td>
        <asp:TextBox ID="txtName" runat="server"></asp:TextBox>
    </td>
</tr>
<tr>
    <td>
        <asp:Label ID="lblEmail" runat="server" Text="Email"></asp:Label>
    </td>
    <td>
        <asp:TextBox ID="txtEmail" runat="server"></asp:TextBox>
    </td>
</tr>
<tr>
    <td>
        <asp:Button ID="btnSubmit" runat="server" Text="Submit" />
    </td>
</tr>
</table>

```

ContactUC.ascx.cs

```

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

namespace UserControlDemo
{
    public partial class ContactUC : System.Web.UI.UserControl
    {
        private string _header;
        public string Header
        {
            get { return _header; }
            set { _header = value; }
        }
        protected void Page_Load(object sender, EventArgs e)
        {
            lblMessage.Text = _header;
        }
    }
}

```

UcDemo.aspx

```

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="UcDemo.aspx.cs" Inheri
ts="UserControlDemo.UcDemo" %>
<%@ Register Src="~/ContactUC.ascx" TagPrefix="uc1" TagName="ContactUC" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <uc1:ContactUC runat="server" id="ContactUC"/>
        </div>
    </form>
</body>
</html>

```

Understand the code

A UserControl is nothing but a webpage where you can put your controls, write their events into a .cs file and use whenever you want. In the preceding example as you can see we have a UserControl where we want to create a contactus kind of page, where we have one lblMessage, where we use the UserControl property for header. Here we have 4 textboxes and one button, for inserting user info. If we want to write code for the submit button then we can write it here.

In ContactUC.ascx.cs, here we add a property and use this property in lblMessage.Text at the PageLoad event.

```

private string _header;
public string Header
{
    get { return _header; }
    set { _header = value; }
}
protected void Page_Load(object sender, EventArgs e)
{
    lblMessage.Text = _header;
}

```

Calling a UserControl in to a web page

When you drag a UserControl onto a page. You can see this code.

```

<%@ Register Src="~/ContactUC.ascx" TagPrefix="uc1" TagName="ContactUC" %>
<uc1:ContactUC runat="server" id="ContactUC" Header="User Contact Us Page" />

```

UcDemo.aspx

```

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="UcDemo.aspx.cs" Inheri
ts="UserControlDemo.UcDemo" %>
<%@ Register Src="~/ContactUC.ascx" TagPrefix="uc1" TagName="ContactUC" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">

```

```
<title></title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <uc1:ContactUC runat="server" id="ContactUC" Header="User Contact Us Page" />
    </div>
  </form>
</body>
</html>
```

You can change the header property depending on your requirements. If you want to add more properties to it, you can create more properties in your .cs file.

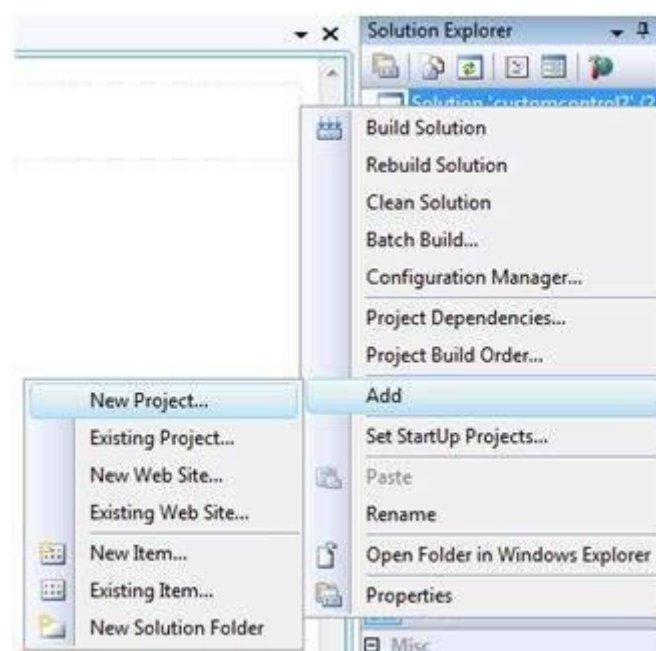
Custom Controls

Custom controls are deployed as individual assemblies. They are compiled into a Dynamic Link Library (DLL) and used as any other ASP.NET server control. They could be created in either of the following way:

- By deriving a custom control from an existing control
- By composing a new custom control combining two or more existing controls.
- By deriving from the base control class.

To understand the concept, let us create a custom control, which will simply render a text message on the browser. To create this control, take the following steps:

Create a new website. Right click the solution (not the project) at the top of the tree in the Solution Explorer.

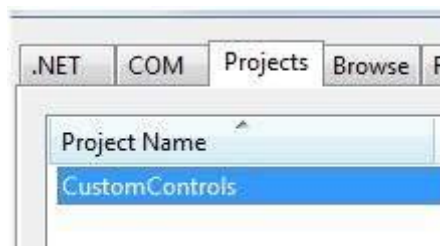


In the New Project dialog box, select ASP.NET Server Control from the project templates.



The above step adds a new project and creates a complete custom control to the solution, called ServerControl1. In this example, let us name the project CustomControls. To use this control, this must be added as a reference to the web site before registering it on a page. To add a reference to the existing project, right click on the project (not the solution), and click Add Reference.

Select the CustomControls project from the Projects tab of the Add Reference dialog box. The Solution Explorer should show the reference.



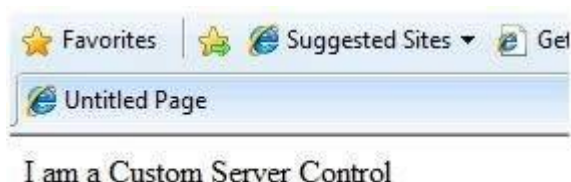
To use the control on a page, add the Register directive just below the @Page directive:

```
<%@ Register Assembly="CustomControls" Namespace="CustomControls" TagPrefix="ccs" %>
```

Further, you can use the control, similar to any other controls.

```
<form id="form1" runat="server">
  <div>
    <ccs:ServerControl1 runat="server" Text = "I am a Custom Server Control" />
  </div>
</form>
```

When executed, the Text property of the control is rendered on the browser as shown:



In the previous example, the value for the Text property of the custom control was set. ASP.NET added this property by default, when the control was created. The following code behind file of the control reveals this.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```

using System.Linq;
using System.Text;

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

namespace CustomControls
{
    [DefaultProperty("Text")]
    [ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1 >")]

    public class ServerControl1 : WebControl
    {
        [Bindable(true)]
        [Category("Appearance")]
        [DefaultValue("")]
        [Localizable(true)]

        public string Text
        {
            get
            {
                String s = (String)ViewState["Text"];
                return ((s == null) ? "[" + this.ID + "]" : s);
            }

            set
            {
                ViewState["Text"] = value;
            }
        }

        protected override void RenderContents(HtmlTextWriter output)
        {
            output.Write(Text);
        }
    }
}

```

The above code is automatically generated for a custom control. Events and methods could be added to the custom control class.

Example

Let us expand the previous custom control named SeverControl1. Let us give it a method named checkpalindrome, which gives it a power to check for palindromes.

Palindromes are words/literals that spell the same when reversed. For example, Malayalam, madam, saras, etc.

Extend the code for the custom control, which should look as:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;

```

```

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

namespace CustomControls
{
    [DefaultProperty("Text")]
    [ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1 >")]

    public class ServerControl1 : WebControl
    {
        [Bindable(true)]
        [Category("Appearance")]
        [DefaultValue("")]
        [Localizable(true)]

        public string Text
        {
            get
            {
                String s = (String)ViewState["Text"];
                return ((s == null) ? "[" + this.ID + "]" : s);
            }

            set
            {
                ViewState["Text"] = value;
            }
        }

        protected override void RenderContents(HtmlTextWriter output)
        {
            if (this.checkpanlindrome())
            {
                output.Write("This is a palindrome: <br />");
                output.Write("<FONT size=5 color=Blue>");
                output.Write("<B>");
                output.Write(Text);
                output.Write("</B>");
                output.Write("</FONT>");
            }
            else
            {
                output.Write("This is not a palindrome: <br />");
                output.Write("<FONT size=5 color=red>");
                output.Write("<B>");
                output.Write(Text);
                output.Write("</B>");
                output.Write("</FONT>");
            }
        }

        protected bool checkpanlindrome()
        {
            if (this.Text != null)
            {
                String str = this.Text;
                String strtoupper = Text.ToUpper();
                char[] rev = strtoupper.ToCharArray();
                Array.Reverse(rev);
                String strrev = new String(rev);
            }
        }
    }
}

```



```

        if (strtoupper == strrev)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    else
    {
        return false;
    }
}
}

```

When you change the code for the control, you must build the solution by clicking Build --> Build Solution, so that the changes are reflected in your project. Add a text box and a button control to the page, so that the user can provide a text, it is checked for palindrome, when the button is clicked.

```

<form id="form1" runat="server">
    <div>
        Enter a word:
        <br />
        <asp:TextBox ID="TextBox1" runat="server" style="width:198px"> </asp:TextBox>

        <br /> <br />

        <asp:Button ID="Button1" runat="server onclick="Button1_Click" Text="Check
        Palindrome" style="width:132px" />

        <br /> <br />

        <ccs:ServerControl1 ID="ServerControl11" runat="server" Text = "" />
    </div>
</form>

```

The Click event handler for the button simply copies the text from the text box to the text property of the custom control.

```

protected void Button1_Click(object sender, EventArgs e)
{
    this.ServerControl11.Text = this.TextBox1.Text;
}

```

When executed, the control successfully checks palindromes.

Enter a word:

madam

Check Palindrome

This is a palindrome:

madam

Observe the following:

(1) When you add a reference to the custom control, it is added to the toolbox and you can directly use it from the toolbox similar to any other control.



(2) The RenderContents method of the custom control class is overridden here, as you can add your own methods and events.

(3) The RenderContents method takes a parameter of HtmlTextWriter type, which is responsible for rendering on the browser.

Authentication and Authorization

Before proceeding ahead we need to understand four important vocabularies which you will see in this article again and again: - authentication, authorization, principal and identity. Let's first start with authentication and authorization. If you search in www.google.com for the dictionary meaning of authentication and authorization, you will land up with something below:-

Authentication: - prove genuineness

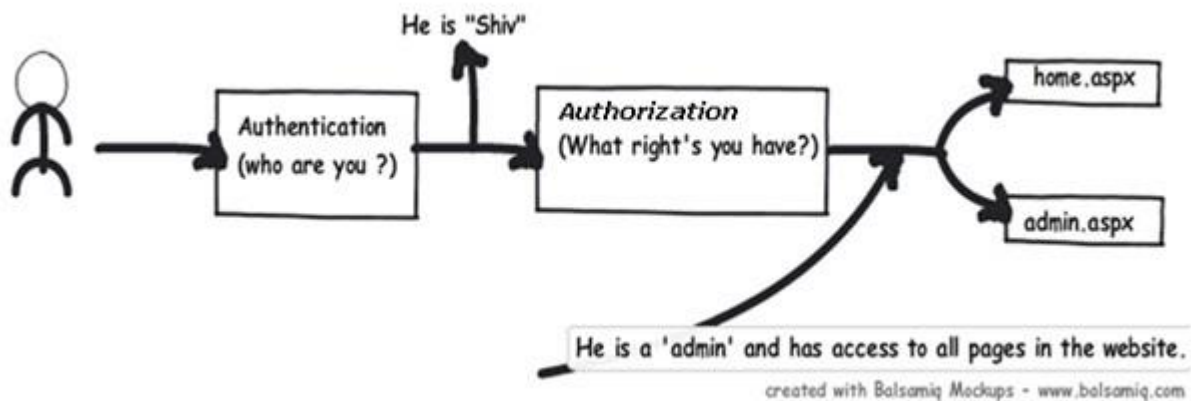
Authorization: - process of granting approval or permission on resources.

The same dictionary meaning applies to ASP.NET as well. In ASP.NET authentication means to identify the user or in other words its nothing but to validate that he exists in your database and he is the proper user. Authorization means does he have access to a particular resource on the IIS website. A resource can be an ASP.NET web page, media files (MP4, GIF, JPEG etc), compressed file (ZIP, RAR) etc.

So the first process which happens is authentication and then authorization. Below is a simple graphical representation of authentication and authorization. So when the user enters 'userid' and 'password' he is first authenticated and identified by the user name.

Now when the user starts accessing resources like pages, ASPDOTNETauthentication, videos etc, he is checked whether he has the necessary access for the resources. The process of identifying the rights for

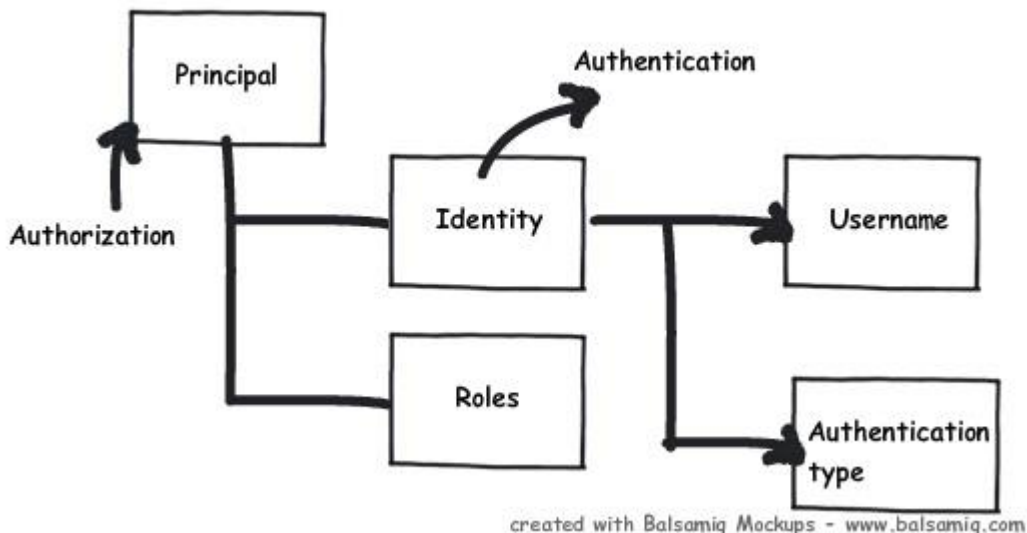
resources is termed as 'Authorization'.



To put it in simple words to identify "he is Shiv" is authentication and to identify that "Shiv is admin" is authorization.

Detecting authentication and authorization: - The principal and identity objects

At any moment of time if you want to know who the user is and what kind of authentication type he using you can use the identity object. If you want to know what kind of roles it's associated with then we need to use the principal object. In other words to get authentication details we need to the identity object and to know about authorization details of that identity we need the principal object.

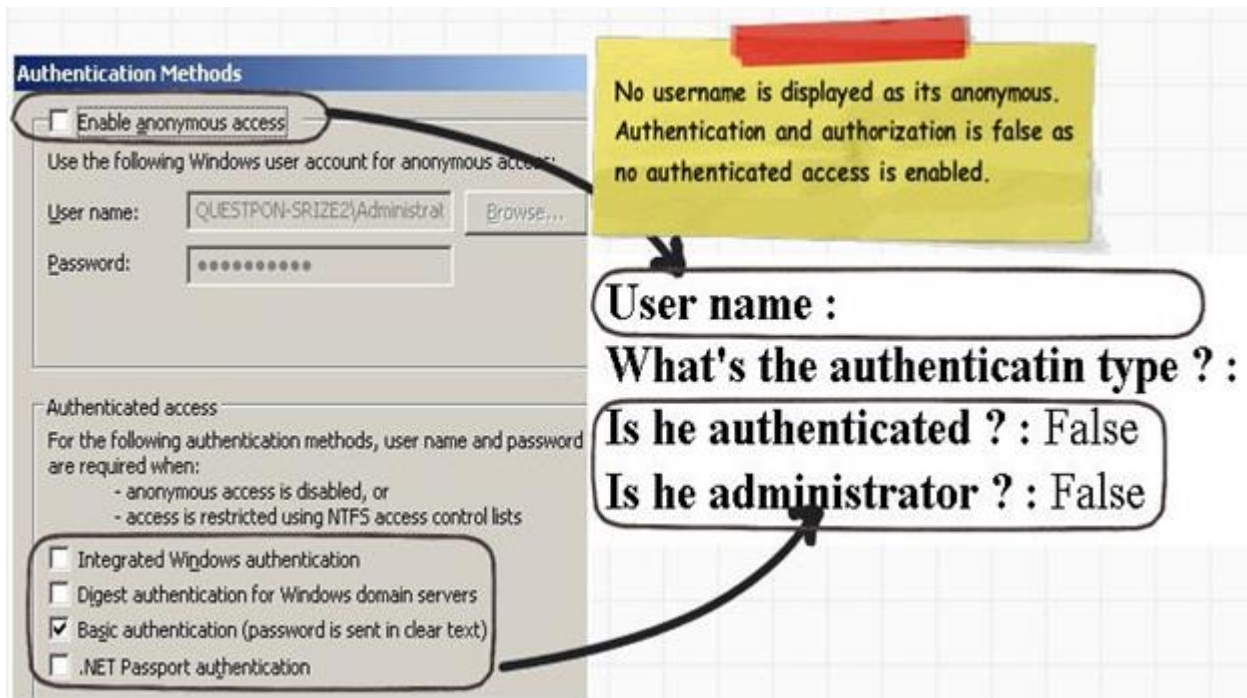


For instance below is a simple sample code which shows how to use identity and principal object to display name and check roles.

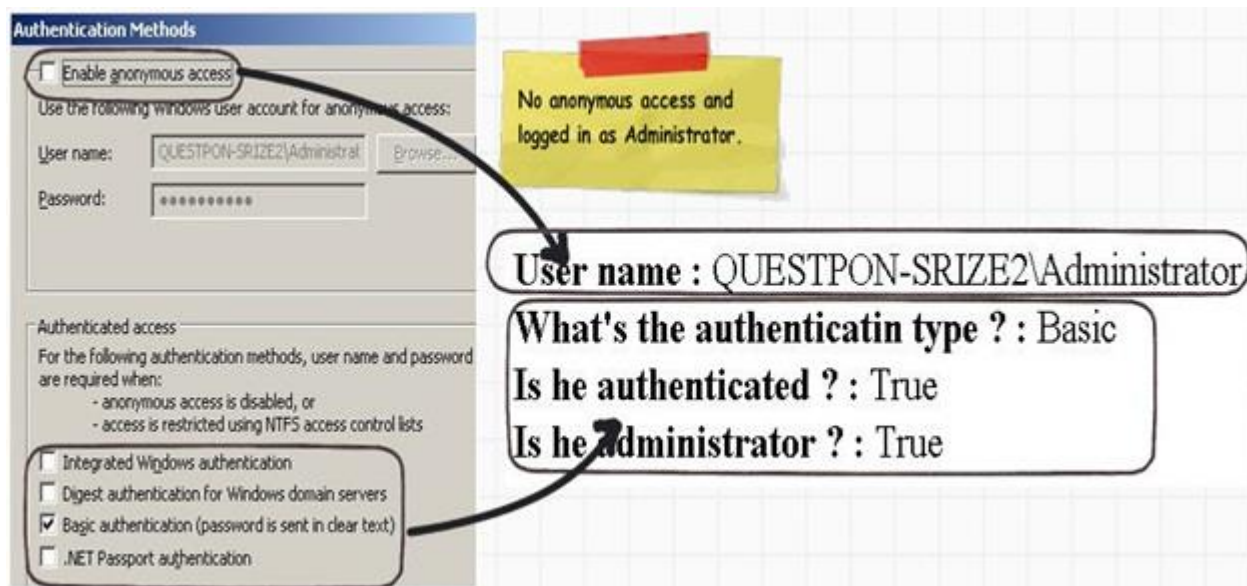
```
Response.Write(User.Identity.Name + "<br>");  
Response.Write(User.Identity.AuthenticationType + "<br>");  
Response.Write(User.Identity.IsAuthenticated + "<br>");
```

```
Response.Write(User.IsInRole("Administrators") + "<br>");
```

Now if you run this code in IIS under anonymous mode it will display no details as shown below.



If you run the above code in IIS using some authentication mode like one shown below "Basic authentication" it will show all the details as shown below.



Types of authentication and authorization in ASP.NET

There are three ways of doing authentication and authorization in ASP.NET:-

Windows authentication: - In this methodology ASP.NET web pages will use local windows users and groups to authenticate and authorize resources.

Forms Authentication: - This is a cookie based authentication where username and password are stored on client machines as cookie files or they are sent through URL for every request. Form-based authentication presents the user with an HTML-based Web page that prompts the user for credentials.

Passport authentication: - Passport authentication is based on the passport website provided by the Microsoft .So when user logs in with credentials it will be reached to the passport website (i.e. hotmail,devhood,windows live etc) where authentication will happen. If Authentication is successful it will return a token to your website.

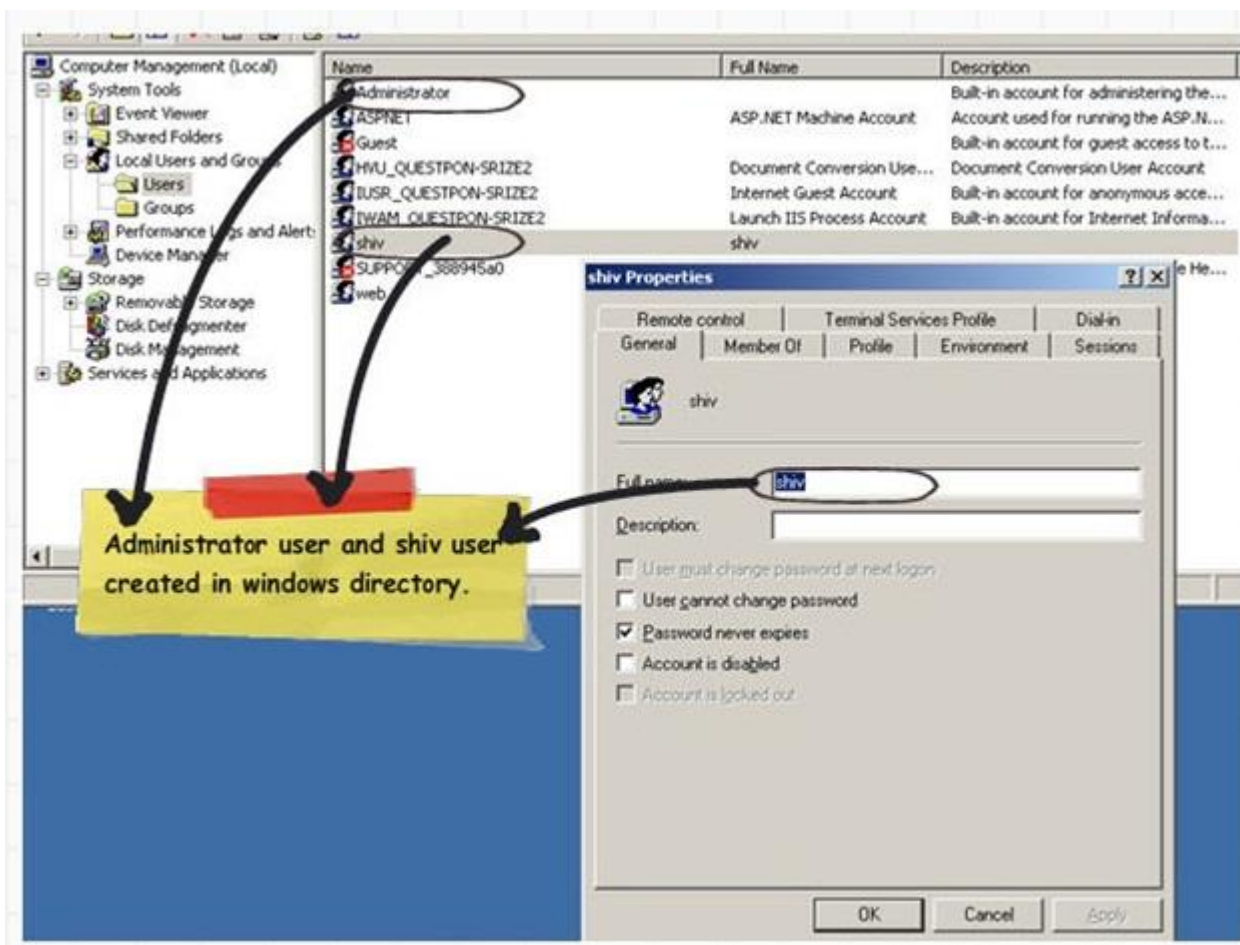
Anonymous access: - If you do not want any kind of authentication then you will go for Anonymous access.

GenericPrincipal and GenericIdentity objects represent users who have been authenticated using Forms authentication or other custom authentication mechanisms. With these objects, the role list is obtained in a custom manner, typically from a database.

FormsIdentity and PassportIdentity objects represent users who have been authenticated with Forms and Passport authentication respectively.

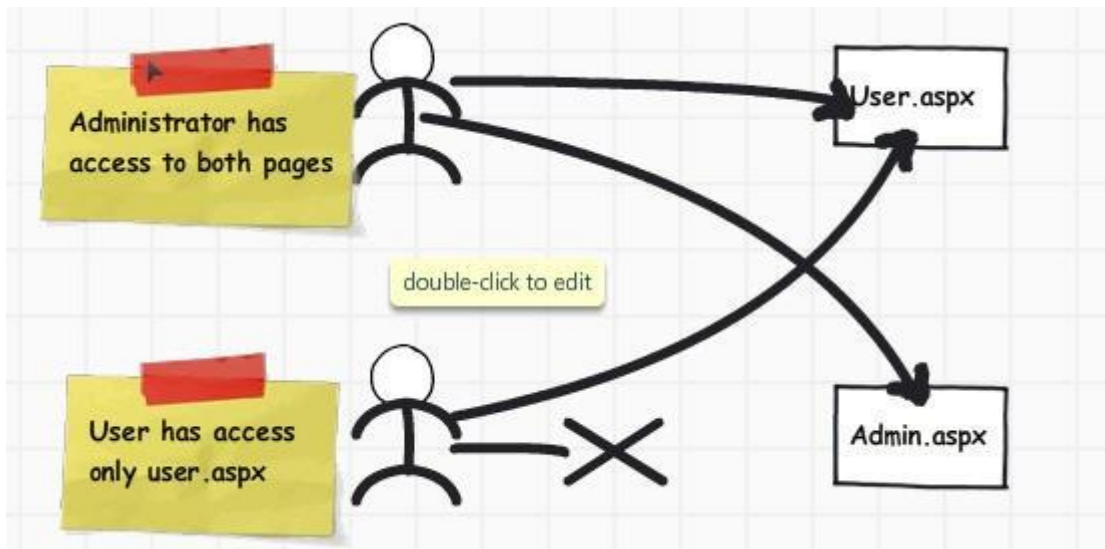
Windows Authentication

When you configure your ASP.NET application as windows authentication it will use local windows user and groups to do authentication and authorization for your ASP.NET pages. Below is a simple snap shot which shows my windows users and roles on my computer.



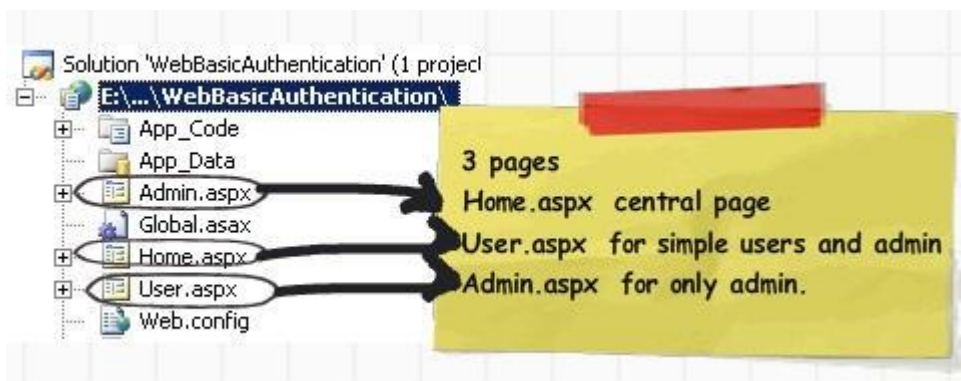
5 steps to enable authentication and authorization using Windows

We will do a small sample to get a grasp of how authentication and authorization works with windows. We will create 2 users one 'Administrator' and other a simple user with name 'Shiv'. We will create two simple ASPX pages 'User.aspx' page and 'Admin.aspx' page. 'Administrator' user will have access to both 'Admin.aspx' and 'User.aspx' page, while user 'Shiv' will only have access to 'User.aspx' page.



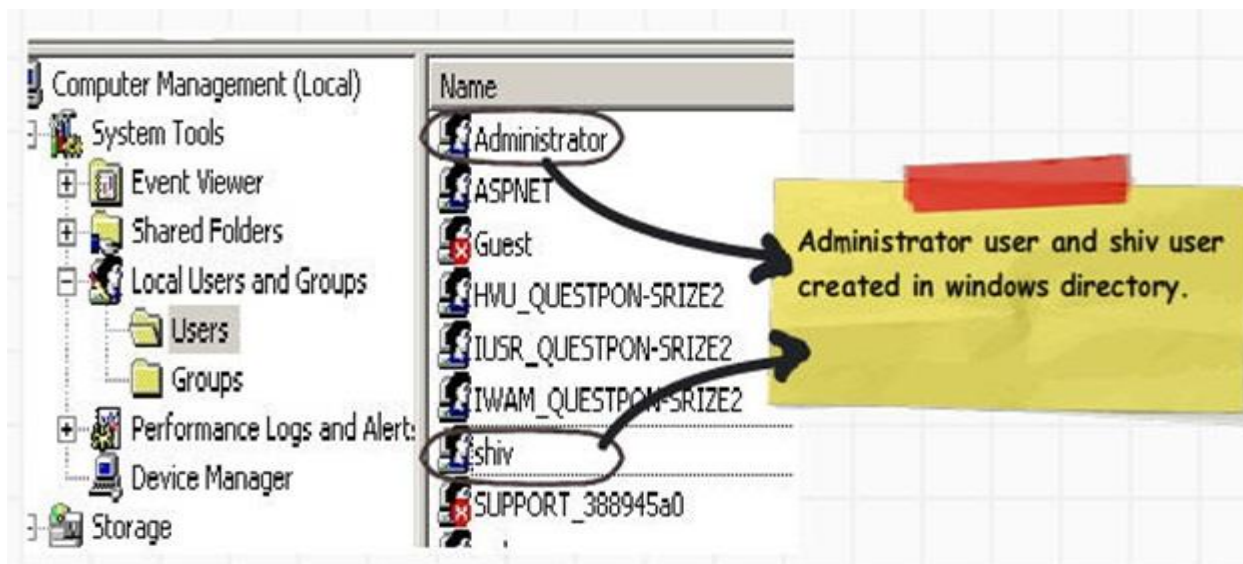
Step 1:- Creation of web site.

The next step is to create a simple web site with 3 pages (User.aspx, Admin.aspx and Home.aspx) as shown below.



Step 2:- Create user in the windows directory

The next step is we go to the windows directory and create two users. You can see in my computer we have 'Administrator' and 'Shiv'.



Step 3:- Setup the 'web.config' file

In 'web.config' file set the authentication mode to 'Windows' as shown in the below code snippets.

```
<authentication mode="Windows"/>
```

We also need to ensure that all users are denied except authorized users. The below code snippet inside the authorization tag that all users are denied. '?' indicates any

```
unknown user.
<authorization>
<deny users="?"/>
</authorization>
```

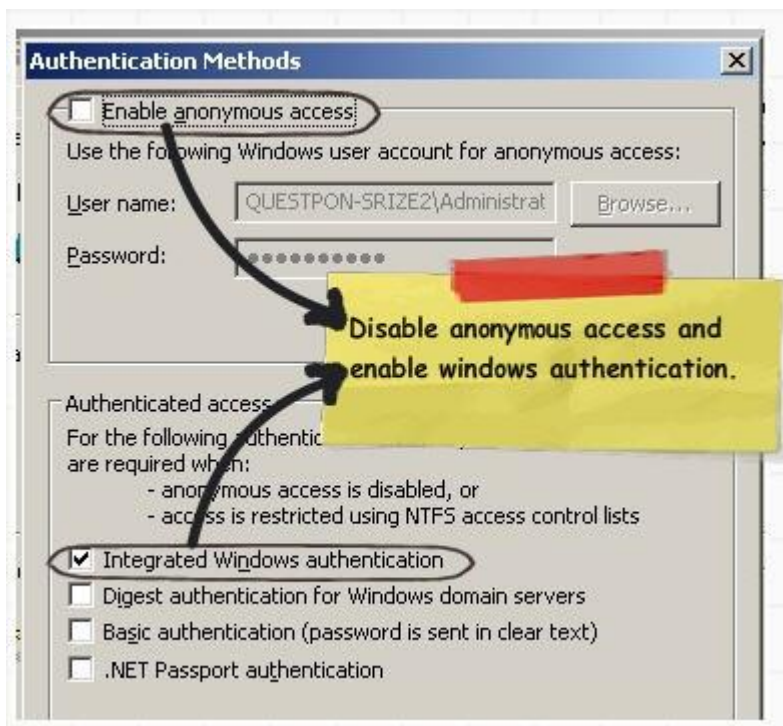
Step 4:- Setup authorization

We also need to specify the authorization part. We need to insert the below snippet in the 'web.config' file stating that only 'Administrator' users will have access to

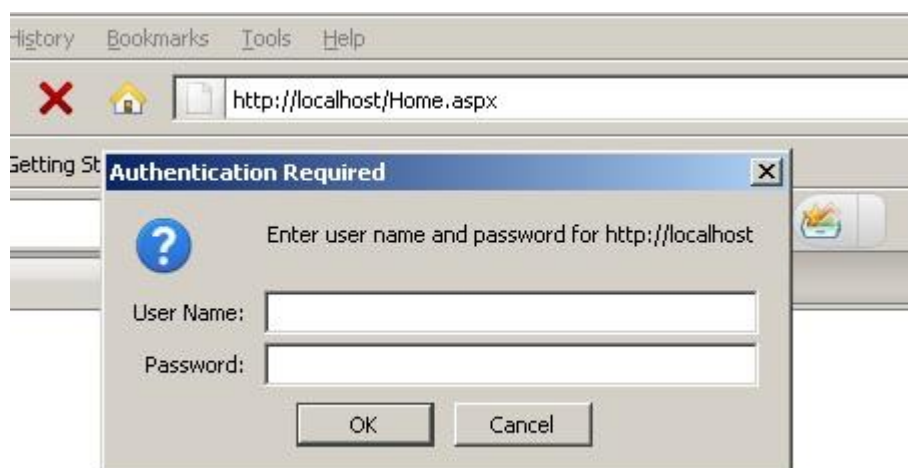
```
'Admin.aspx' pages.
<location path="Admin.aspx">
<system.web>
<authorization>
<allow roles="questpon-srize2\Administrator"/>
<deny users="*" />
</authorization>
</system.web>
</location>
```

Step 5:-Configure IIS settings

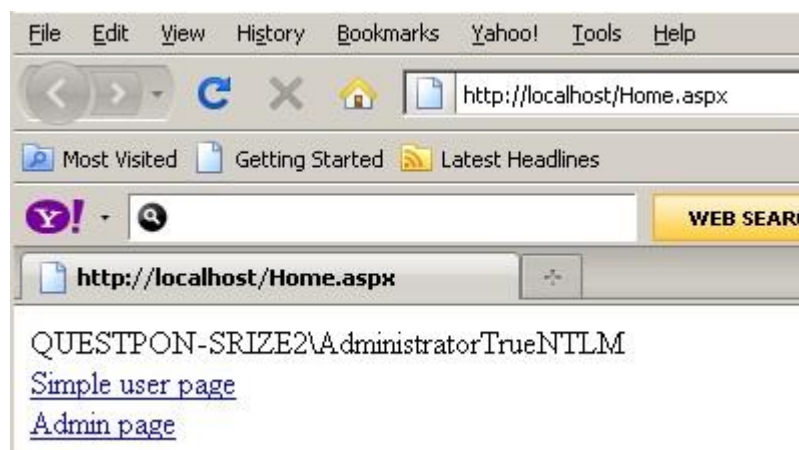
The next step is to compile the project and upload the same on an IIS virtual directory. On the IIS virtual directory we need to ensure to remove anonymous access and check the integrated windows authentication as shown in the below figure.



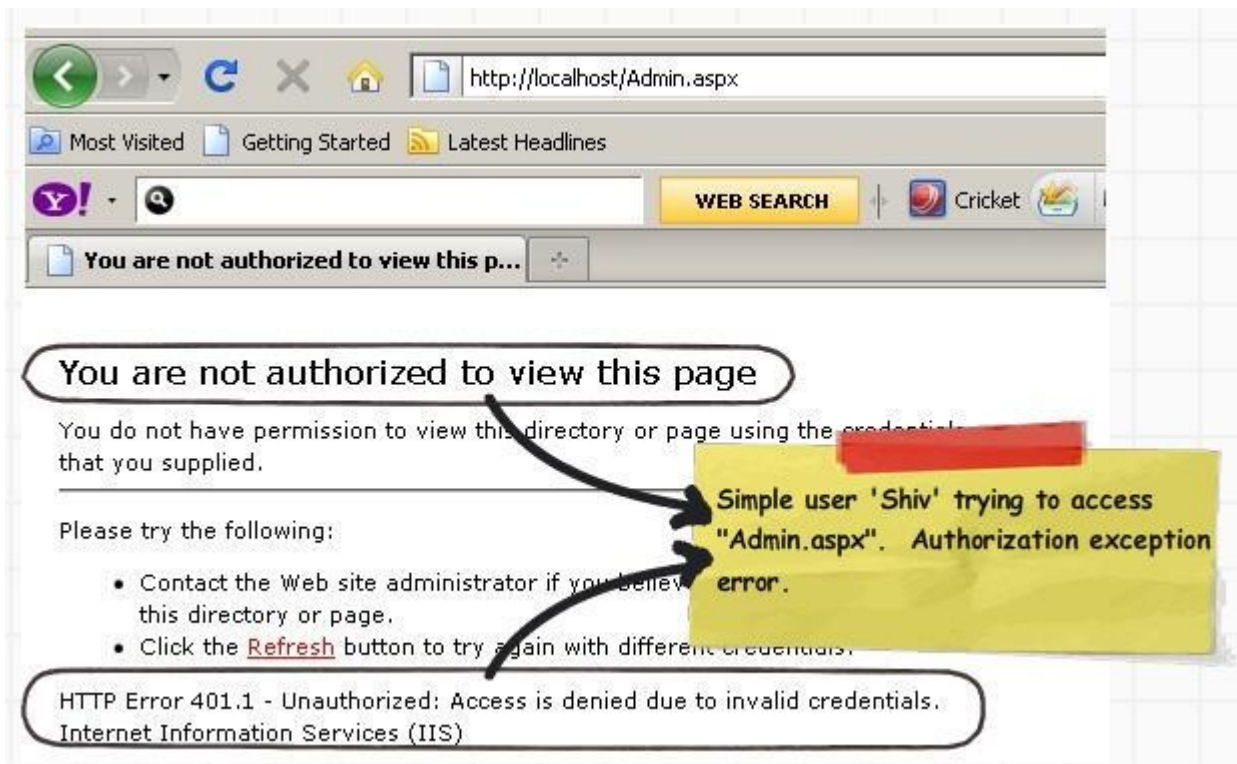
Now if you run the web application you will be popped with a userid and password box as shown below.



Once you enter credentials you should be able to see home.aspx as shown below.



In case you are not an administrator (i.e in this case its 'shiv') and you navigate to 'Admin.aspx' it will throw an error as shown in the below figure.



In case you want to read who the user is and with what authorization rights has he logged in you can use 'WindowsPrincipal' and 'WindowsIdentity'. These two objects represent users who have been authenticated with Windows authentication. You can also get the roles these users have.

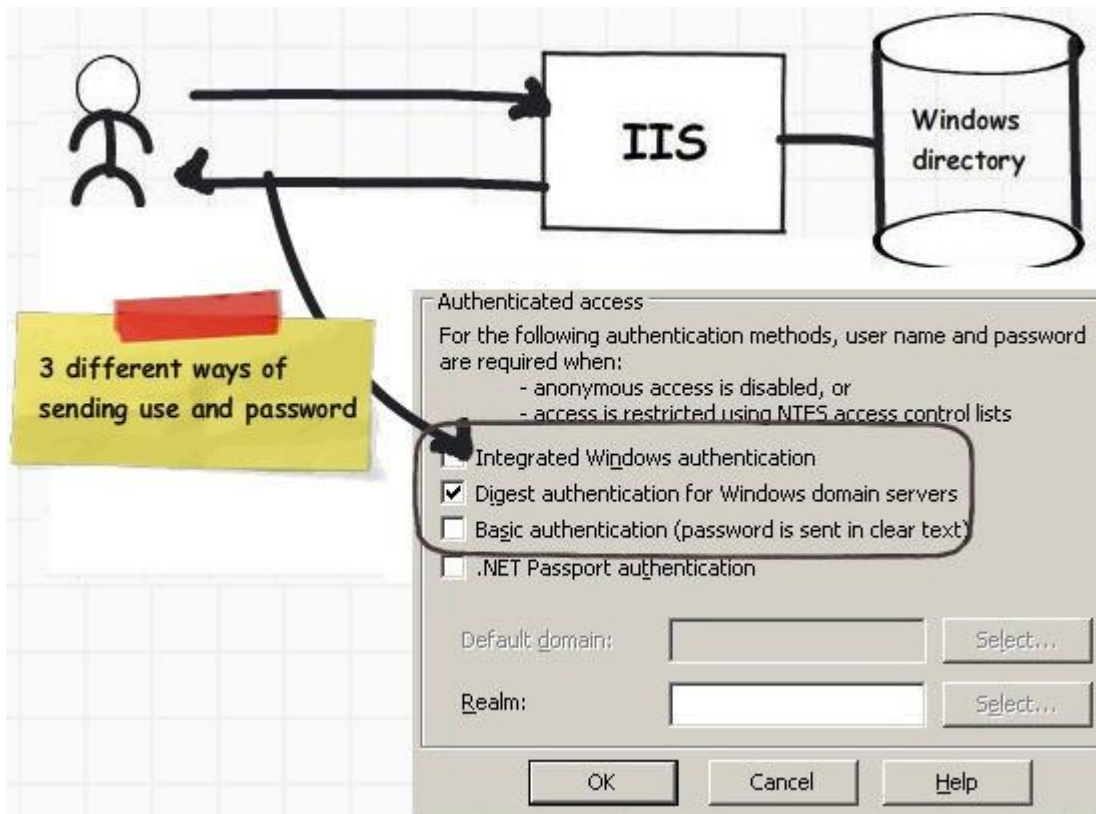
Different methods of collecting username and password

In the above step by step article you must have noticed the below options on IIS (Integrated, digest and basic). These three checkboxes decide how the windows username and password credentials are passed from the client desktop to the IIS.

There are three different way of passing the windows username and password to IIS:-

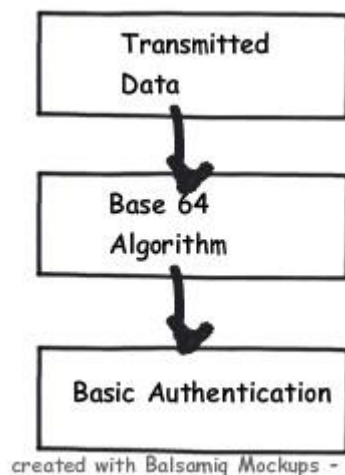
- Basic
- Digest
- Windows

In the coming session we will understand in depth what these 3 options are.



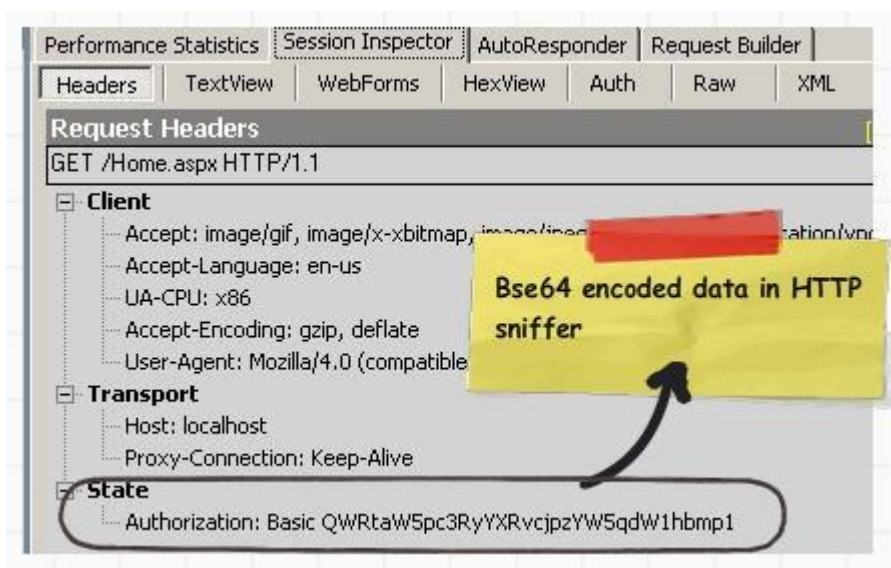
Basic Authentication

When basic authentication is selected the 'userid' and 'password' are passed by using Base64 encoded format . i.e. why the name is basic authentication. 'Base64' is a encoding and not encryption. So it's very easy to crack. You can read more about 'Base64' encoding at <http://en.wikipedia.org/wiki/Base64> . Its a very weak form of protection.

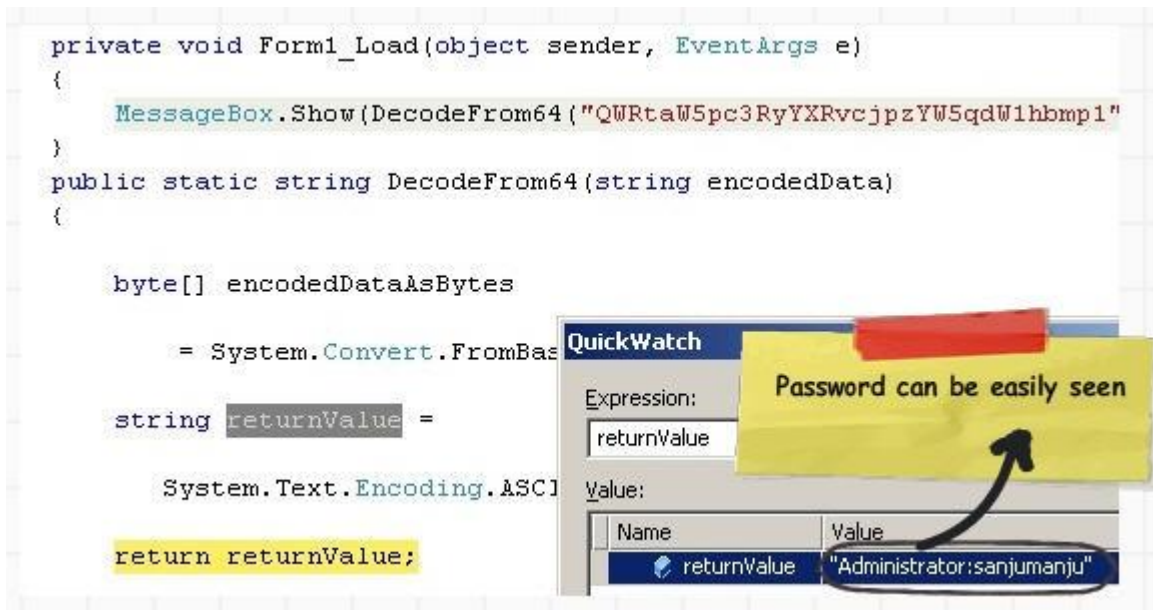




Below is a small demonstration how easy it is to crack a basic authentication. You can see in the below figure we have checked 'Basic authentication' check and we ran the fiddler tool to see the data.



We then copied the 'Authorization:Basic' data and ran the below program. LOL, we can see our windows userid and password.



Below is the code snippet of how to decode 'Base64' encoding.

```
public static string DecodeFrom64(string encodedData)
{
    byte[] encodedDataAsBytes = System.Convert.FromBase64String(encodedData);
    string returnValue = System.Text.Encoding.ASCII.GetString(encodedDataAsBytes);
    return returnValue;}
}
```

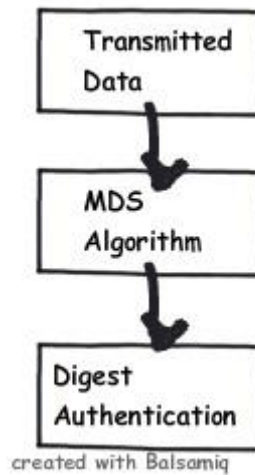
Base64 is an encoding mechanism and not encryption

Basic authentication in other words 'Base64' encoding is used to transmit binary data and convert them to text so that they can run over the network. Some protocols may interpret your binary data as control characters. For instance the FTP protocol for certain combination of binary characters can interpret the same as FTP line endings.

At the end of the days it's not an encryption algorithm it's an encoding mechanism. That's why in our previous section we demonstrated how easy it was to decode basic authentication.

Digest Authentication

The problem associated with basic authentication is solved by using digest authentication. We saw in our previous section how easy it was to crack basic authentication. Digest authentication transfers data over wire as MD5 hash or message digest. This hash or digest is difficult to decipher. In other words digest authentication replaces the lame basic authentication.



Said and done there one of the big problems of digest authentication is it's not supported on some browsers.

Integrated Authentication

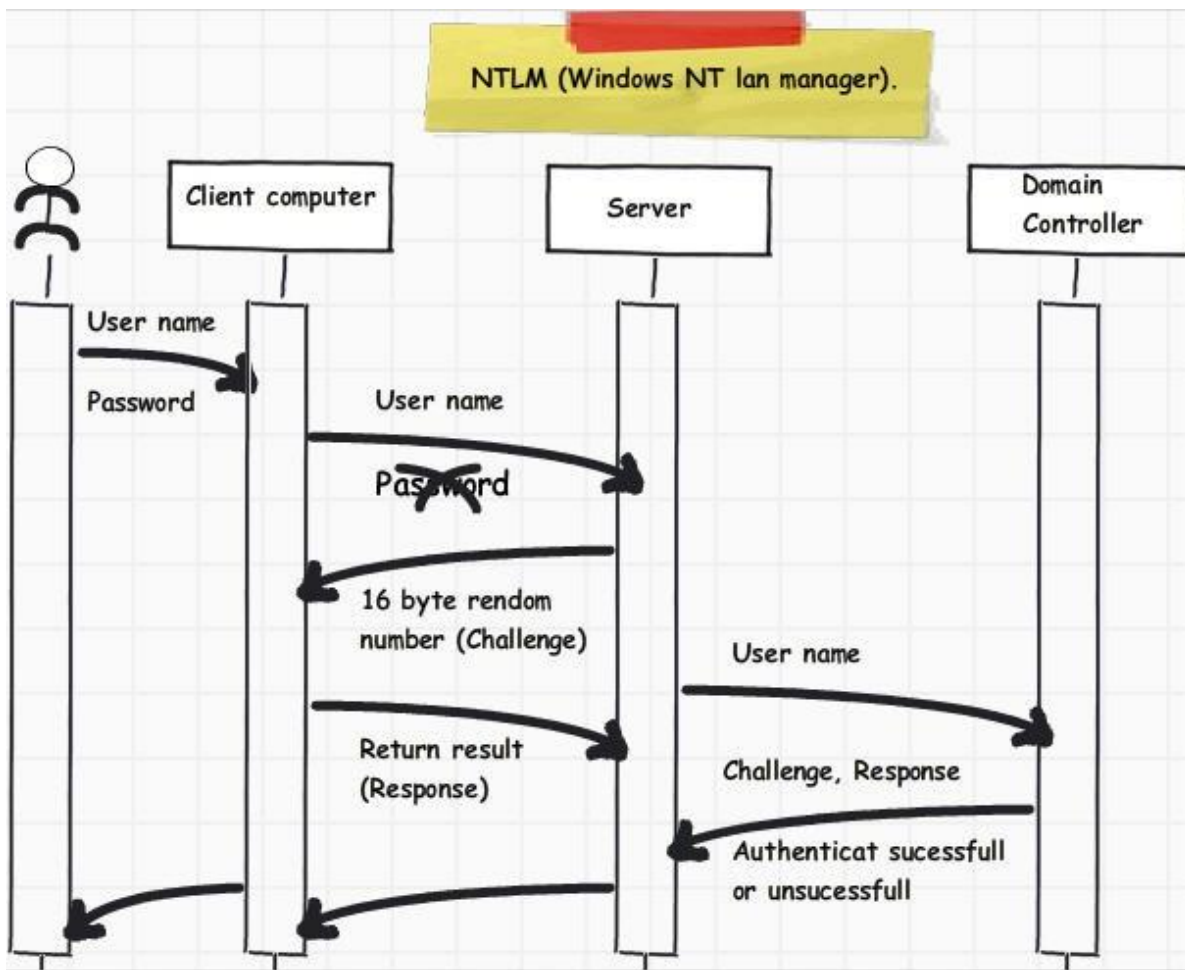
Integrated Windows authentication (formerly called NTLM, and also known as Windows NT Challenge/Response authentication) uses either Kerberos v5 authentication or NTLM authentication, depending upon the client and server configuration.

(The above paragraph is ripped from <http://msdn.microsoft.com/en-us/library/ff647076.aspx>).

Let's try to understand what NTLM and Kerberos authentication is all about and then we will try to understand other aspects of integrated authentication.

NTLM is a challenge response authentication protocol. Below is how the sequence of events happens:-

- Client sends the username and password to the server.
- Server sends a challenge.
- Client responds to the challenge with 24 byte result.
- Servers checks if the response is properly computed by contacting the domain controller.
- If everything is proper it grants the request.

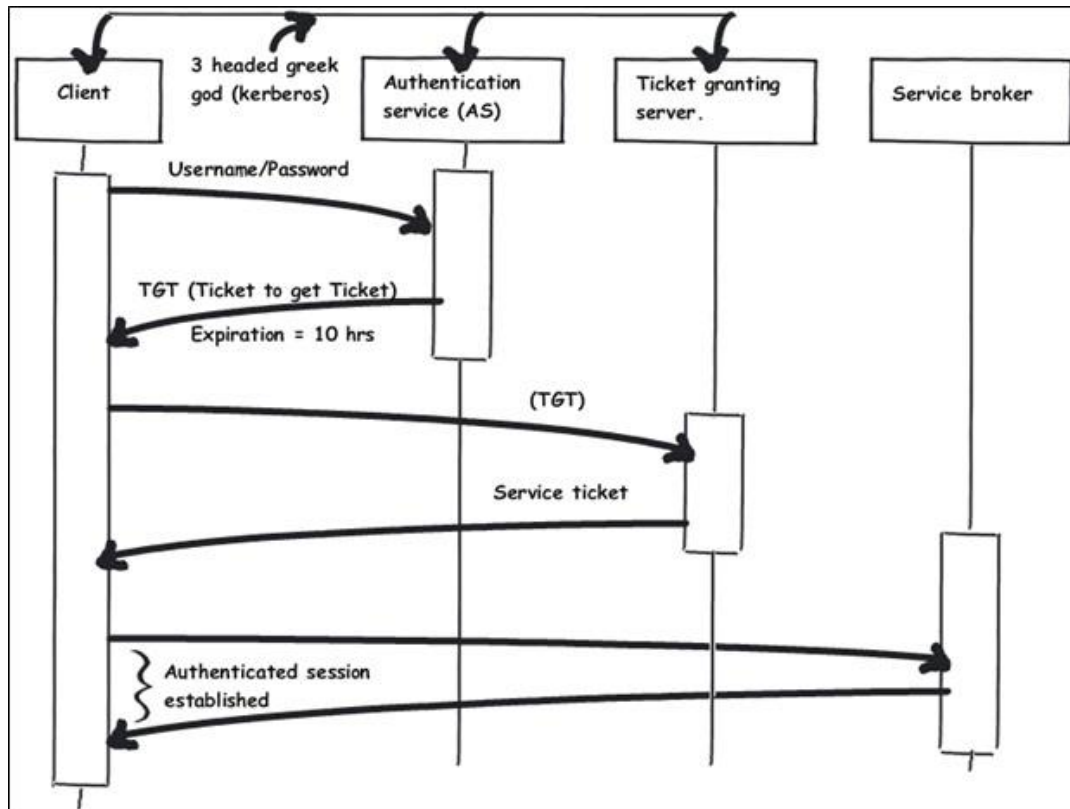


Kerberos is a multi-headed (3 heads) who guards the gates of hades. In the same way Kerberos security has 3 participants authenticating service, service server and ticket granting server. Let's try to understand step by step how these 3 entities participate to ensure security.



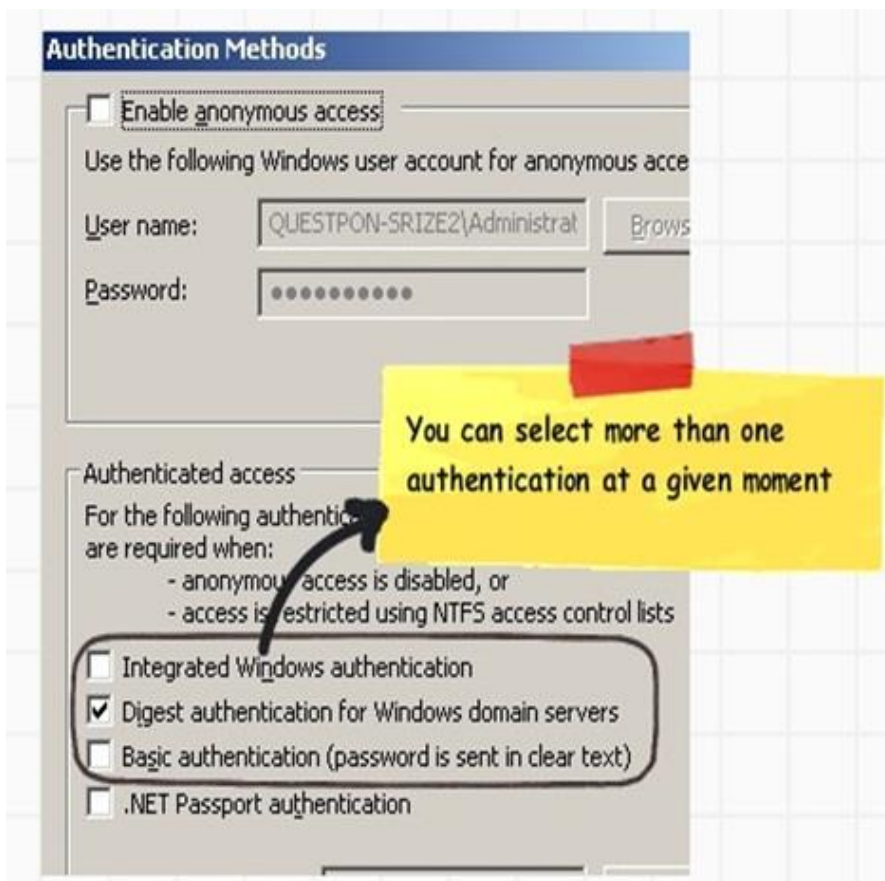
Kerberos uses symmetric key for authentication and authorization. Below is how the things work for Kerberos:-

- In step 1 client sends the username and password to AS (Authenticating service).
- AS authenticates the user and ensures that he is an authenticated user.
- AS then asks the TGT (Ticket granting service) to create a ticket for the user.
- This ticket will be used to verify if the user is authenticated or not. In other words in further client interaction no password will be sent during interaction.



Order of Precedence

One of the things which you must have noticed is that integrated, digest and basic authentication are checkboxes. In other words we can check all the three at one moment of time. If you check all the 3 options at one moment of time depending on browser security support one of the above methods will take precedence.



Let's understand how the security precedence works as per browser security.

- Browser makes a request; it sends the first request as Anonymous. In other words it does not send any credentials.
 - If the server does not accept Anonymous IIS responds with an "Access Denied" error message and sends a list of the authentication types that are supported by the browser.
 - If Windows NT Challenge/Response is the only one supported method then the browser must support this method to communicate with the server. Otherwise, it cannot negotiate with the server and the user receives an "Access Denied" error message.
 - If Basic is the only supported method, then a dialog box appears in the browser to get the credentials, and then passes these credentials to the server. It attempts to send these credentials up to three times. If these all fail, the browser is not connected to the server.
 - If both Basic and Windows NT Challenge/Response are supported, the browser determines which method is used. If the browser supports Windows NT Challenge/Response, it uses this method and does not fall back to Basic. If Windows NT Challenge/Response is not supported, the browser uses Basic.
- You can read more about precedence from <http://support.microsoft.com/kb/264921>.

In order words the precedence is:-

1. Windows NT challenge (Integrated security)
2. Digest
3. Basic

Comparison of Basic, digest and windows authentication

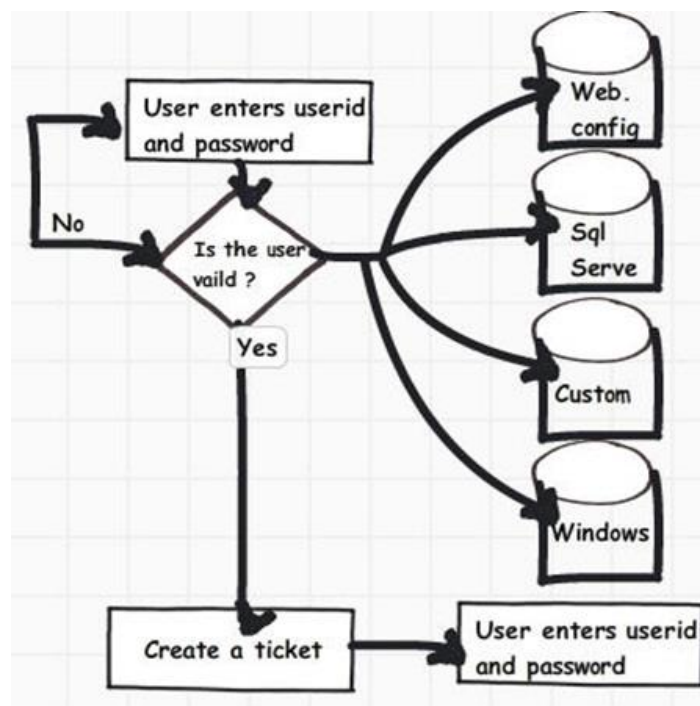
	Browse support	Authentication mechanism
Basic	Almost all browsers	Weak uses Base64.
Digest	IE 5 and later version	Strong MD5
Integrated windows		
• Kerberos	IE5 and above	Ticket encryption using AD , TGT and KDC
• Challenge / response	IE5 and above	Send a challenge

Forms Authentication

Forms authentication is a cookie/URL based authentication where username and password are stored on client machines as cookie files or they are sent encrypted on the URL for every request if cookies are not supported.

Below are the various steps which happen in forms authentication:-

- **Step 1:-** User enters “userid” and “password” through a custom login screen developed for authentication and authorization.
- **Step 2:-** A check is made to ensure that the user is valid. The user can be validated from ‘web.config’ files, SQL Server, customer database, windows active directory and various other kinds of data sources.
- **Step 3:-** If the user is valid then a cookie text file is generated on the client end. This cookie text file signifies that the user has been authenticated. Hence forth when the client computer browses other resources of your ASP.NET site the validation is not conducted again. The cookie file indicates that the user has logged in.



Forms authentication using 'web.config' as a data store

So let's understand step by step how to configure forms authentication. As said in the previous sections you can store user in 'web.config' files. Definitely it's not the best way to store user in "web.config" files but it will really help us to understand forms authentication. Once we understand the same we can then move ahead to better improvised versions of forms authentication.

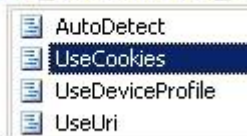
Step 1:- The first thing we need to do is make an entry in to the web.config file with authentication mode as forms as shown below. We need to also provide the following things :-

- LoginUrl :- This property helps us to provide the start page of authentication and authorization.
- defaultUrl :- Once the user is validated he will be redirected to this value , currently its "Home.aspx".
- Cookieless :- As said previously forms authentication uses cookies. There are four ways by which you can change this behavior :-
 - o AutoDetect: - Depending on your browser configuration it can either use cookies or pass the authentication information encrypted via browser URL.
 - o UseCookies: - You would like the forms authentication mechanism to create cookie when the authentication is successful.
 - o UseUri :- You would like to pass data encrypted via the browser URL query string.
 - o UseDeviceProfile :- This is the default value. When you set this value the forms authentication mechanism will do look up at "C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG\Browsers" to see if the browser support cookies and then decides whether it should use cookies or should not. In other words it does not check on actual runtime if the browser has cookies enabled.
- Credentials: - In the credentials tag we have also some users with name and password. As said previously we will first use forms authentication with username's stored in web.config files.

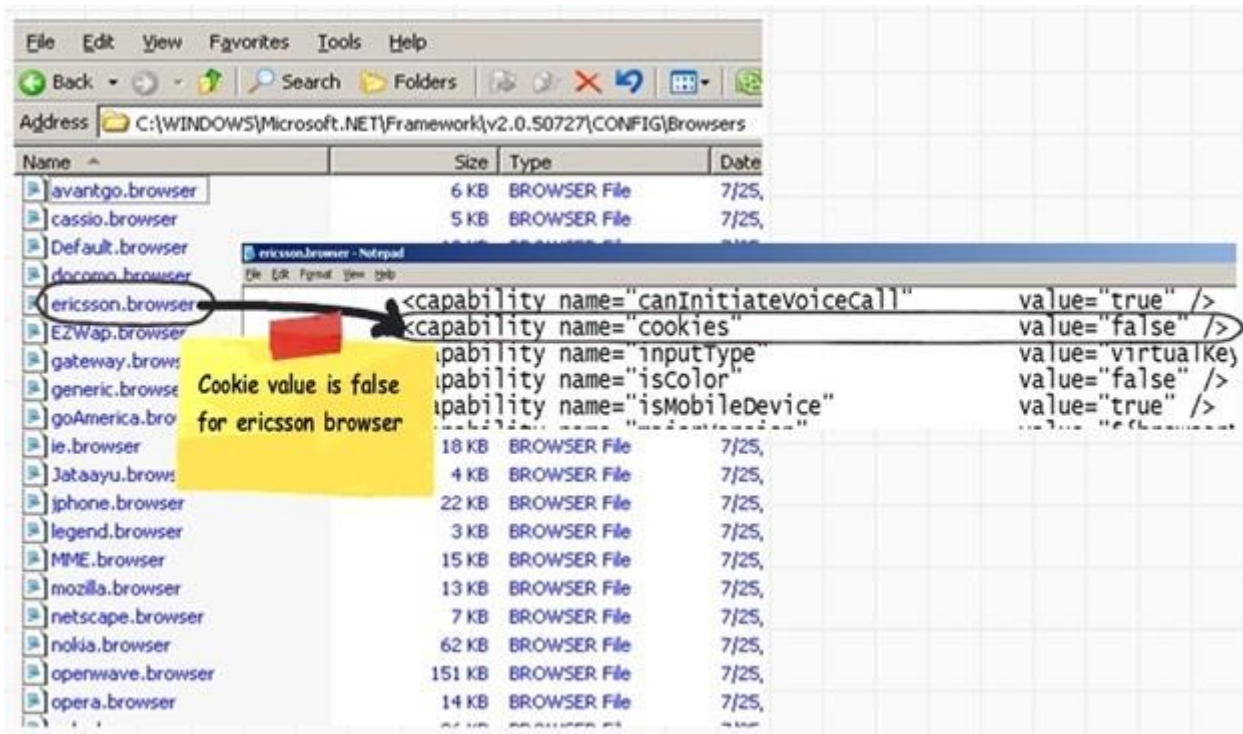
```
<authentication mode="Forms">
<forms loginUrl="Login.aspx" timeout="30" defaultUrl="Home.aspx" cookieless="AutoDetect">
<credentials passwordFormat="Clear">
<user name="Shiv" password="pass@123"/>
<user name="Raju" password="pass@123"/>
</credentials>
</forms>
</authentication>
```

Different customization values for 'cookieless' property.

```
defaultUrl="Home.aspx" cookieless="UseCookies">
123"/>
123"/>
```



If you set the cookieless as 'UseDeviceProfile' it will use the browser data from the below file. You can see how Ericsson browser does not support cookies. So if any one connects with ericsson browser and the value is 'UseDeviceProfile', forms authentication will pass data through query strings.



Step 2:- Once you have set the "forms" tag values, it's time to ensure that anonymous users are not able to browse your site. You can set the same by using the authorization tag as shown in the below code snippet.

```
<authorization>
<deny users="?" />
</authorization>
```

Step 3:- We also need to define which user have access to which page. In this project we have created two pages "Admin.aspx" and "User.aspx". "Admin.aspx" is accessible to only user "Shiv" while "Admin.aspx" and "User.aspx" is accessible to both the users.

Below web.config settings show how we can set the user to pages.

```
<location path="Admin.aspx">
<system.web>
<authorization>
<allow users="Shiv"/>
<deny users="*" />
</authorization>
</system.web>
</location>
<location path="User.aspx">
<system.web>
<authorization>
<allow users="Shiv"/>
<allow users="Raju"/>
<deny users="*" />
```

```
</authorization>  
</system.web>  
</location>
```

Step 4 :- We now create our custom page which will accept userid and password.

Username

Password

Button

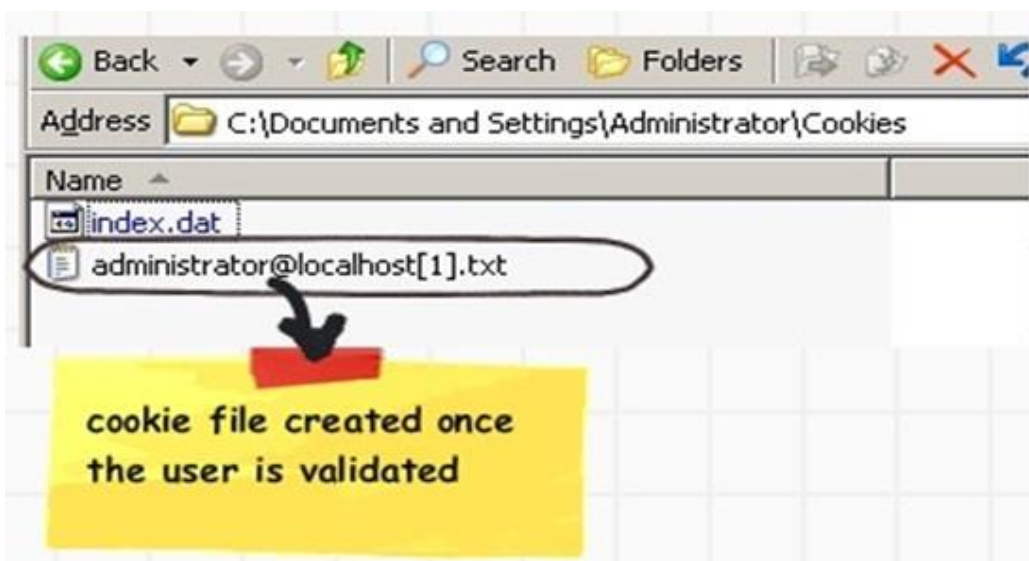
In the button click we provide the below code. The “FormsAuthentication.Authenticate” looks in the web.config the username and passwords. The “FormsAuthentication.RedirectFromLoginPage” creates cookies at the browser end.

Check if user is present in web.config file

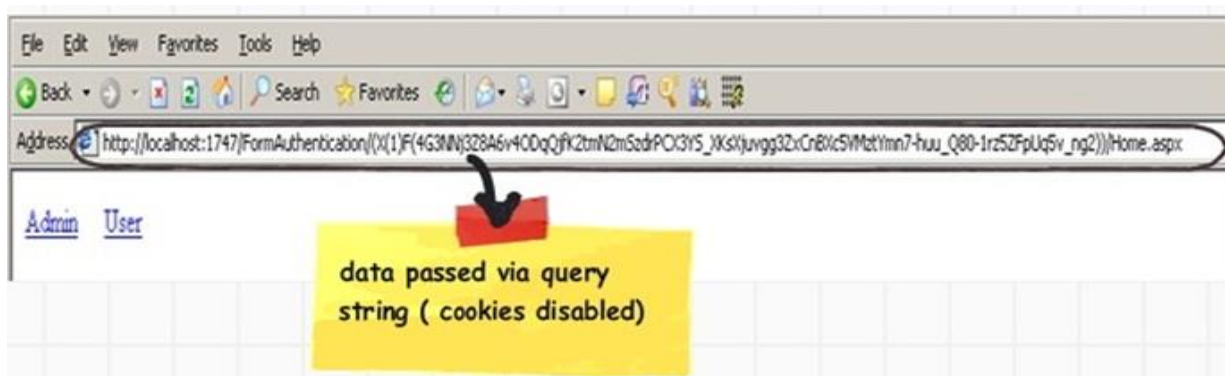
```
if (FormsAuthentication.Authenticate(txtUser.Text, txtPass.Text))  
{  
    FormsAuthentication.RedirectFromLoginPage(txtUser.Text, true);  
}
```

Creates cookie in the client browser

If you run your application , enter proper credentials , you should be able to see a cookie txt file created as shown in the below figure.



If you disable cookies using the browser settings, credentials will be passed via query string as shown in the below figure.



Forms Authentication using SQL server as a data store

In order to do custom authentication you need to need to just replace “FormsAuthentication.Authenticate” statement with your validation. For instance in the below code we have used ‘clsUser’ class to do authentication but we have yet used the cookie creation mechanism provided by ‘FormAuthentication’ system.

```
clsUser objUser = new clsUser();
if (objUser.IsValid(txtUser.Text,txtPass.Text))
{
FormsAuthentication.RedirectFromLoginPage(txtUser.Text, true);
}
```

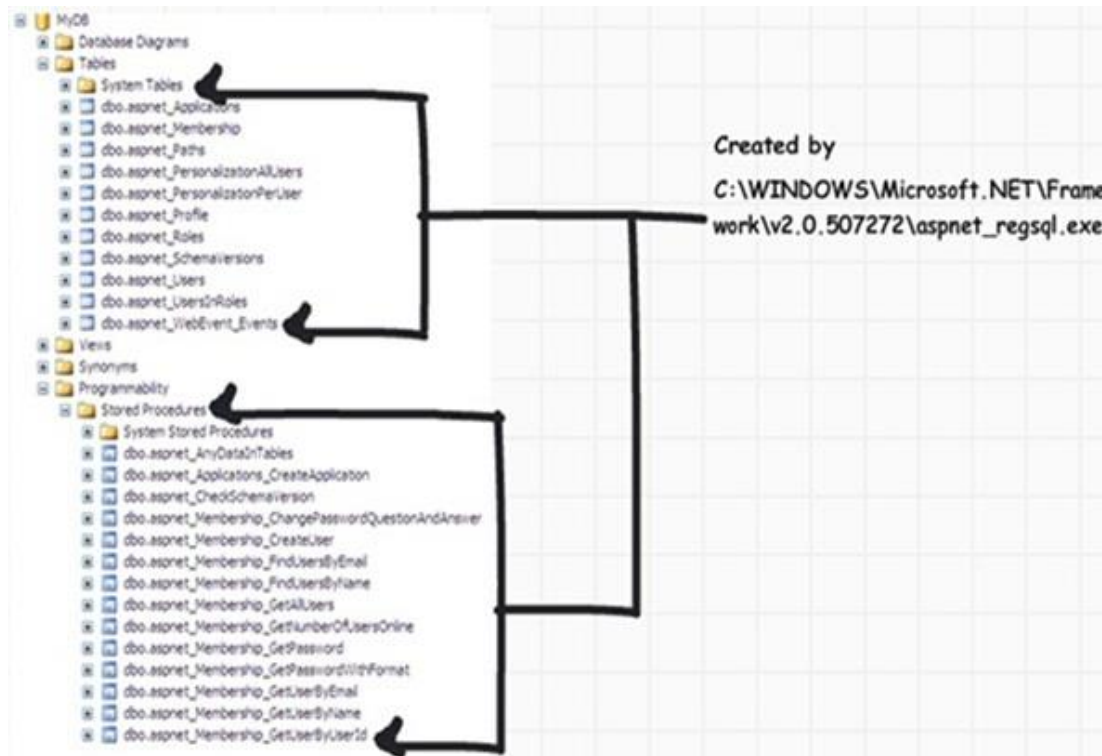
Forms authentication using ASP.NET Membership and role

We have used forms authentication mechanism to generate cookie which has minimized lot of our development effort. Many other tasks we are still performing like:-

- Creation of user and roles tables.
- Code level implementation for maintaining those tables.
- User interface for userid and password.

We are sure you must have done the above task for every project again and again. Good news!!! All the above things are now made simple with introduction of membership and roles. To implement ASP.NET membership and roles we need to do the following steps :-

- Run aspnet_regsql.exe from 'C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727' folder. Enter SQL Server credentials and run the exe. This will install all the necessary stored procedures and tables as shown in figure 'Object created by aspnet_regsql.exe'



- Specify the connection string in the 'web.config' file where your ASP.NET roles tables and stored procedures are created.

```
<connectionStrings>
<remove name="LocalSqlServer1"/>
<add name="LocalSqlServer1" connectionString="Data Source=localhost;Initial
Catalog=test;Integrated Security=True"/>
</connectionStrings>
```

- Specify the ASP.NET membership provider and connect the same with the connection string provided in the previous step.

```
<membership>
<providers>
<remove name="AspNetSqlMembershipProvider"/>
<add name="AspNetSqlMembershipProvider"
type="System.Web.Security.SqlMembershipProvider, System.Web,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
connectionStringName="LocalSqlServer1" enablePasswordRetrieval="false"
enablePasswordReset="true" applicationName="/" minRequiredPasswordLength="7"/>
</providers>
```

```
</membership>
```

- We also need to specify the role provider and connect the same with the connection string provided in the previous session.

```
<roleManager enabled="true">
<providers>
<clear/>
<add name="AspNetSqlRoleProvider" connectionStringName="LocalSqlServer1"
applicationName="/" type="System.Web.Security.SqlRoleProvider, System.Web,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"/>
</providers>
</roleManager>
```

Now you can use the “Membership” class to create users and roles as shown in the below 2 figures.

```
Membership.CreateUser("Shiv_koirala", "Interview123");
```

Creates a user

Table - dbo.aspnet_Users			
ApplicationId	UserName	LoweredUserN...	
b7d-b9eda977305x	Shiv_koirala	shiv_koirala	/
3fec7697-8255-...	Shiv	shiv	/
3fec7697-8255-...	Shiv123	shiv123	/
87d88698-b0e5-...	Shiv123	shiv123	/
NULL	NULL	NULL	/

```
user[0]="Shiv_koirala";
Roles.CreateRole("Developer");
Roles.AddUsersToRole(user, "Developer");
```

1

2



Username	RoleName
Shiv123	Developer

You can get a feel how easy it is to use develop authentication and authorization by using forms authentication and ASP.NET membership and roles.

The dual combination

Authentication and authorization in any application needs 2 things:-

- Mechanism by which you can generate a cookie: - Provided by Forms authentication.
- Custom tables in SQL Server to store user and roles: - Provided by ASP.NET provider and roles.

In other words by using the combination of ticket generation via forms authentication and ASP.NET provider and roles we can come up with a neat and quick solution to implement authentication and authorization in ASP.NET applications.

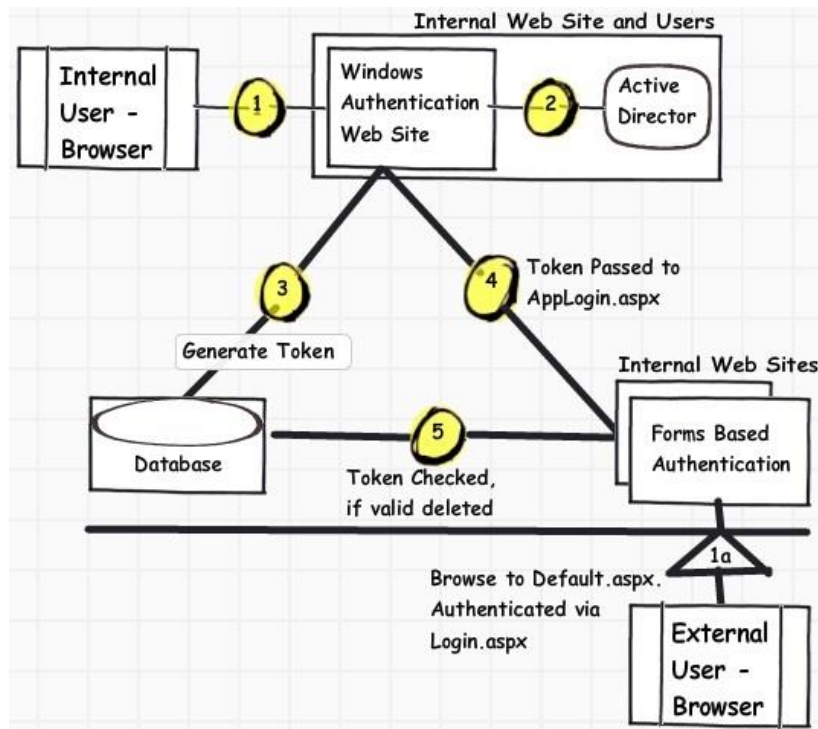
Forms Authentication using Single Sign on

Many time we would like to implement single sign on across multiple sites. This can be done using forms authentication. You can implement forms authentication in both the websites with same machine key. Once the validation is done in one website a cookie text file will be created. When that user goes to the other website the same cookie file will used to ensure that the user is proper or not.

Please note you need to have same machine key in both the web.config files of your web application.

```
<machineKey  
validationKey="C50B3C89CB21F4F1422FF158A5B42D0E8DB8CB5CDA1742572A487D9401E340  
0267682B202B746511891C1BAF47F8D25C07F6C39A104696DB51F17C529AD3CABE"  
decryptionKey="8A9BE8FD67AF6979E7D20198CFEA50DD3D3799C77AF2B72F"  
validation="SHA1" />
```

You can see a very detail article on Single sign at <http://msdn.microsoft.com/en-us/library/ms972971.aspx> . The above discusses how a internal intranet and internet application login through one single sign-on facility.



Passport Authentication

Passport authentication is based on the passport website provided by the Microsoft .So when user logs in with credentials it will be reached to the passport website (i.e. hotmail,devhood,windows live etc) where authentication will happen. If Authentication is successful it will return a token to your website. I am leaving this section for now, will update in more details soon