

- **The Structure of an ASP.NET Page**

You are now in a position to examine the formal structure of an ASP.NET page. The following is a list of the important elements of an ASP.NET page:

- Directives
- Code declaration blocks
- ASP.NET controls
- Code render blocks
- Server-side comments
- Server-side include directives
- Literal text and HTML tags

Each element is discussed in the following sections.

Directives

A *directive* controls how an ASP.NET page is compiled. The beginning of a directive is marked with the characters `<%@` and the end of a directive is marked with the characters `%>`. A directive can appear anywhere within a page. By convention, however, a directive typically appears at the top of an ASP.NET page.

There are several types of directives that you can add to an ASP.NET page. Two of the most useful types are page and import directives.

Page Directives

You can use a page directive to specify the default programming language for a page. Page directives can also be used to enable tracing and debugging for a page.

To change the default programming language of an ASP.NET page from Visual Basic to C#, for example, you would use the following page directive:

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

NOTE

The keyword `Page` in a page directive is optional. The following two directives are equivalent:

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

Another extremely useful page directive is the `Trace` directive. If you enable tracing for a page, additional information about the execution of the page is displayed along with the content of the page (see Figure 5). You can enable tracing for a page by including the following directive:

```
<%@ Page Trace="True" %>
```

After you enable tracing for a page, you can display trace messages by using two methods of the `Trace` class: `Write()` and `Warn()`. You can use either method to display a custom message in the trace information displayed at the bottom of the page. The only difference between the two methods is that the former method displays messages in black text, and the latter method displays messages in red text, which is easier to see.

Figure 5 Enabling page tracing.

The ASP.NET page in Listing 11 uses these methods to display various trace messages.

Listing 11—Trace.aspx

```

<%@ Page Trace="True" %>
<Script Runat="Server">

Sub Page_Load
    Dim strTraceMessage As String

    Trace.Warn( "Page_Load event executing!" )
    strTraceMessage = "Hello World!"
    Trace.Write( "The value of strTraceMessage is " & strTraceMessage )
End Sub

</Script>

<html>
<head><title>Trace.aspx</title></head>
<body>

<h2>Testing Page Trace</h2>

<% Trace.Warn( "Rendering page content!" ) %>

</body>
</html>

```

To enable runtime error messages to be displayed on a page, you need to use the Debug directive. To display errors in your ASP.NET page, include the following directive:

```

<%@ Page Debug="True" %>

```

When you include this directive, if an error is encountered when processing the page, the error is displayed. In most cases, you also can view the source code for the exact statement that generated the error.

To enable both tracing and debugging for a page, combine the directives like this (the order of Debug and Trace is not important):

```

<%@ Page Debug="True" Trace="True" %>

```

Import Directives

By default, only certain namespaces are automatically imported into an ASP.NET page. If you want to refer to a class that isn't a member of one of the default namespaces, then you must explicitly import the namespace of the class or you must use the fully qualified name of the class.

For example, suppose that you want to send an email from an ASP.NET page by using the Send method of the SmtpMail class. The SmtpMail class is contained in the System.Web.Mail namespace. This is not one of the default namespaces imported into an ASP.NET page.

The easiest way to use the Smtplib class is to add an Import directive to your ASP.NET page to import the necessary namespace. The page in Listing 12 illustrates how to import the System.Web.Mail namespace and send an email message.

Listing 12—ImportNamespace.aspx

```
<%@ Import Namespace="System.Web.Mail" %>

<Script Runat="Server">

Sub Page_Load
  Dim mailMessage As Smtplib

  mailMessage.Send( _
    "bob@somewhere.com", _
    "alice@somewhere.com", _
    "Sending Mail!", _
    "Hello!" )
End Sub

</Script>

<html>
<head><title>ImportNamespace.aspx</title></head>
<body>

<h2>Email Sent!</h2>

</body>
</html>
```

The first line in Listing 12 contains an import directive. Without the import directive, the page would generate an error because it would not be able to resolve the Smtplib class. Instead of importing the System.Web.Mail namespace with the import directive, you could alternatively use its fully qualified name. In that case, you would declare an instance of the class like this:

```
Dim mailMessage As System.Web.Mail.Smtplib
```

Code Declaration Blocks

A *code declaration block* contains all the application logic for your ASP.NET page and all the global variable declarations, subroutines, and functions. It must appear within a <Script Runat="Server">tag.

Note that you can declare subroutines and functions only within a code declaration block. You receive an error if you attempt to define a function or subroutine in any other section of an ASP.NET page.

ASP Classic Note

In previous versions of ASP, you could declare a subroutine or function like this:

```
<%  
Sub mySub  
... subroutine code  
End Sub  
%>
```

Don't try this declaration with ASP.NET because it generates an error. You must use the <Script>tag with ASP.NET like this:

```
<Script runat="Server">  
Sub mySub  
...subroutine code  
End Sub  
</Script>
```

The <Script Runat="Server"> tag accepts two optional attributes. First, you can specify the programming language to use within the <Script> tag by including a language attribute like this:

```
<Script Language="C#" Runat="Server">
```

If you don't specify a language, the language defaults to the one defined by the <%@ Page Language %> directive mentioned in the previous section. If no language is specified in a page, the language defaults to Visual Basic.

ASP Classic Note

In previous versions of Active Server Pages, you could include multiple languages in the same page by using the language attribute with the <Script> tag. This technique doesn't work with ASP.NET. You get an error if you attempt to use more than one language in the same page.

The second optional attribute of the <Script Runat="Server"> tag is SRC. You can use it to specify an external file that contains the contents of the code block. This is illustrated in Listings 13 and 14.

Listing 13—ExternalFile.aspx

```
<Script Runat="Server" SRC="ApplicationLogic.aspx"/>  
  
<html>  
<head><title>ExternalFile.aspx</title></head>  
<body>  
  
<form Runat="Server">  
  
<asp:label id="lblMessage" Runat="Server"/>  
  
<asp:Button  
Text="Click Here!"  
OnClick="Button_Click"
```

```
Runat="Server"/>
```

```
</form>
```

```
</body>
```

```
</html>
```

Listing 14—ApplicationLogic.aspx

```
Sub Button_Click( s As Object, e As EventArgs )
```

```
    lblMessage.Text = "Hello World!"
```

```
End Sub
```

The page in Listing 13, `ExternalFile.aspx`, uses the `SRC` attribute to include the code in Listing 14, `ApplicationLogic.aspx`.

ASP.NET Controls

ASP.NET controls can be freely interspersed with the text and HTML content of a page. The only requirement is that the controls should appear within a `<form Runat="Server">` tag. And, for certain tags such as `` and `<ASP:Label Runat="Server"/>`, this requirement can be ignored without any dire consequences.

One significant limitation of ASP.NET pages is that they can contain only one `<form Runat="Server">` tag. This means that you cannot group ASP.NET into multiple forms on a page. If you try, you get an error.

Code Render Blocks

If you need to execute code within the HTML or text content of your ASP.NET page, you can do so within *code render blocks*. The two types of code render blocks are *inline code* and *inline expressions*. Inline code executes a statement or series of statements. This type of code begins with the characters `<%` and ends with the characters `%>`.

Inline expressions, on the other hand, display the value of a variable or method (this type of code is shorthand for `Response.Write`). Inline expressions begin with the characters `<%=` and end with the characters `%>`.

The ASP.NET page in Listing 15 illustrates how to use both inline code and inline expressions in an ASP.NET page.

Listing 15—CodeRender.aspx

```
<Script Runat="Server">
```

```
    Dim strSomeText As String
```

```
Sub Page_Load
```

```
    strSomeText = "Hello!"
```

```
End Sub
```

```
</Script>
```

```
<html>
```

```
<head><title>CodeRender.aspx</title></head>
```

```
<body>
```

```
<form Runat="Server">
```

The value of strSomeText is:

```
<%=strSomeText%>
```

```
<p>
```

```
<% strSomeText = "Goodbye!" %>
```

The value of strSomeText is:

```
<%=strSomeText%>
```

```
</form>
```

```
</body>
```

```
</html>
```

Notice that you can use variables declared in the code declaration block within the code render block. However, the variable has to be declared with page scope. The variable could not, for example, be declared within the Page_Load subroutine.

Server-side Comments

You can add comments to your ASP.NET pages by using *server-side comment* blocks. The beginning of a server-side comment is marked with the characters `<%--` and the end of the comment is marked with the characters `--%>`.

Server-side comments can be added to a page for the purposes of documentation. Note that you cannot see the contents of server-side comment tags, unlike normal HTML comment tags, by using the View Source command on your Web browser.

Server-side comments can also be useful when you're debugging an ASP.NET page. You can temporarily remove both ASP.NET controls and code render blocks from a page by surrounding these elements with server-side comments. The page in Listing 16 illustrates this use.

Listing 16—ServerComments.aspx

```
<Script Runat="Server">
```

```
Dim strSomeText As String
```

```
Sub Page_Load
```

```
    strSomeText = "Hello!"
```

```
End Sub
```

```
</Script>
```

```
<html>
```

```
<head><title>ServerComments.aspx</title></head>
```

```
<body>
```

```
<form Runat="Server">
```

```
<%--
```

This is inside the comments

```
<asp:Label Text="hello!" Runat="Server" />  
<%= strSomeText %>  
--%>
```

This is outside the comments

```
</form>
```

```
</body>
```

```
</html>
```

Server-side Include Directives

You can include a file in an ASP.NET page by using one of the two forms of the *server-side include directive*. If you want to include a file that is located in the same directory or in a subdirectory of the page including the file, you would use the following directive:

```
<!-- #INCLUDE file="includefile.aspx" -->
```

Alternatively, you can include a file by supplying the full virtual path. For example, if you have a subdirectory named `myDirectory` under the `wwwroot` directory, you can include a file from that directory like this:

```
<!-- #INCLUDE virtual="/myDirectory/includefile.aspx" -->
```

The include directive is executed before any of the code in a page. One implication is that you cannot use variables to specify the path to the file that you want to include. For example, the following directive would generate an error:

```
<!-- #INCLUDE file="<%=myVar%>" -->
```

TIP

Instead of working with include directives, consider working with user controls instead. User controls provide you with more flexibility. For example, you can create custom properties and methods with a user control.

Literal Text and HTML Tags

The final type of element that you can include in an ASP.NET page is HTML content. The static portion of your page is built with plain old HTML tags and text.

You might be surprised to discover that the HTML content in your ASP.NET page is compiled along with the rest of the elements. HTML content in a page is represented with the `LiteralControl` class. You can use the `Text` property of the `LiteralControl` class to manipulate the pure HTML portion of an ASP.NET page.

The HTML content contained in the page in Listing 17, for example, is reversed when it is displayed (see Figure 6). The `Page_Load` subroutine walks through all the controls in a page (including the `LiteralControl`) and reverses the value of the `Text` property of each control.

Listing 17—Literal.aspx

```

<Script Runat="Server">

Sub Page_Load
    Dim litControl As LiteralControl

    For each litControl in Page.Controls
        litControl.Text = strReverse( litControl.Text )
    Next
End Sub

</Script>

<html>
<head><title>Literal.aspx</title></head>
<body>

<b>This text is reversed</b>

</body>
</html>

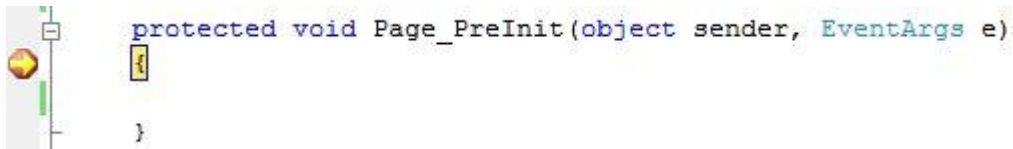
```

- **Page Level Events:**

When an ASP.NET page runs, the page goes through a life cycle in which it performs a series of processing steps. These include initialization, instantiating controls, restoring and maintaining state, running event handler code, and rendering. The following are the various stages or events of ASP.Net page life cycle.

PreInit

1. Check the IsPostBack property to determine whether this is the first time the page is being processed.
2. Create or re-create dynamic controls.
3. Set a master page dynamically.
4. Set the Theme property dynamically.



Note

If the request is a postback then the values of the controls have not yet been restored from the view state. If you set a control property at this stage, its value might be overwritten in the next event.

Init

1. This event fires after each control has been initialized.
2. Each control's UniqueID is set and any skin settings have been applied.
3. Use this event to read or initialize control properties.
4. The "Init" event is fired first for the bottom-most control in the hierarchy, and then fired up the hierarchy until it is fired for the page itself.



```
protected void Page_Init(object sender, EventArgs e)
{
}
```

InitComplete

1. Until now the viewstate values are not yet loaded, hence you can use this event to make changes to the view state that you want to ensure are persisted after the next postback.
2. Raised by the Page object.
3. Use this event for processing tasks that require all initialization to be complete.



```
protected void Page_InitComplete(object sender, EventArgs e)
{
}
```

OnPreLoad

1. Raised after the page loads view state for itself and all controls, and after it processes postback data that is included with the Request instance.
2. Before the Page instance raises this event, it loads view state for itself and all controls, and then processes any postback data included with the Request instance.
3. Loads ViewState: ViewState data are loaded to controls.
4. Loads Postback data: Postback data are now handed to the page controls.



```
protected override void OnPreLoad(EventArgs e)
{
}
```

Load

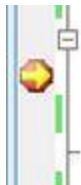
1. The Page object calls the OnLoad method on the Page object, and then recursively does the same for each child control until the page and all controls are loaded. The Load event of individual controls occurs after the Load event of the page.
2. This is the first place in the page lifecycle that all values are restored.
3. Most code checks the value of IsPostBack to avoid unnecessarily resetting state.
4. You may also call Validate and check the value of IsValid in this method.
5. You can also create dynamic controls in this method.
6. Use the OnLoad event method to set properties in controls and establish database connections.



```
protected void Page_Load(object sender, EventArgs e)
```

Control PostBack Event(s)

1. ASP.NET now calls any events on the page or its controls that caused the PostBack to occur.
2. Use these events to handle specific control events, such as a Button control's Click event or a TextBox control's TextChanged event.
3. In a postback request, if the page contains validator controls, check the IsValid property of the Page and of individual validation controls before performing any processing.
4. This is just an example of a control event. Here it is the button click event that caused the postback.



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

LoadComplete

1. Raised at the end of the event-handling stage.
2. Use this event for tasks that require that all other controls on the page be loaded.

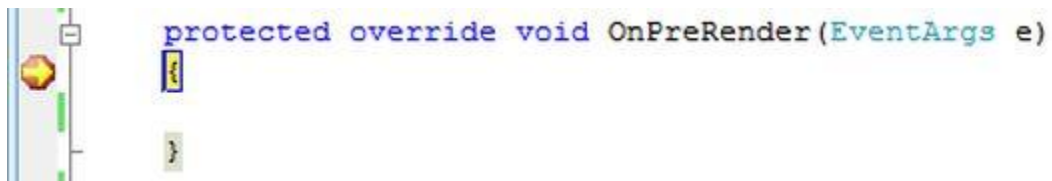


```
protected void Page_LoadComplete(object sender, EventArgs e)
```

OnPreRender

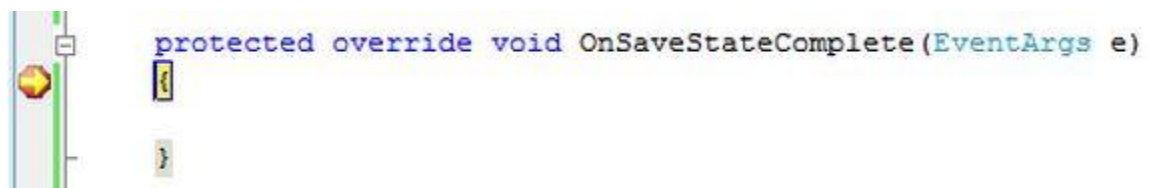
1. Raised after the Page object has created all controls that are required in order to render the page, including child controls of composite controls.
2. The Page object raises the PreRender event on the Page object, and then recursively does the same for each child control. The PreRender event of individual controls occurs after the PreRender event of the page.
3. The PreRender event of individual controls occurs after the PreRender event of the page.
4. Allows final changes to the page or its control.
5. This event takes place before saving ViewState, so any changes made here are saved.
6. For example: After this event, you cannot change any property of a button or change any viewstate value.
7. Each data bound control whose DataSourceID property is set calls its DataBind method.

8. Use the event to make final changes to the contents of the page or its controls.



OnSaveStateComplete

1. Raised after view state and control state have been saved for the page and for all controls.
2. Before this event occurs, ViewState has been saved for the page and for all controls.
3. Any changes to the page or controls at this point will be ignored.
4. Use this event perform tasks that require the view state to be saved, but that do not make any changes to controls.

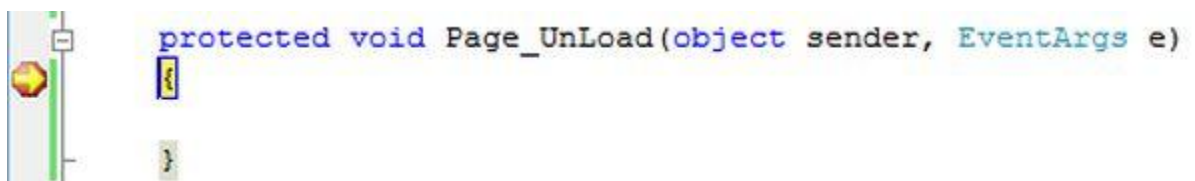


Render Method

1. This is a method of the page object and its controls (and not an event).
2. The Render method generates the client-side HTML, Dynamic Hypertext Markup Language (DHTML), and script that are necessary to properly display a control at the browser.

UnLoad

1. This event is used for cleanup code.
2. At this point, all processing has occurred and it is safe to dispose of any remaining objects, including the Page object.
3. Cleanup can be performed on:
 - Instances of classes, in other words objects
 - Closing opened files
 - Closing database connections.
4. This event occurs for each control and then for the page.
5. During the unload stage, the page and its controls have been rendered, so you cannot make further changes to the response stream.
6. If you attempt to call a method such as the Response.Write method then the page will throw an exception.



- **ASP.NET Page Lifecycle**

In ASP.NET, a web page has execution lifecycle that includes various phases. These phases include initialization, instantiation, restoring and maintaining state etc. it is required to understand the page lifecycle so that we can put custom code at any stage to perform our business logic.

Page Lifecycle stages

The following table contains the lifecycle stages of ASP.NET web page.

Stage	Description
Page request	This stage occurs before the lifecycle begins. When a page is requested by the user, ASP.NET parses and compiles that page.
Start	In this stage, page properties such as Request and response are set. It also determines the Request type.
Initialization	In this stage, each control's UniqueID property is set. Master page is applied to the page.
Load	During this phase, if page request is postback, control properties are loaded with information.
Postback event handling	In this stage, event handler is called if page request is postback. After that, the Validate method of all validator controls is called.
Rendering	Before rendering, view state is saved for the page and all controls. During the rendering stage, the page calls the Render method for each control, providing a text writer that writes its output to the OutputStream object of the page's Response property.
Unload	At this stage the requested page has been fully rendered and is ready to terminate. at this stage all properties are unloaded and cleanup is performed.

A requested page first loaded into the server memory after that processes and sent to the browser. At last it is unloaded from the server memory. ASP.NET provides methods and events at each stage of the page lifecycle that we can use in our application. In the following table, we are tabled events.

ASP.NET Life Cycle Events

Page Event	Typical Use
PreInit	This event is raised after the start stage is complete and before the initialization stage.
Init	This event occurs after all controls have been initialized. We can use this event to read or initialize control properties.

InitComplete	This event occurs at the end of the page's initialization stage. We can use this event to make changes to view state that we want to make sure are persisted after the next postback.
PreLoad	This event is occurs before the post back data is loaded in the controls.
Load	This event is raised for the page first time and then recursively for all child controls.
Control events	This event is used to handle specific control events such as Button control ' Click event.
LoadComplete	This event occurs at the end of the event-handling stage. We can use this event for tasks that require all other controls on the page be loaded.
PreRender	This event occurs after the page object has created all controls that are required in order to render the page.
PreRenderComplete	This event occurs after each data bound control whose DataSourceID property is set calls its DataBind method.
SaveStateComplete	It is raised after view state and control state have been saved for the page and for all controls.
Render	This is not an event; instead, at this stage of processing, the Page object calls this method on each control.
Unload	This event raised for each control and then for the page.

- **Designing Websites with Master pages and themes:**

Introduction

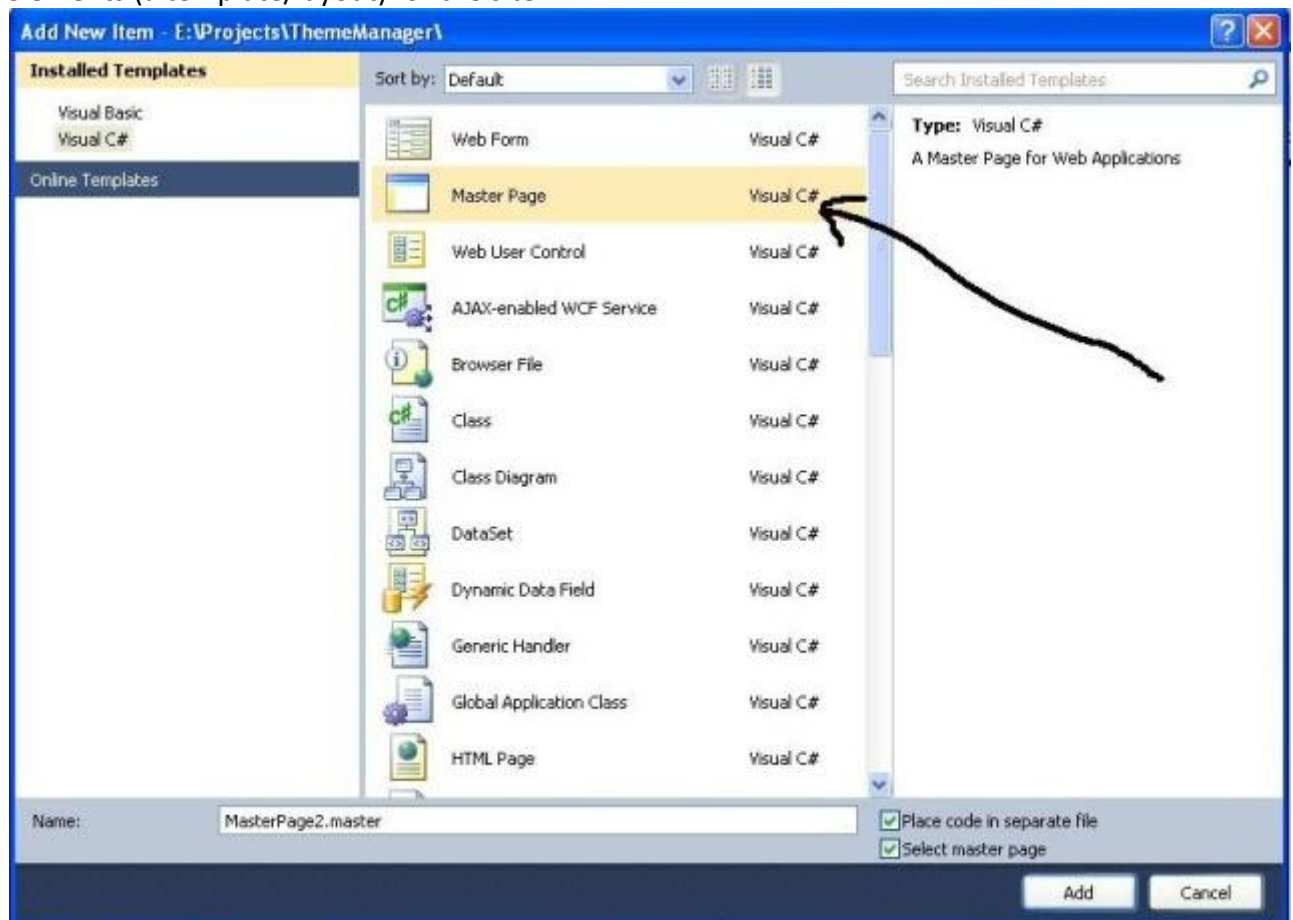
Master Pages in ASP.NET allow you to control the layout, design, and common controls of all pages in a website/web application. With a single master page, you can define the look and feel of your application which includes multiple content place holders. Along with Master Pages, you can work with themes to provide the user with a great User Interface. In this article, we will see an overview of how to:

1. Create a Master Page with single and multiple content place holders
2. Add Content Pages
3. Nested Master Pages
4. Programming the Master Page
5. Creating themes for the application.
6. Finally we will see how we can work with multiple themes.

Creating a Master Page

Creating a master page is very simple. In your website, right click on Solution Explorer, select Add New Item, and then select the Master Page from the pop up window, as shown in the

figure. The Master Page comes with the extension *.master* which consists of all the common elements (a template/layout) for the site.



By default the Master Page comes with two content place holders placed in it. The code for the Master Page is as follows:

[Hide](#) [Copy Code](#)

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

In the above, the @ Master directive identifies it as a master page, whereas all other pages have an @pagedirective.

[Hide](#) [Copy Code](#)

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

    </asp:ContentPlaceHolder>
  </div>
```

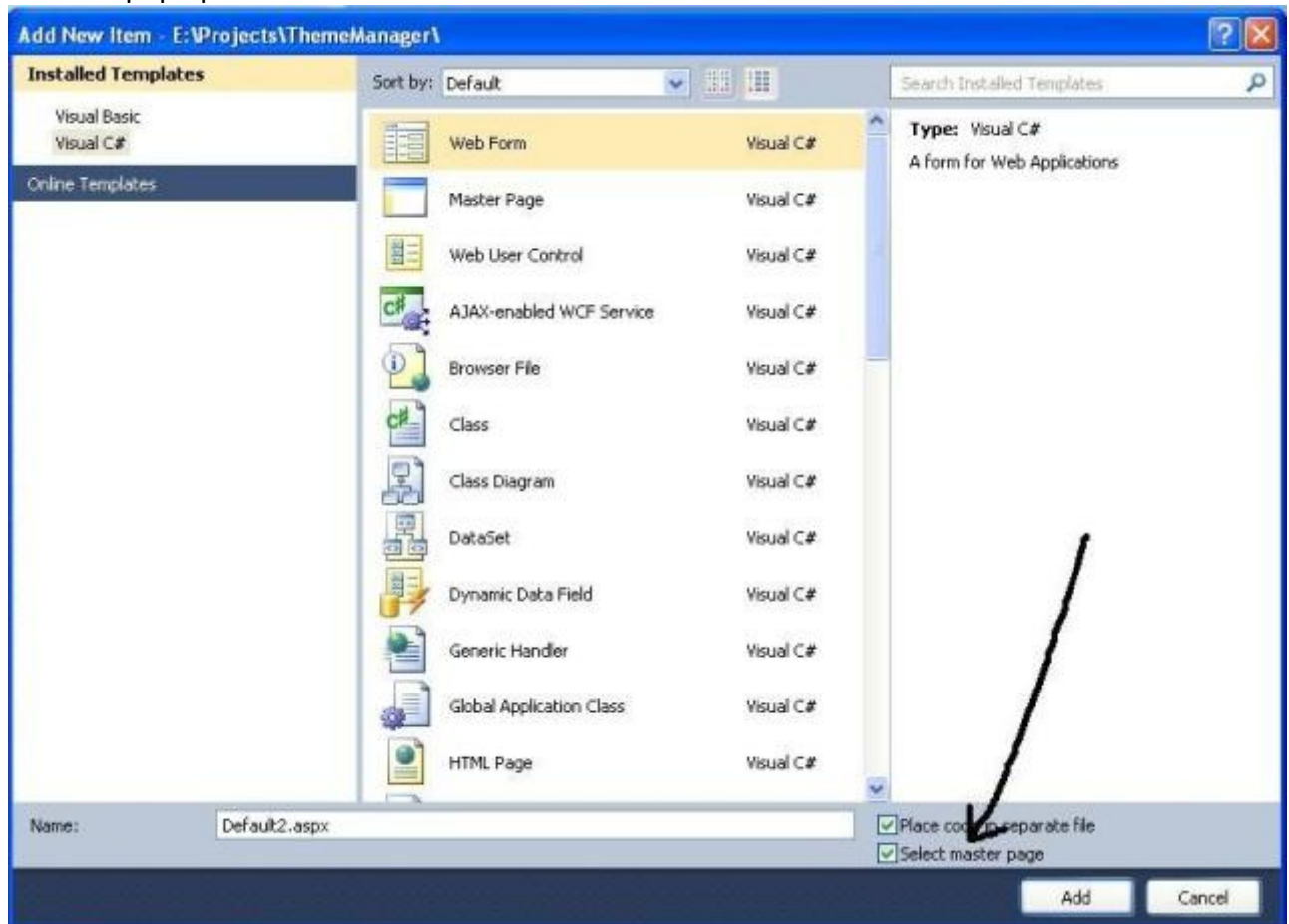
```
</form>
</body>
</html>
```

The content place holders in the above are used to place static content or dynamic content by using ASP.NET server controls in the site. The content place holder is placed in the header section and in the form tags which in general allows users to place page content in the header and body part, but there is no such restriction to place a content placeholder under a particular tag nor is there any restriction on the number of content placeholders to be used. In the latter part of the article, we will see in detail about the layout and usage of the content placeholder.

Content placeholders allow the user to flexibly maintain site content with a common page layout.

Adding a Content Placeholder

We have now created a Master Page, we now need to add a content placeholder to the site. Right click on Solution Explorer, select Add New Item, and select a content place holder from the pop up window:



This adds the content place holder to the site with the following code included in the page:

[Hide](#) [Copy Code](#)

```
<%@ Page Title="" Language="C#" MasterPageFile="~/MasterPage.master"
```

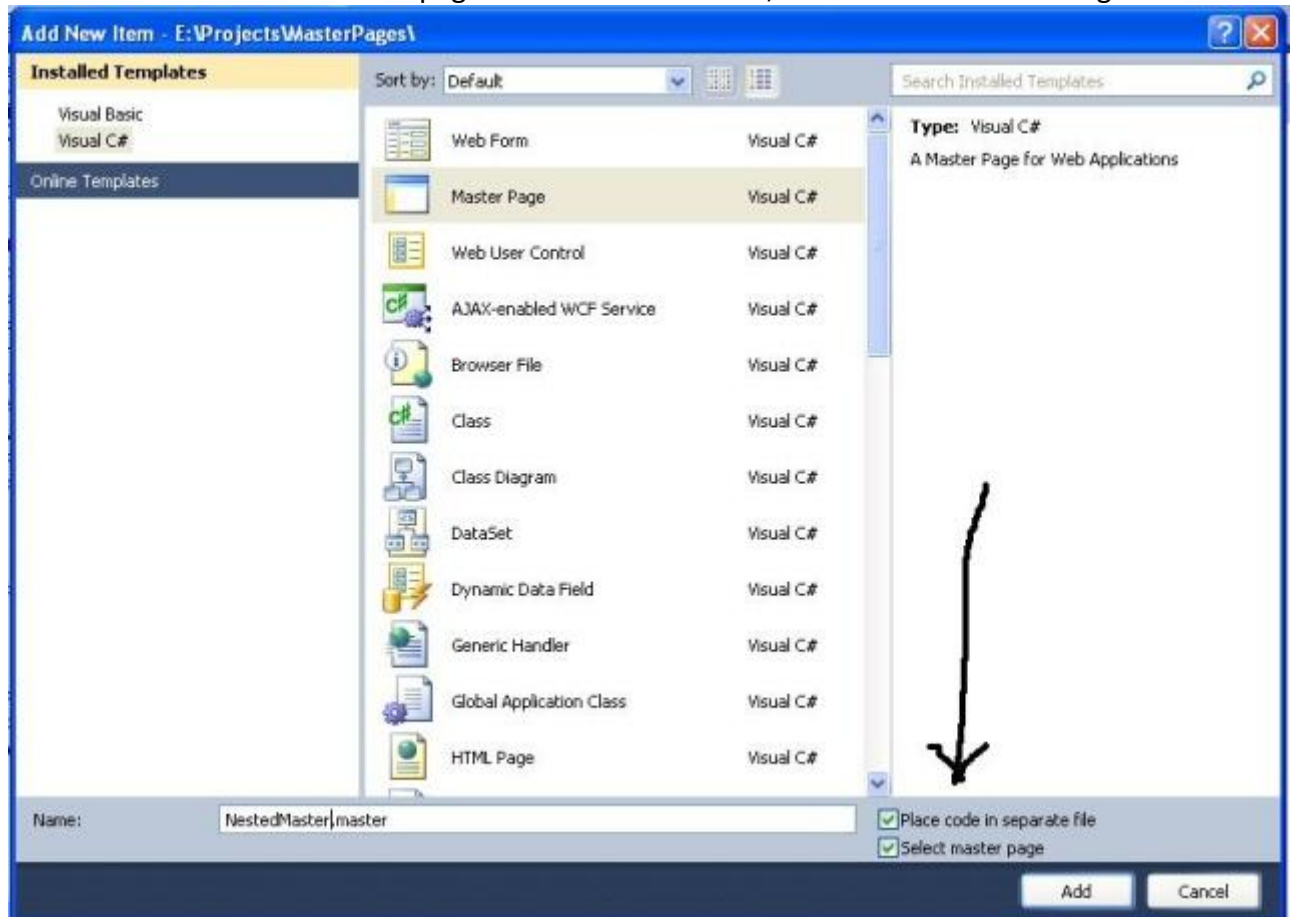
```
AutoEventWireup="true" CodeFile="Default2.aspx.cs" Inherits="Default2" %>
```



```
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
Runat="Server">
</asp:Content>
```

Creating a Nested Master Page

In order to create a nested master page, follow the below steps. Repeat the above steps to add a Master Page and then right click on Solution Explorer and select Add New Item and select Master Page from the installed templates and then select the 'Select Master Page' checkbox and name the master page *nestedmaster.master*, as shown in the below figure.



From the below code, we can observe the differences in the code for a master page, content page, and nested master page.

Master Page

[Hide](#) [Copy Code](#)

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

Nested Master Page:

[Hide](#) [Copy Code](#)

```
<%@ Master Language="C#" MasterPageFile="~/MasterPage.master"
```

```
AutoEventWireup="true" CodeFile="NestedMaster.master.cs" Inherits="NestedMaster"
%>
```

Content Page:

[Hide](#) [Copy Code](#)


```
<%@ Page Title="" Language="C#" MasterPageFile="~/MasterPage.master"
```

```
AutoEventWireup="true" CodeFile="Default2.aspx.cs" Inherits="Default2" %>
```

By this, we have created a nested master page for the site. Now we need to design the child master page.

[Hide](#) [Copy Code](#)

```
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
Runat="Server">
<div>
<h1>Image Slide Show</h1>
<h3>Product's Show Case:</h3>
<asp:Image ID="Image1" runat="server" ImageUrl ="~/Blue hills.jpg" Height="173px"

Width="287px" />
<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
</asp:ContentPlaceHolder>
</div>
</asp:Content>
```

Activity: Developing a Theme Manager Application

Creating the Page Layout

We have already seen how to create a Master Page for our application. We will start now with creating a page layout. We can create a page layout by using different techniques and tools. We can create the Page Layout by using HTML and CSS or by using a web design tool like Photoshop. In our example we will create a page template by using HTML and CSS. We can create a simple two column page layout by using either div tags of HTML or in the Design view just by dragging a table from the toolbox. The latter one you can try by yourself. Here we will design the page layout by using div tags and CSS. The following HTML code shows you a two column page layout and the design effects for the layout are applied by using CSS code as shown below:

HTML Layout

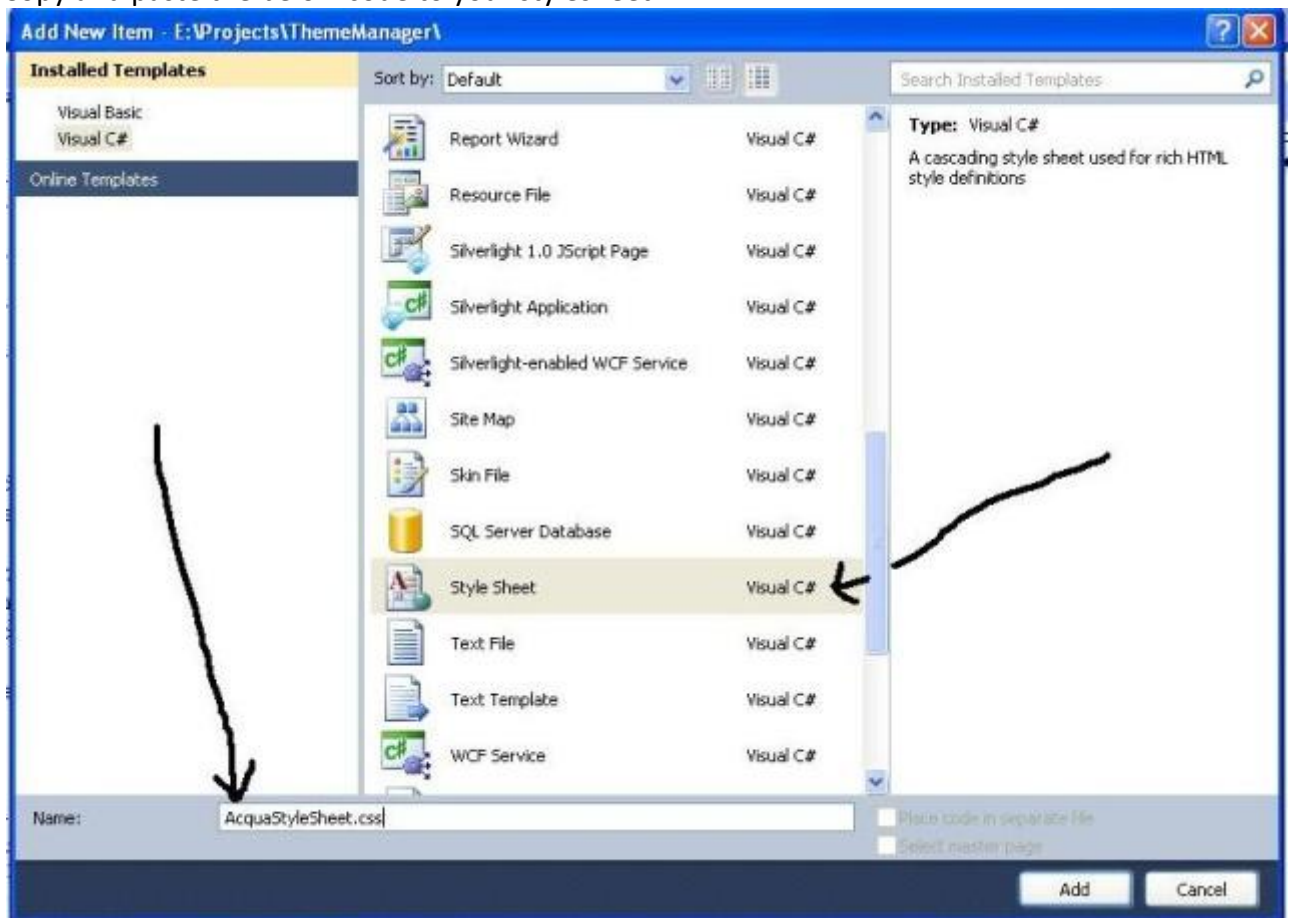
[Hide](#) [Copy Code](#)

```
<div id = "Container">
<div id ="Container">
<div id="header">
</div>
<div id="SideBar">
</div>
<div id="Content">
</div>
<div id="Footer">
</div>
</div>
```

From the div tags, it is clear that now we have a header, side navigation, content, and footer.

1. Now by using the following CSS, we can arrange the divs in the page.

2. Right click on Solution Explorer, select Add New Item, and then select StyleSheet from the popup window.
3. Name the file as 'AcquaStyleSheet.css'.
4. Copy and paste the below code to your stylesheet.



CSS Code

Hide Shrink ▲ Copy Code

```
body
{
margin:20px 20px 20px 20px;
padding: 0;
font-family: Georgia, Times, "Times New Roman", serif;
color: black;
width: 960px;
border-right: 1px solid black;
background-color:#98B3F6;
}

#header
{
background-color:#1B7DCE;
margin:10px 10px 0px 10px;
height:120px;
overflow:hidden;
```

```

}
#SideBar
{
  float: left;
  width: 160px;
  margin-left: 10px;
  padding-top: 1em;
  height:900px;
}

#Content
{
  padding-top: 1em;
  margin: 0 12px 0 180px;
  background-color:White;
  overflow:hidden;
}

#Footer
{
  clear: both;
  background-color: #62645C;
  padding-bottom: 1em;
  border-top: 1px solid #333;
  padding-left: 200px;
}

```

Now moving further, we add some more common features that we will make available through the website, the title, tag-line, and menu.

Title and Tag Line

Add the following HTML code inside the header div tag which displays the title and tag line of your site.

[Hide](#) [Copy Code](#)

```

<h1 class ="Title">Mysite Title</h1>
<span class ="TagLine">Mysite Tag Lline</span>

```

and the CSS code goes here:

[Hide](#) [Copy Code](#)

```

#header .Title
{
  color:White;
}
#header h1
{
  margin:10px 40px 10px 40px;
}

#header .TagLine

```

```
{
    color:White;
    margin:40px 40px 10px 70px;
}
```

Creating the Navigation and Menu for the Site

ASP.NET provides a wide range of controls for site navigation, you can directly use them for your site or use the simple HTML tags to design the navigation for your site.

Designing the Menu

[Hide](#) [Copy Code](#)

```
<ul class ="Menu">
<li><a href ="#">Home</a></li>
<li><a href ="#">AboutMe</a></li>
<li><a href ="#">Articles</a></li>
<li><a href ="#">ContactUs</a></li>
</ul>
```

CSS Code

[Hide](#) [Copy Code](#)

```
.Menu li
{
    display:inline;
    margin:0px;
}

.Menu a
{
    text-decoration:none;
    background-color:Blue;
    padding:5px;
    color:White;
    border-right:4px solid Maroon;
    margin:0px;
    border-left:none;
}

.Menu a:hover
{
    background-color:Maroon; padding:5px;
    border-right:4px solid Blue; margin:0px;
}
```

Finally, you can also apply the CSS style for the Grid control. The CSS code shown here can be applied to the ASP.NET Grid control.

[Hide](#) [Shrink](#) ▲ [Copy Code](#)

```
.gridAlternatingRowStyle
{
    background-color:White;
```

```

}
.gridEditRowStyle
{
    background-color:#2461BF;
}
.gridFooterStyle
{
    background-color:#98B3F6;
    color:White;
    font-weight:bold;
}
.gridHeaderStyle
{
    background-color:#1B7DCE;
    font-weight:bold;
    color:White;
}
.gridPagerStyle
{
    background-color:#98B3F6;
    color:White;
    text-align:center;
    margin:20px;
    border:1px solid black;
    background-image:url(Images/abc.gif);
}
.gridRowStyle
{
    background-color:#EFF3FB;
}

.gridSelectedRowStyle
{
    background-color:#D1DDF1;
    color:#333333;
}
.grid
{
    border:none;
    width:100%;
}

```

The GridView code goes here:

[Hide](#) [Copy Code](#)

```

<EditRowStyle CssClass ="gridEditRowStyle" />
<FooterStyle CssClass="gridFooterStyle" />
<HeaderStyle CssClass ="gridHeaderStyle" />

```

```
<PagerStyle CssClass="gridPagerStyle" />
<RowStyle CssClass="gridRowStyle" />
<SelectedRowStyle CssClass="gridSelectedRowStyle" />
```

Using a Skin File to Apply Style

Using a skin file you can apply styles to server controls. The following samples show the usage of a skin file. The code in the skin file is the same as the server control syntax except that the ID attribute is eliminated for the control.

[Hide](#) [Copy Code](#)

```
<asp:TextBox ID="TextBox1" runat="server" BorderColor="#FFFFB7" Width="288px">
</asp:TextBox>
```

```
<asp:Button ID="Button1" runat="server" BackColor="#FF9933"

    ForeColor="White" Text="Button" Width="119px" />
```

```
<asp:GridView ID="GridView1" runat="server" CellPadding="4"

    ForeColor="#333333" GridLines="None">
    <AlternatingRowStyle BackColor="White" />
    <FooterStyle BackColor="#990000" Font-Bold="True" ForeColor="White" />
    <HeaderStyle BackColor="#1B7DCE" Font-Bold="True" ForeColor="White" />
    <PagerStyle BackColor="#98B3F6" ForeColor="#333333" HorizontalAlign="Center" />
    <RowStyle BackColor="#EFF3FB" ForeColor="#333333" />
</asp:GridView>
```

Adding Some Styles for the Home Page (Content Pages)

As we have seen how to use skin files, now in our application we will add a Feedback form using the above skin file styles. The above style defined for textbox and button will be applied to all controls in our feedback form. But if we want to be more specific, for each individual textbox, we can use the SkinID property in the skin file for the controls as shown below: coming back again, we'll define some more styles in the CSS page for "Quotes" and "Images". The following code shows code for displaying images.

[Hide](#) [Copy Code](#)

```
.imagecells img
{
    width:140px;
    height:100px;
    border:2px solid Gray;
    padding:15px;
    margin:15px;
    background-color:#98B3F6;
    border-style:inset;
}
```

The following CSS can be used to display quotations.

[Hide](#) [Copy Code](#)

```
.quotehomepage blockquote {
```

```

        font: italic 1em/1.6em Georgia, "Times New Roman", Times, serif;
        width: 300px;
        background: url(/ThemeManager/Images/closequote.gif) no-repeat right bottom;
        padding-left: 18px;
        text-indent: -18px;
        float: right;
        margin: 20px 0 10px 10px;
        color: #999999;
    }
    .quotehomepage blockquote:first-letter {
        background: url(/ThemeManager/Images/openquote.gif) no-repeat left top;
        padding-left: 18px;
        font: italic 1.4em Georgia, "Times New Roman", Times, serif;
    }

```

We can define styles for the span tag which can be used to highlight the elements:

[Hide](#) [Copy Code](#)

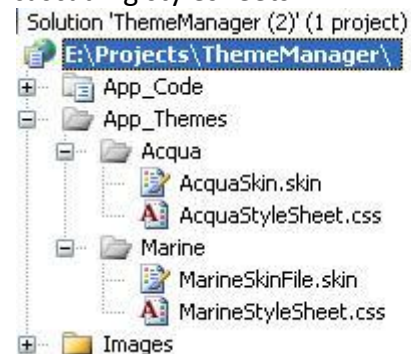
```

.welcome span
{
    color:#98B3F6;
}

```

Themes in ASP.NET

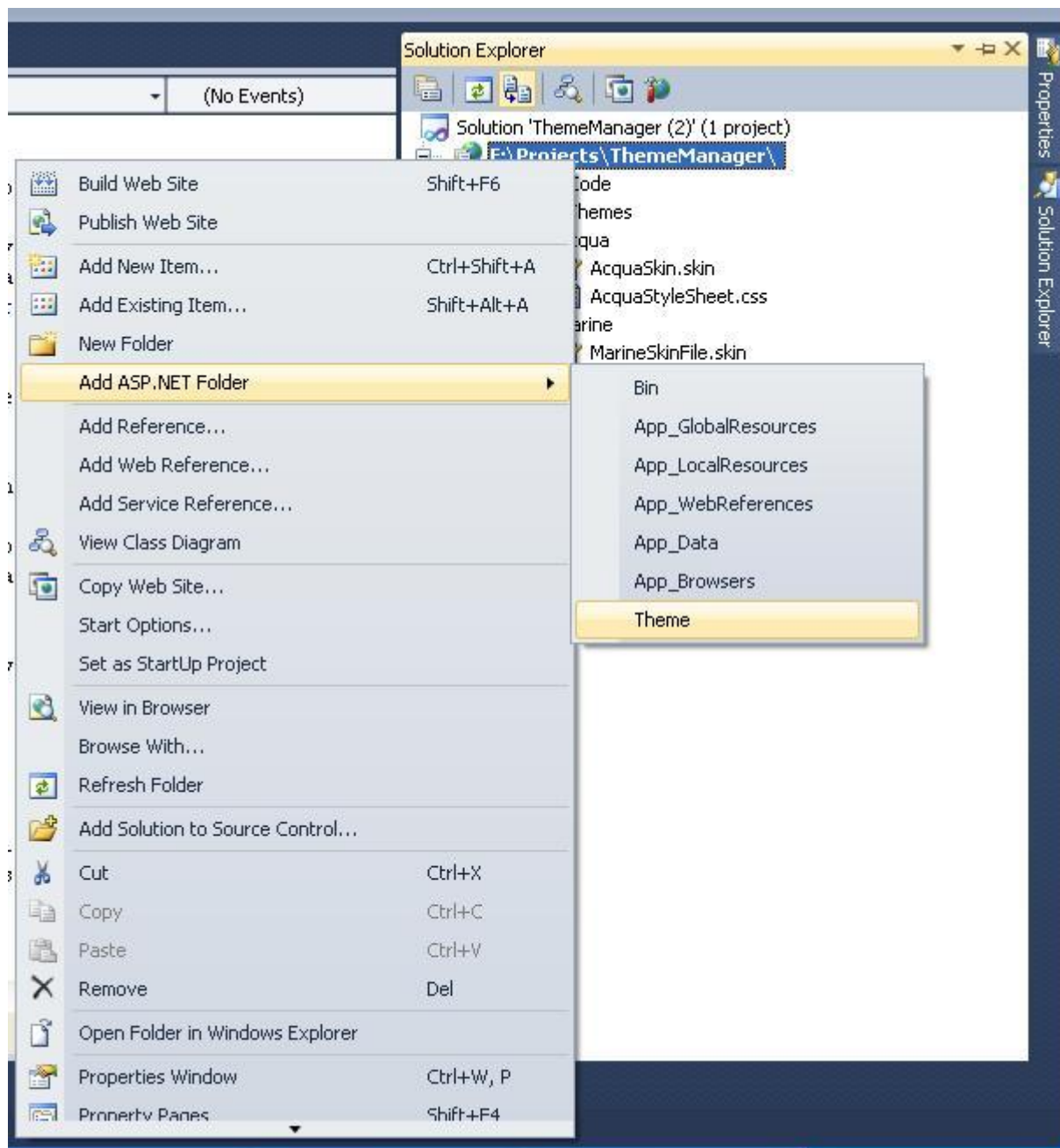
The styles we have created till now can be applied as a theme for the application. We can create more CSS styles as the one we have created above and we can allow the user to select one. ASP.NET provides you with a flexible way to achieve this. Follow the below steps to create themes in your application: The theme folder consists of the skin files and cascading stylesheets.



As we have seen how to create stylesheets and skin files, now we will see how to use them for themes. The skin files allow you to apply styles same as the cascading style for the application. You can apply styles to the site by either using a CSS stylesheet or a skin file or both. Mainly you can use the skin file to apply styles to server controls.

Adding the Themes Folder

1. Right click on Solution Explorer and select 'Add ASP.NET Folder' and then select 'Theme'. This adds the theme folder to the application.
2. In *App_Theme*, add a new theme folder and you can give your own name for the folder.
3. In the theme folder, add CSS and skin files.
4. By repeating the above steps, you create as many themes for the application.



Theme Settings

To apply the theme for the application, apply the following settings in the *web.config*.

[Hide](#) [Copy Code](#)

```
<pages theme ="Acqua">
</pages>
```

Page Level Settings

To apply the theme settings at page level, set the theme attribute to the theme name in the @ page directory as shown:

[Hide](#) [Copy Code](#)

```
<%@ Page Title="" Language="C#" MasterPageFile="~/MasterPage.Master" Theme ="Acqa"
```

Dynamic Themes

Before I could discuss about the code for managing dynamic themes, I want you to refer to this excellent article about dynamic themes in CodeProject: Dynamic themes. Here we'll see an alternate method to achieve the same: the following code shows you how to select dynamic themes from the available themes:

DropDown control

Hide Copy Code

```
<asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="True"
onselectedindexchanged="DropDownList1_SelectedIndexChanged">
  <asp:ListItem Value="Select Any Theme">Select Any Theme</asp:ListItem>
  <asp:ListItem Value="Acqua">Acqua</asp:ListItem>
  <asp:ListItem Value="Marine">Marine</asp:ListItem>
</asp:DropDownList>
```

C# code:

Hide Copy Code

```
protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
{
    Configuration config = WebConfigurationManager.OpenWebConfiguration("~/");
    //Get the required section of the web.config file by using configuration object.
    PagesSection pages = (PagesSection)config.GetSection("system.web/pages");
    //Update the new values.
    pages.Theme = DropDownList1.SelectedItem.Text.ToString();
    //save the changes by using Save() method of configuration object.
    if (!pages.SectionInformation.IsLocked)
    {
        config.Save();
        Response.Redirect("Default.aspx");
    }
    else
    {
        Response.Write("<script>alert('Could not save configuration')</script>");
    }
}
```

- **Localization:**

Implicit & Explicit localization

Implicit localization

Implicit localization is used within your markup, and its main advantage is that it allows you to localize several properties of the same control/object, without explicitly referencing each property. In our Localized Hello World chapter, we used it, and saw how we could set the meta:resourcekey property of a control, and then automatically have its Text property mapped to the resource row in our resource file. Let's expand a bit on that example, to have several properties localized. If you haven't already done so, you should read the Localized Hello World chapter now and create the project from it, as the following example uses it:

```
<asp:Label runat="server" ID="lblHelloWorld" Text="Hello, world!" Font-Names="Verdana"
ForeColor="Blue" meta:resourcekey="lblHelloWorld" />
```

In the three resource files we created (default, German and Spanish), you can now add two extra rows to each of them, with the names "lblHelloWorld.ForeColor" and "lblHelloWorld.Font-Names", and then define different values for it. For instance, the default label color could be blue, the German version could be green and the Spanish version could be red, and you could define different font names for each of them as well. This makes it really easy to localize e.g. controls, because we only have to define the name of the resource row - each property is automatically mapped.

Explicit localization:

With explicit localization, you pick out a specific resource row from your resource file, which is returned to you, and while implicit localization can be helpful when localizing webcontrols and other declarative objects, explicit localization is the only possible way of doing things in any other circumstance. Let's see how the above example would look with explicit localization:

```
<asp:Label runat="server" ID="lblHelloWorld2" Text="<%$ Resources:lblHelloWorld.Text %>" Font-Names="<%$ Resources:lblHelloWorld.Font-Names %>" ForeColor="<%$ Resources:lblHelloWorld.ForeColor %>" />
```

We simply re-use the resource rows from our previous example, and as you can see, the markup for explicit localization is a bit more complicated and verbose than for implicit localization. The syntax for retrieving a resource row is like this:

```
<%$ Resources:[resource class name,]RowName %>
```

As you can see, we didn't specify a resource class name in our example, because we use a local resource. If you use a global resource, you should specify the name of the class it results in, which as a rule of thumb is the same as the filename, but without the .resx extension. So for instance, if you have a global resource file with the name of MyGlobalResources.resx, you would obtain a resource from it like this:

```
<%$ Resources: MyGlobalResources,NameOfRow %>
```

- **Multiple Languages:**

Step 1:

Start a new web site with C# and give it the Name multilingual. It will give you a Default.aspx and Default.aspx.cs two files.

Step 2:

Now Design the form by putting three labels and three buttons on the form; set the id for the label as well as the button.

Step 3:

Now go to Solution Explorer window and add a "App_GlobalResources" folder to your project by right clicking and selecting "Add Asp.Net Folder" option. It will add "App_GlobalResources" folder to your project. Now here right click and add Resource File and give it the name "Strings.resx"; it will add a resource file.

Step 4:

Open that "String.resx" file and add a name to it and a value that you want on the first page load. We will give some English values here for the first time page load. Here we will add some fields like AboutMe, Desc, Header, Eng, Hindi, Marathi and also give some default values like About Me, Hi, Localization In Asp.Net and the two values for Hindi and Marathi taken from gmail option.

Step 5:

Like in Step 4, create three more files for English, Hindi and Marathi and give the name with culture code like Strings.en-US.resx, Strings.hi-IN.resx, Strings.mr-IN.resx respectively and add same name and the different values with respect to language.

Step 6:

Now open the code file of Default page and import the namespaces given bellow.

```
using System.Globalization;
using System.Resources;
using System.Threading;
using System.Reflection;
```

Step 7:

Now in global declaration section of .cs file create an instance of ResourceManager and CultureInfo Class.

```
ResourceManager rm;
CultureInfo ci;
```

Step 8:

In Page_Load Event write the following code to set default locale for our page i.e. strings.resx. as follows. As well call our resource file values in LoadString method which will take cultureinfo as parameter.

```
if (!IsPostBack)
{
    Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
    rm = new ResourceManager("Resources.Strings",
```

```

System.Reflection.Assembly.Load("App_GlobalResources"));
    ci = Thread.CurrentThread.CurrentCulture;
    LoadString(ci);
}
else
{
    rm = new ResourceManager("Resources.Strings",
System.Reflection.Assembly.Load("App_GlobalResources"));    ci
=Thread.CurrentThread.CurrentCulture;
    LoadString(ci);
}
private void LoadString(CultureInfo ci)
{
    lblabtme.Text = rm.GetString("AboutMe",ci);
    lbldesc.Text = rm.GetString("Desc",ci);
    Button1.Text = rm.GetString("Eng",ci);
    lblheader.Text = rm.GetString("Header", ci);
    Button2.Text = rm.GetString("Hindi",ci);
    Button3.Text = rm.GetString("Marathi",ci);
}

```

Step 9:

Create click events for button for English, Hindi And Marathi and write the code/change the cultureinfo on by passing specific cultureinfo for resourcemanager.

In English_Click Event Write This Code.

```

Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
LoadString(Thread.CurrentThread.CurrentCulture);

```

This will change our asp.net page in Standerd En-US language.

In Hindi_Click event write this code which will change our page into hindi language.

```

Thread.CurrentThread.CurrentCulture = new CultureInfo("hi-IN");

LoadString(Thread.CurrentThread.CurrentCulture);

```

And finally in Marathi_Click event write code to change the page in Marathi language.

```

Thread.CurrentThread.CurrentCulture = new CultureInfo("mr-IN");
LoadString(Thread.CurrentThread.CurrentCulture);

```

This way you are able to change the page language of your Asp.net page. The following is a table of langauges and their respective codes.

Table For Language And Code For India:

English-> en-US

Hindi-> hi-IN

Marathi-> mr-IN

Telugu-> te-IN

Gujrati-> gu-IN

Panjabi-> pa-IN

Malayalam-> ml-IN

Oriya-> or-IN

Tamil-> ta-IN

Kannad-> kn-IN

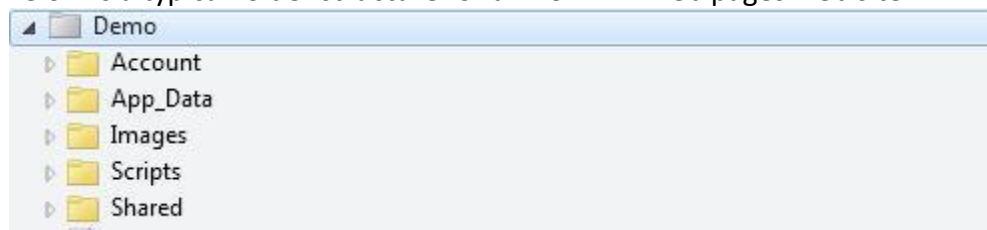
Conclusion:

In this way you can create your multilingual website in ASP.Net.

ASP.Net Folder Structure:

Logical Folder Structure

Below is a typical folder structure for an ASP.NET web pages web site:



- The "Account" folder contains logon and security files
- The "App_Data" folder contains databases and data files
- The "Images" folder contains images
- The "Scripts" folder contains browser scripts
- The "Shared" folder contains common files (like layout and style files)

- **Physical Folder Structure**

The physical structure for the "Images" folder at the website above might look like this on a computer:

C:\Johnny\Documents\MyWebSites\Demo\Images

Virtual and Physical Names

From the example above:

The virtual name of a web picture might be "Images/pic31.jpg".

But the physical name is "C:\Johnny\Documents\MyWebSites\Demo\Images\pic31.jpg"

URLs and Paths

URLs are used to access files from the web: https://www.w3schools.com/html/html5_intro.asp

The URL corresponds to a physical file on a server:

C:\MyWebSites\w3schools\html\html5_intro.asp

A virtual path is shorthand to represent physical paths. If you use virtual paths, you can move your pages to a different domain (or server) without having to update the paths.

URL	<code>https://www.w3schools.com/html/html5_intro.asp</code>
-----	---

Server name	<code>w3schools</code>
-------------	------------------------

Virtual path	<code>/html/html5_intro.asp</code>
--------------	------------------------------------

Physical path	<code>C:\MyWebSites\w3schools\html\html5_intro.asp</code>
---------------	---

The root on a disk drive is written like C:\, but the root on a web site is / (forward slash).

The virtual path of a web folder is (almost) never the same as the physical folder.

In your code you will, reference both the physical path and the virtual path, depending on what you are coding.

ASP.NET has 3 tools for working with folder paths: the ~ operator, the Server.MapPath method, and the Href method.

The ~ Operator

To specify the virtual root in programming code, use the ~ operator.

If you use the ~ operator, instead of a path, you can move your website to a different folder or location without changing any code:

```
var myImagesFolder = "~/images";  
var myStyleSheet = "~/styles/StyleSheet.css";
```

The Server.MapPath Method

The Server.MapPath method converts a virtual path (/default.cshtml) to a physical path that the server can understand (C:\Johnny\MyWebSited\Demo\default.cshtml).

You will use this method when you need to open data files located on the server (data files can only be accessed with a full physical path):

```
var pathName = "~/dataFile.txt";  
var fileName = Server.MapPath(pathName);
```

You will learn more about reading from (and writing to) data files on the server in the next chapter of this tutorial.

The Href Method

The Href method converts a path used in the code to a path that the browser can understand (the browser cannot understand the ~ operator).

You use the Href method to create paths to resources like image files, and CSS files.

You will often use this method in HTML <a>, , and <link> elements:

```
@{var myStyleSheet = "~/Shared/Site.css";}
```

```
<!-- This creates a link to the CSS file. -->
```

```
<link rel="stylesheet" type="text/css" href="@Href(myStyleSheet)" />
```

```
<!-- Same as : -->
```

```
<link rel="stylesheet" type="text/css" href="/Shared/Site.css" />
```

The Href method is a method of the WebPage Object.

- **Comparison:**

HTML Controls	ASP.Net Controls
HTML control runs at client side .	ASP.Net controls run at server side .
You can run HTML controls at server side by adding attribute runat="server" .	You can not run ASP.Net Controls on client side as these controls have this attribute runat="server" by default.
HTML controls are client side controls, so it does not provide STATE management .	ASP.Net Controls are Server side controls, provides STATE management.
HTML control does not require rendering.	ASP.Net controls require rendering .
As HTML controls runs on client side, execution is fast .	As ASP.Net controls run on server side, execution is slow .
HTML controls do not have any separate class for its controls.	ASP.Net controls have separate class for its each control.
HTML controls does not support Object Oriented paradigm .	With ASP.Net controls, you have full support of Object oriented paradigm.
HTML controls can not be accessed form code behind files.	ASP.Net controls can be directly worked and accessed from code behind files .
HTML control have limited set of properties and/or methods.	ASP.Net controls have rich set of properties and/or methods.
?	?
1 Enter Name :	1 Enter Name :
2 <input type="text" ID="txtName">	2 <asp:TextBox Id="txtName" runat="server">
	3 </asp:TextBox>

- **SQL DataSource Control:**

SqlDataSource control is a data source control that is used to connect to a relational database. You can even connect to Oracle or any other data source that can be accessible through OLEDB and ODBC. Connection String and Provider Name are the two properties that is used to connect the database. The first one specify the connection string and second specify the provider name. By default the provider is System.Data.SqlClient, it means connect to Sql Server database.

Following are some important properties that are very useful.

Properties for Configuring Data Source	
InsertCommand, InsertParameters, InsertCommandType	Gets or sets the SQL Statement, parameter and type of command (text or stored procedure) to insert a record.
DeleteCommand, DeleteParameters,	Gets or sets the SQL Statement, parameters and type of the command (text or stored procedure) to delete a record.

DeleteCommandType	
UpdateCommand, UpdateParameters, UpdateCommandType	Gets or sets the SQL Statement, parameters and type of the command (text or stored procedure) to update a record.
SelectCommand, SelectParameters, SelectCommandType	Gets or sets the SQL Statement, parameters and type of the command (text or stored procedure) to select record(s).
Parameter Types of DataSource Controls	
ControlParameter	Gets the value from any public property of the server control.
FormParameter	Gets the value of any input field from the form.
QueryStringParameter	Gets the value from specified querystring.
Parameter	Gets the parameter value assigned in the code.
Other Properties of SqlDataSource Control	
ProviderName	Gets or sets the .NET ProviderName name.
DataSourceMode	DataSet/DataReader. Used to specify the data source mode. In case of DataReader paging is not supported in the target control (GridView etc.).
ConnectionString	Gets or sets the connection string to connect to the database.
ConflictDetection	CompareAllValues/OverWriteChanges. Used to determine how conflict will be handled in case of updation or deletion of the records.
Caching Properties of SqlDataSource Control	
EnableCaching	true/false. Used to indicate whether enable caching or not.
CacheDuration	Used to indicate number of seconds the data should be maintained in the cache.
CacheExpirationPolicy	Absolute/Sliding. Used to indicate if the cache policy is absolute or sliding. Absolute: The data is removed after specified duration. Sliding: The data is removed if it is not used for specified duration.

EnableCaching	true/false. Used to indicate whether enable caching or not.
---------------	---

- **List Controls:**

ASP.NET provides the following list controls.

- Drop-down list
- List box
- Radio button list
- Check box list
- Bulleted list

These controls display list of options to select. You can select one or more options, the choice depends upon control. They all derive from the `System.Web.UI.WebControls.ListControl` class

Some of the important common properties of list controls are as follows:

- **SelectedValue:** Get the value of the selected item from the dropdown list.
- **SelectedIndex:** Gets the index of the selected item from the dropdown box.
- **SelectedItem:** Gets the text of selected item from the list.
- **Items:** Gets the collection of items from the dropdown list.
- **DataTextField:** Name of the data source field to supply the text of the items. Generally this field came from the datasource.
- **DataValueField:** Name of the data source field to supply the value of the items. This is not visible field to list controls, but you can use it in the code.
- **DataSourceID:** ID of the datasource control to provide data.

There are several ways through which you can populate these controls such as:

- By using data from database.
- Directly write code to add items.
- Add items through the items collection from property window.
- Write HTML to populate these controls.
- Use inbuilt datasource controls.

1)DropDownList control

DropDownList control is used select single option from multiple listed items.

Example

```
using System;
using System.Collections.Generic;
public partial class ListControls : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            List<string> cityList = new List<string>();
            cityList.Add("Pune");
            cityList.Add("Kanpur");
            cityList.Add("Jaipur");
        }
    }
}
```

```

        cityList.Add("Delhi");
        cityList.Add("Agra");
        DropDownList1.DataSource = cityList;
        DropDownList1.DataBind();
    }
}
protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
{
    Label1.Text = DropDownList1.SelectedItem.Text;
}
}

```

Select the item from the DropDownList, the label control will display the selected item. Please keep in mind that, you should write code with association with IsPostBack property otherwise you will get the first item of the DropDownList.

You can also add items directly to write HTML as follows:

```

<asp:DropDownList ID="DropDownList2" runat="server">
    <asp:ListItem Value="1">India</asp:ListItem>
    <asp:ListItem Value="2">USA</asp:ListItem>
    <asp:ListItem Value="2">Australia</asp:ListItem>
    <asp:ListItem Value="4">Canada</asp:ListItem>
    <asp:ListItem Value="5">Newzealand</asp:ListItem>
</asp:DropDownList>

```

2)ListBox control

The ListBox control is similar to the DropDownList but main difference is that you can select multiple items from ListBox at a time. ListBox control has SelectionMode property that enables you to select multiple items from ListBox control. By default SelectionMode property is set as single. If you want to select multiple items from the ListBox, then set SelectionMode property value as Multiple and press Ctrl or Shift key when clicking more than one list item.

The ListBox control also supports data binding. You can bind the ListBox through coding with database or attach it with one of the predefined DataSourceControl objects, which contains the items to display in the control. DataTextField and DataValueField properties are used to bind to the Text and Value field in the data source.

Example

```

using System;
using System.Collections.Generic;
using System.Text;
using System.Web.UI.WebControls;
public partial class ListControls : System.Web.UI.Page
{
    List<string> empList;
    protected void Page_Load(object sender, EventArgs e)

```

```

{
    if (!IsPostBack)
    {
        empList = new List<string>();
        empList.Add("Raj");
        empList.Add("Rajesh");
        empList.Add("John");
        empList.Add("Elina");
        empList.Add("Samy");
        empList.Add("Reena");
        ListBox1.DataSource = empList;
        ListBox1.DataBind();
    }
}

protected void Button1_Click(object sender, EventArgs e)
{
    StringBuilder sb = new StringBuilder();
    foreach(ListItem item in ListBox1.Items)
    {
        if(item.Selected)
        {
            sb.Append(item + "<br>");
        }
    }
    Label1.Text = sb.ToString();
}
}

```

Execute the above program and select the items from ListBox. Click on show button, you will get the selected items.

The ListBox control has a SelectedIndexChanged event handler. If AutoPostBack property is set to true, then this event is raised whenever you select new item from the List control.



3) RadioButtonList Control

RadioButtonList Control is same as DropDownList but it displays a list of radio buttons that can be arranged either horizontally or vertically. You can select only one item from the given RadioButtonList of options. These options are mutually exclusive.

The RadioButtonList control supports three important properties that affect its layout:

RepeatColumns: It displays the number of columns of radio buttons.

RepeatDirection: The direction that the radio buttons repeat. By default RepeatDirection value is vertical. Possible values are Horizontal and Vertical.

RepeatLayout: Determines whether the radio buttons display in an HTML table.

Possible values are as follows:

- Table
- Flow
- OrderedList
- UnorderedList

Example

```
using System;
using System.Web.UI.WebControls;
public partial class ListControls : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Button1_Click(object sender, EventArgs e)
```

```

{
    Label1.Text = "You have selected </br> Item=" +
        RadioButtonList1.SelectedItem.Text + "</br> Value =" +
        RadioButtonList1.SelectedValue + "</br> Index =" +
        RadioButtonList1.SelectedIndex ;
}
protected void Button2_Click(object sender, EventArgs e)
{
    if (RadioButtonList1.RepeatDirection == RepeatDirection.Vertical)
    {
        RadioButtonList1.RepeatDirection = RepeatDirection.Horizontal;
    }
    else
    {
        RadioButtonList1.RepeatDirection = RepeatDirection.Vertical;
    }
}
}

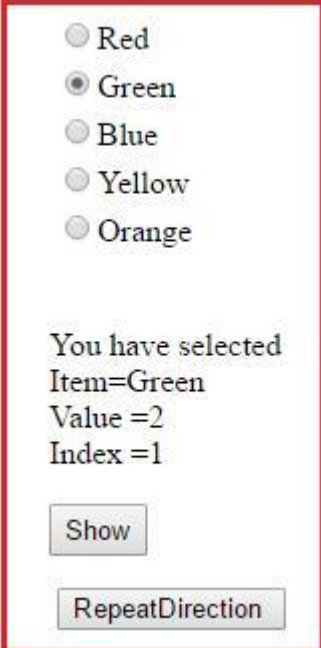
```

You can add items in RadioButtonList through item collection using property window.

```

<asp:RadioButtonList ID="RadioButtonList1" runat="server">
    <asp:ListItem Value="1">Red</asp:ListItem>
    <asp:ListItem Value="2">Green</asp:ListItem>
    <asp:ListItem Value="3">Blue</asp:ListItem>
    <asp:ListItem Value="4">Yellow</asp:ListItem>
    <asp:ListItem Value="5">Orange</asp:ListItem>
</asp:RadioButtonList>

```



☐ Red
☒ Green
☐ Blue
☐ Yellow
☐ Orange

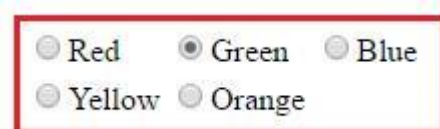
You have selected
 Item=Green
 Value =2
 Index =1

When you click on the RepeatDirection button the layout will be changed as Horizontal.

```
RadioButtonList1.RepeatDirection = RepeatDirection.Horizontal;
```



By default RepeatColumns value is zero. You can change this value according to application need. If you set this value as three then output will be changed as:



4) CheckBoxList Control

CheckBoxList is generally used, when you want to select one or more options from given several choices. Most of the properties are same as RadioButtonList control, but the main difference is that you can select more than one item from CheckBoxList control.

The CheckBoxList control is easier for use, when you have set of options of checkboxes.

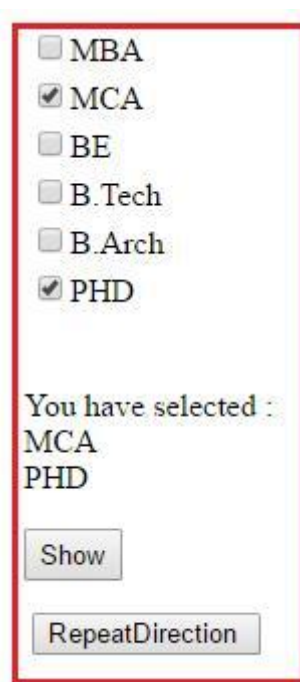
Example

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Web.UI.WebControls;
public partial class ListControls : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            List<string> educationList = new List<string>();
            educationList.Add("MBA");
            educationList.Add("MCA");
            educationList.Add("BE");
            educationList.Add("B.Tech");
            educationList.Add("B.Arch");
            educationList.Add("PHD");
            CheckBoxList1.DataSource = educationList;
            CheckBoxList1.DataBind();
        }
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        StringBuilder sb = new StringBuilder();
        foreach (ListItem item in CheckBoxList1.Items)
```

```

{
    if (item.Selected)
    {
        sb.Append("</br>" + item);
    }
}
Label1.Text = "You have selected : " + sb.ToString();
}
protected void Button2_Click(object sender, EventArgs e)
{
    if (CheckBoxList1.RepeatDirection == RepeatDirection.Vertical)
    {
        CheckBoxList1.RepeatDirection = RepeatDirection.Horizontal;
    }
    else
    {
        CheckBoxList1.RepeatDirection = RepeatDirection.Vertical;
    }
}
}

```



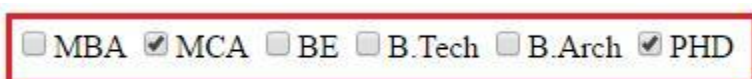
☐ MBA
☒ MCA
☐ BE
☐ B.Tech
☐ B.Arch
☒ PHD

You have selected :
MCA
PHD

Show

RepeatDirection

When you click on the RepeatDirection button the layout will be changed as Horizontal. By default, RepeatDirection is set to RepeatDirection.Vertical.



☐ MBA ☒ MCA ☐ BE ☐ B.Tech ☐ B.Arch ☒ PHD

5) BulletedList Control

BulletedList control is very rich in displaying the items in different styles. It displays the list either in unordered or ordered list. Each list item can be rendered as plain text, a LinkButton control, or a link to another web page.

BulletedList control supports the BulletStyle property. The default value of BulletStyle property is NotSet and rendered as in list of bulleted items. **Possible values are as follows:**

- Circle
- CustomImage
- Disc
- LowerAlpha
- LowerRoman
- NotSet
- Numbered
- Square
- UpperAlpha
- UpperRoman

BulletedList control also supports the DisplayMode property that is used to modify the appearance of list items. **Possible values are as follows:**

- HyperLink
- LinkButton
- Text

Example

```
using System;
using System.Collections.Generic;
using System.Web.UI.WebControls;
public partial class ListControls : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            List<string> educationList = new List<string>();
            educationList.Add("MBA");
            educationList.Add("MCA");
            educationList.Add("BE");
            educationList.Add("B.Tech");
            educationList.Add("B.Arch");
            educationList.Add("PHD");
            BulletedList1.DataSource = educationList;
            BulletedList1.DataBind();
        }
    }
    protected void Style_Command(object sender, CommandEventArgs e)
    {
        switch (e.CommandName)
        {
```



```

case "Circle":
    BulletedList1.BulletStyle = BulletStyle.Circle;
    break;
case "Disc":
    BulletedList1.BulletStyle = BulletStyle.Disc;
    break;
case "Numbered":
    BulletedList1.BulletStyle = BulletStyle.Numbered;
    break;
case "Square":
    BulletedList1.BulletStyle = BulletStyle.Square;
    break;
case "LowerRoman":
    BulletedList1.BulletStyle = BulletStyle.LowerRoman;
    break;
case "UpperAlpha":
    BulletedList1.BulletStyle = BulletStyle.UpperAlpha;
    break;
}
}
}

```

Output:



When you click on different button, the button style of list control will be changed accordingly. In this example we have used Command Button concept. If you want to learn Command Button concept, please read chapter three.

Style = LowerRoman.

- i. MBA
- ii. MCA
- iii. BE
- iv. B.Tech
- v. B.Arch
- vi. PHD

Style = Numbered.

1. MBA
2. MCA
3. BE
4. B.Tech
5. B.Arch
6. PHD

Style = Square.

- MBA
- MCA
- BE
- B.Tech
- B.Arch
- PHD

- **GridView:**

What is a GridView ?

The GridView control is a feature rich and versatile control used to accept, display, and edit data on a web page. It is a commonly used control in ASP.Net web applications.

To use a GridView control a DataSource control has to be attached to the GridView control. The property DataSourceID of the GridView control binds the GridView control to the DataSource control and allows paging, sorting and database operations with the DataSource.

There are four popular DataSource controls and SqlDataSource control is one of them. We use SqlDataSource control, to attach the GridView control to the Sql Server Data base. The SqlDataSource control contains ConnectionString, SelectCommand, UpdateCommand and DeleteCommand properties. The ConnectionString property connects the DataSource (SqlDataSource) control to the database and properties such as SelectCommand, InsertCommand, UpdateCommand, DeleteCommand contains SQL statements that are executed when we display, insert and delete records using the GridView control.

A GridView control and a SqlDataSource control is placed on the web form. The Select, Delete, Insert and Update commands are used to interact with the Sql Server database.

The Figure shown above gives a overview of how the gridView control works in tandem with SqlDataSource control and the Sql server database. The GridView control is placed on the web form and is visible to the user. The SqlDataSource control is placed on the web form and is not visible to the user at runtime.

After the DataSourceID of the GridView control is set, the SqlDataSource control is ready to establish a connection with the Sql Server database, retrieve data from the database table and pass the data to the GridView control.

For inserting records, either the DetailsView or the FormView control is commonly used with the GridView control. However, we can also use GridView controls FooterTemplate to insert records.

While developing commercial database software such as Accounting or ERP software, we can use GridView controls FooterTemplate to insert records. We need not use DetailsView control or the FormView control.

GridView control supports programmatic access to the GridView object model and sets properties and handles events dynamically.

The GridView control is the successor to the DataGrid control. The GridView control displays database records in a HTML table with each record in a table row and each field in a column.

To display data in the GridView control from the sql server database, we need to bind the GridView control to the SqlDataSource control either declaratively or programmatically.

By default, the AutoGenerateColumns property is set to 'true', which renders each field in the data source as a column in the GridView control.

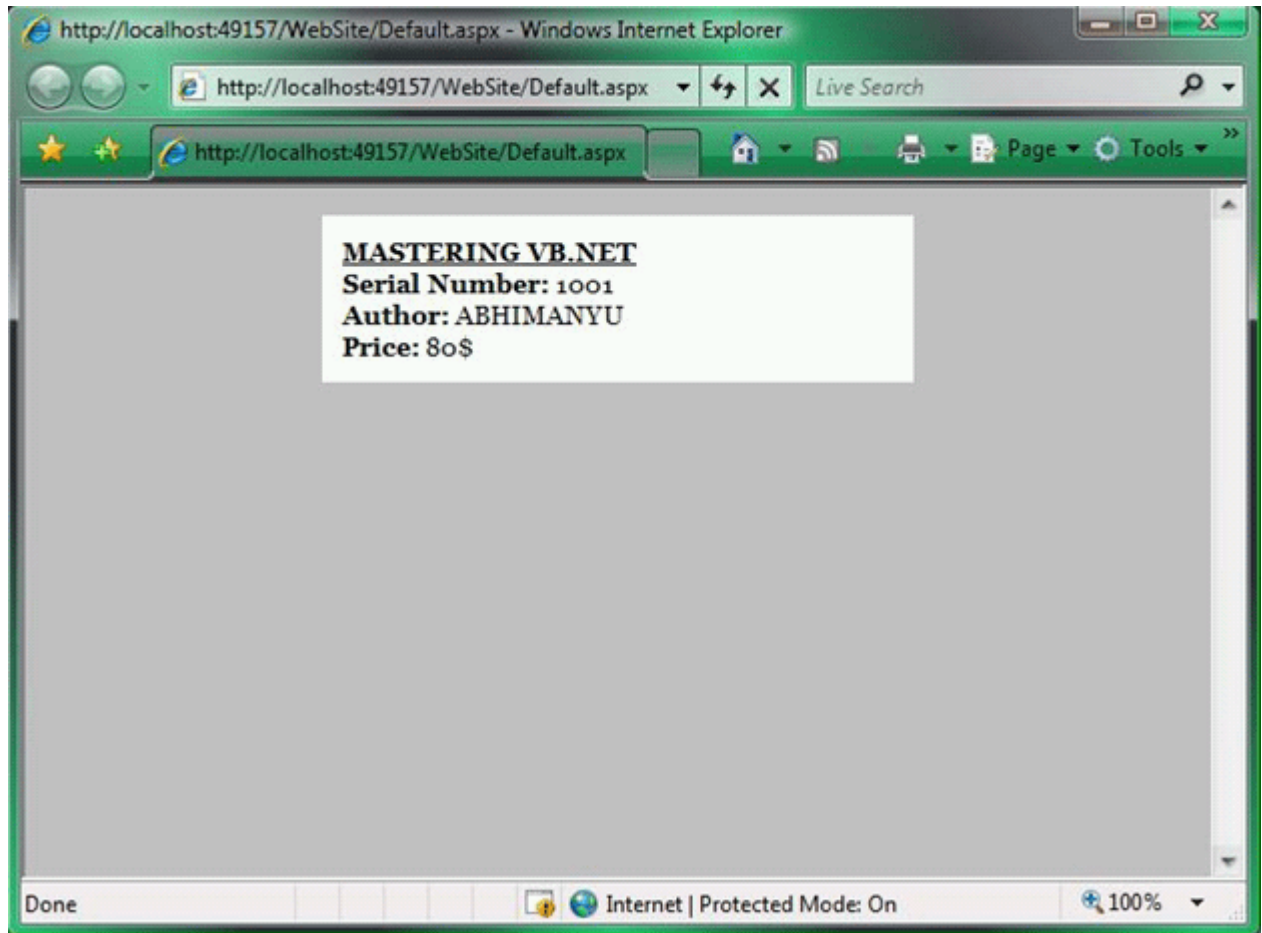
- **FormView:**

Introduction

We use the FormView control to do anything that we also can do with the DetailsView control. Just as we can with the DetailsView control, we can use the FormView control to display, page, edit, insert, and delete database records. However, unlike the DetailsView control, the FormView control is entirely template driven. I end up using the FormView control much more than the DetailsView control. The FormView control provides us with more control over the layout of a form. Furthermore, adding validation controls to a FormView is easier than adding validation controls to a DetailsView control.

Displaying Data with FormView Control

We can display a database record with the FormView control by using an ItemTemplate. For example, the page given below displays a record from the BOOK_LIST database table.



```
<%@ Page Language="VB" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <style type="text/css">
    html
    {
      background-color:silver;
    }
  #content
  {
    margin:auto;
    width:300px;
```

```

padding:10px;
background-color:white;
font:14px Georgia,Serif;
}
</style>
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div id="content">
<asp:FormView
id="frmBooks"
DataSourceID="SqlDataSource1"
Runat="server">
<ItemTemplate>
<b><u><%# Eval("TITLE")%></u></b>
<br />
<b>Serial Number:</b>
<%# Eval("ID")%>
<br />
<b>Author:</b>
<%# Eval("AUTHOR")%>
<br />
<b>Price:</b>
<%# Eval("PRICE")%>
</ItemTemplate>
</asp:FormView>
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
ConnectionString="<%%$ ConnectionStrings:DatabaseConnectionString1 %>"
ProviderName="<%%$ ConnectionStrings:DatabaseConnectionString1.ProviderName %
>"
SelectCommand="SELECT [ID], [TITLE], [AUTHOR], [PRICE] FROM [BOOK_LIST]">
</asp:SqlDataSource>
</div>
</form>
</body>
</html>

```

In above code, FormView control's DataSourceID property points to the SqlDataSource control. The SqlDataSource control retrieves the first record from the Movies database table.

Notice that the ItemTemplate contains databinding expressions that display the values of the TITLE, ID, AUTHOR and PRICE columns. The Eval() method retrieves the values of these columns. I have also used CSS for its better look.

- **ListView & DataPager Controls:**

Introduction

ASP.NET 3.5 introduces a new data binding control named the ListView. ASP.NET already has a lot of data bind controls; it should be more than 10. But the good news is, ListView can literally replace all other data binding controls in ASP.NET. ListView control makes data binding easier than previous controls. It has included styling with CSS, flexible pagination, and sorting, inserting, deleting, and updating features.

Complete ListView

<u>Id</u>	<u>Name</u>	<u>Type</u>	
1	Ashrafur	Rahaman Faisal	Family
2	Kader Razwan Mishu	Friends	Update Delete Cancel
3	Ajanta Das	Friends	Edit
4	Mahfuzur Rahaman	Family	Edit
5	Raisul Kabir	Business	Edit
	First Name	Last Name	Contact Type
			Insert

1 2 ...

In this article, I will describe features in ListView step by step with related code review.

1. Data binding
2. Data pager
3. Sorting
4. Insert, update and delete

DataBinding

The ListView is a template driven control which means that it will not render anything. By default, the developer must completely specify the HTML he/she wants to render in the form of templates. To show data in ListView control, you need to take a LayoutTemplate (to define top level of HTML for output rendering). To show data, you need to take ItemTemplate and AlternativItemTemplate to show alternative row as with different CSS. Here is the example of simple data binding with AlternativItemTemplate.

```
//very simple databinding in ListView
<LayoutTemplate>
<table border="0" cellpadding="1">
  <tr style="background-color:#E5E5FE">
    <th align="left"><asp:LinkButton ID="lnkId" runat="server">Id</asp:LinkButton></th>
    <th align="left"><asp:LinkButton ID="lnkName"
runat="server">Name</asp:LinkButton></th>
    <th align="left"><asp:LinkButton ID="lnkType"
runat="server">Type</asp:LinkButton></th>
    <th></th>
  </tr>
  <tr id="itemPlaceholder" runat="server"></tr>
</table>
</LayoutTemplate>
<ItemTemplate>
<tr>
```

```

<td><asp:Label runat="server" ID="lblId"><%=#Eval("ID") %></asp:Label></td>
<td><asp:Label runat="server" ID="lblName"><%=#Eval("FirstName")+
    "+Eval("LastName") %></asp:Label></td>
<td><asp:Label runat="server" ID="lblType"><%=#Eval("ContactType")
%></asp:Label></td>
<td></td>
</tr>
</ItemTemplate>
<AlternatingItemTemplate>
<tr style="background-color:#EFEFEF">
<td><asp:Label runat="server" ID="lblId"><%=#Eval("ID") %></asp:Label></td>
<td><asp:Label runat="server" ID="lblName"><%=#Eval("FirstName")+ " "+
    Eval("LastName") %></asp:Label></td>
<td><asp:Label runat="server" ID="lblType"><%=#Eval("ContactType")
%></asp:Label></td>
<td></td>
</tr>
</AlternatingItemTemplate>

```

Here Layout template is making header of the control, and ItemTemplate is showing data taken from table by Binding columns with Label controls, and AlternatingItemTemplate does the same as ItemTemplate just changing CSS for alternative columns.

DataPager

To add pagination in the listview, you need to add an asp:DataPager control, better to put this control inside LayoutTemplate at the end of the LayoutTemplate. DataPager control has many options to show pagination and these are useful too. Here I have just used a simple one.

```

<asp:DataPager ID="ItemDataPager" runat="server" PageSize="5">
<Fields>
    <asp:NumericPagerField ButtonCount="2" />
</Fields>
</asp:DataPager>

```

- **Architecture of our sample 3 tier application**

A three tiered application need not be necessarily distributed. That means various layers (Data Access Layer, Business Logic Layer and User Interface Layer) can be within a single web site. You can of course separate out individual layers and host them on a dedicated server if you so wish. In such cases you may need to use Remoting or Web Services.

The components in a three tiered system are generally implemented in a class library. However in ASP.NET 2.0 there is one more alternative. The special folder App_Code is intended to store the classes used by your web site. You can neatly organize the classes inside the App_Code folder by creating subdirectories.

In ASP.NET 1.x you would have used either DataSet or ArrayList kind of collection to pass data from one layer to the other. In ASP.NET 2.0 you can use DataSet in the same way as in 1.x. However, instead of using ArrayList you can go for generic List class.

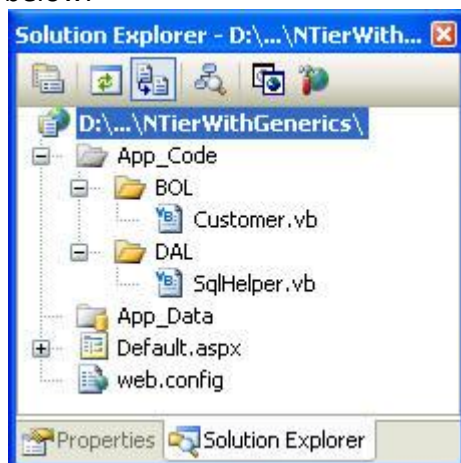
In ASP.NET 1.x the user interface layer consists of web forms. You need to assemble various server controls on the form and manually data bind them with the underlying data source. Tasks such as inserting, updating and deleting data were all manual i.e. you need to write code to accomplish them. In ASP.NET 2.0 new data bound controls such as GridView, DetailsView and FormView along with Data Source Controls come handy. They allow you to design the UI with minimal coding.

To summarize the architecture of our sample application will be like this:

- A class called SqlHelper will act as a Data Access Layer and will reside in a subfolder called DAL inside the App_Code folder. This class will take the data in and out of Customers table of Northwind database
- A class called Customer will act as a Business Logic Layer and will be placed in a subfolder called BOL inside the App_Code folder
- A UP will consist of a web form. We will use Object Data Source and DetailsView control for adding, updating and deleting data

Developing the sample application

Let's begin our development. First of all create a new web site using VS.NET 2005. Open the solution explorer, right click on the web site and choose Add ASP.NET Folder option. Select App_Code folder so as to create the App_Code folder. Add two subfolders called DAL and BOL to the App_Code folder. Add a class called SqlHelper in the DAL folder. Similarly add a class called Customer in the BOL folder. Your folder structure should resemble as shown below:



Open the web.config file and add a connection string as shown below:

```
<connectionStrings>
<add name="connectionstring" connectionString="data source
=.\\sqlexpress;initial catalog=northwind;integrated security
=true" providerName="System.Data.SqlClient"/>
</connectionStrings>
```

The <connectionStrings> section is used to store database connection strings. In our example the connection string points to Northwind database from a SQL Server Express database.

Creating Data Access Layer

The SqlHelper class represents our Data Access Layer. Open SqlHelper.vb and key in following code in it:

```
Public Class SqlHelper
Private Shared connectionstring As String
```



```

Shared Sub New()
connectionstring = ConfigurationManager.ConnectionStrings
("connectionstring").ConnectionString
End Sub

Public Shared Function ExecuteNonQuery(ByVal sql As String,
ByVal params() As SqlParameter) As Integer
Dim cnn As New SqlConnection(connectionstring)
Dim cmd As New SqlCommand(sql, cnn)
For i As Integer = 0 To params.Length - 1
cmd.Parameters.Add(params(i))
Next
cnn.Open()
Dim retval As Integer = cmd.ExecuteNonQuery()
cnn.Close()
Return retval
End Function

Public Shared Function ExecuteDataSet(ByVal sql As String)
As DataSet
Dim ds As New DataSet
Dim da As New SqlDataAdapter(sql, connectionstring)
da.Fill(ds)
Return ds
End Function

Public Shared Function ExecuteDataSet(ByVal sql As String,
ByVal params() As SqlParameter) As DataSet
Dim ds As New DataSet
Dim da As New SqlDataAdapter(sql, connectionstring)
For i As Integer = 0 To params.Length - 1
da.SelectCommand.Parameters.Add(params(i))
Next
da.Fill(ds)
Return ds
End Function
End Class

```

The SqlHelper class consists of several shared methods. In the shared constructor the class reads the connection string that we stored in the web.config file previously. In order to read the connection string the code uses ConfigurationManager class.

The ExecuteNonQuery() method is designed for action queries such as INSERT, UPDATE and DELETE. The overloads of ExecuteDataSet() method is intended to return a DataSet filled with the required records.

Creating Business Logic Layer

The Customer class represents our Business Logic Layer. Open Customer class and add the following code to it.

```
Imports System.Data
Imports System.Data.SqlClient
Imports System.Collections.Generic
```

```
Public Class Customer
Private strID As String
Private strCompany As String
Private strContact As String
Private strCountry As String
```

```
Public Property CustomerID() As String
Get
Return strID
End Get
Set(ByVal value As String)
strID = value
End Set
End Property
```

```
Public Property CompanyName() As String
Get
Return strCompany
End Get
Set(ByVal value As String)
strCompany = value
End Set
End Property
```

```
Public Property ContactName() As String
Get
Return strContact
End Get
Set(ByVal value As String)
strContact = value
End Set
End Property
```

```
Public Property Country() As String
Get
Return strCountry
End Get
Set(ByVal value As String)
strCountry = value
End Set
End Property
```

```
Public Shared Function Insert(ByVal c As Customer) As Integer
```

```

Dim sql As String
sql = "insert into customers(customerid,companyname,
contactname,country) values(@custid,@company,@contact,@country)"
Dim params(3) As SqlParameter
params(0) = New SqlParameter("@custid", c.CustomerID)
params(1) = New SqlParameter("@company", c.CompanyName)
params(2) = New SqlParameter("@contact", c.ContactName)
params(3) = New SqlParameter("@country", c.Country)
Return SqlHelper.ExecuteNonQuery(sql, params)
End Function

```

```

Public Shared Function Update(ByVal c As Customer)
As Integer
Dim sql As String
sql = "update customers set companyname=@company,contactname
=@contact,country=@country where customerid=@custid"
Dim params(3) As SqlParameter
params(0) = New SqlParameter("@company", c.CompanyName)
params(1) = New SqlParameter("@contact", c.ContactName)
params(2) = New SqlParameter("@country", c.Country)
params(3) = New SqlParameter("@custid", c.CustomerID)
Return SqlHelper.ExecuteNonQuery(sql, params)
End Function

```

```

Public Shared Function Delete(ByVal c As Customer)
As Integer
Dim sql As String
sql = "delete from customers where customerid=@custid"
Dim params(0) As SqlParameter
params(0) = New SqlParameter("@custid", c.CustomerID)
Return SqlHelper.ExecuteNonQuery(sql, params)
End Function

```

```

Public Shared Function SelectAll() As List(Of Customer)
Dim ds As DataSet = SqlHelper.ExecuteDataSet
("select customerid,companyname,contactname,
country from customers")
Dim arr As New List(Of Customer)
For Each row As DataRow In ds.Tables(0).Rows
Dim c As New Customer
c.CustomerID = row("customerid")
c.CompanyName = row("companyname")
c.ContactName = row("contactname")
c.Country = row("country")
arr.Add(c)
Next
Return arr

```

End Function

```
Public Shared Function SelectSingle(ByVal custid As String)
    As Customer
    Dim params(0) As SqlParameter
    params(0) = New SqlParameter("@custid", custid)
    Dim ds As DataSet = SqlHelper.ExecuteDataSet
    ("select customerid,companyname,contactname,country from
    customers where customerid=@custid", params)
    Dim row As DataRow = ds.Tables(0).Rows(0)
    Dim c As New Customer
    c.CustomerID = row("customerid")
    c.CompanyName = row("companyname")
    c.ContactName = row("contactname")
    c.Country = row("country")
    Return c
End Function
End Class
```

The Customer class consists of four public properties and five public shared methods. The method names are self-explanatory. The methods essentially manipulate data from Customers table of Northwind database.

Note the code marked in bold. We have imported System.Collections.Generic namespace because we want to return records as collection of Customer objects. Have a look carefully at SelectAll() method. The method returns a List of Customers. Inside we have declared a variable (arr) of type List. The SelectAll() method calls ExecuteDataSet() method of the SqlHelper class and retrieves a DataSet filled with required records. Then the method iterates through all the records from the DataSet and with each iterations add a Customer object to the List.

Creating the User Interface

The UI consists of a single web form. Open the default web form in the VS.NET IDE and design it as shown below:

CustomerID	Country	CompanyName	ContactName
abc	abc	abc	abc

[Edit](#) [Delete](#) [New](#)

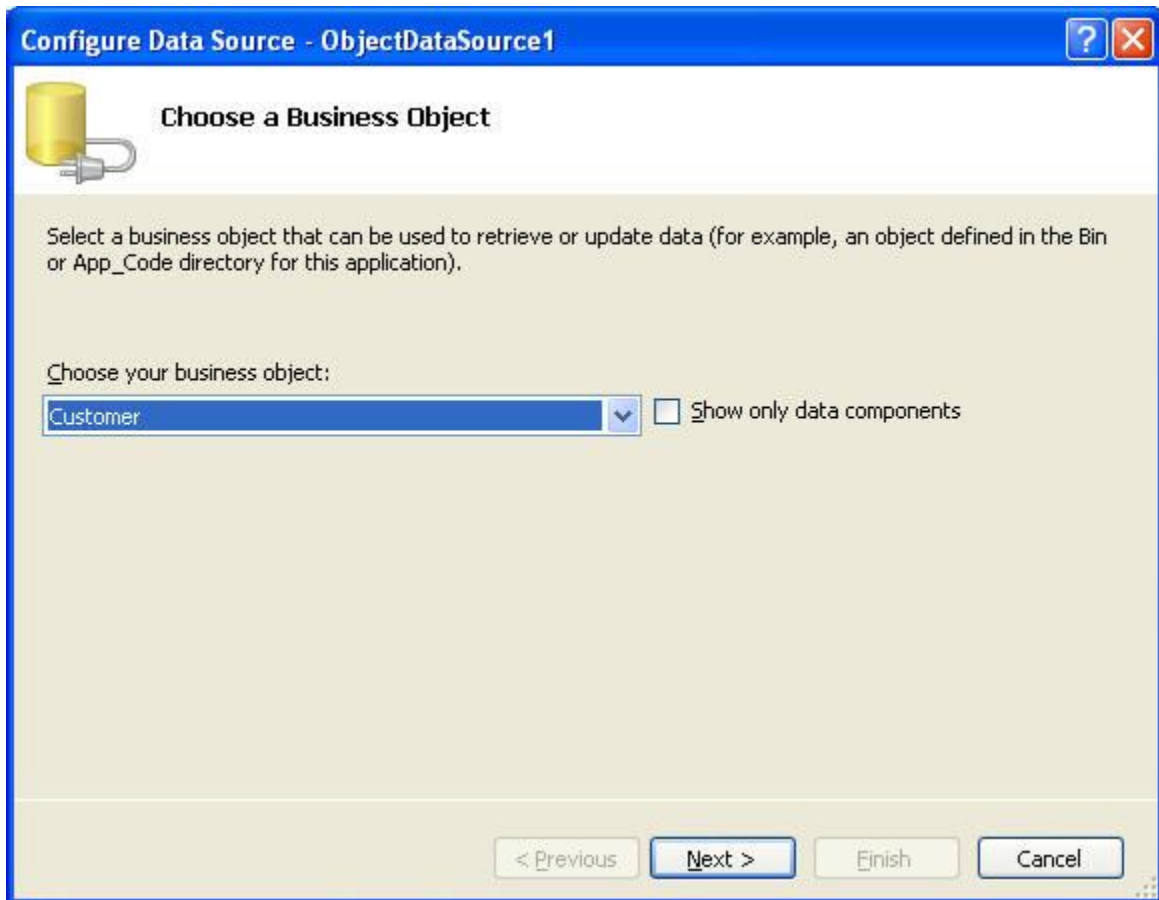
ObjectDataSource - ObjectDataSource1

ObjectDataSource - ObjectDataSource2

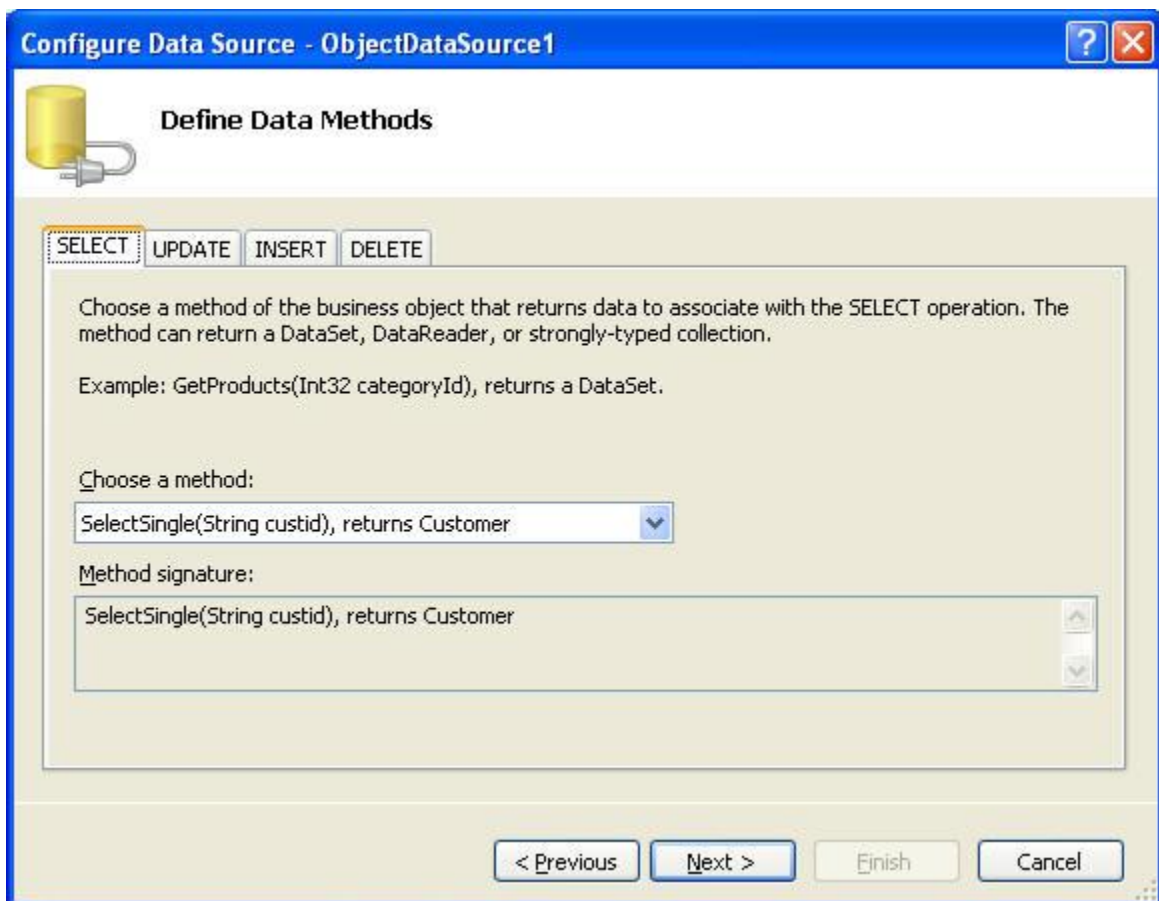
The DropDownList shows a list of all CustomerIDs from the database. Once you select a particular customer ID its details are displayed in the DetailsView below. The DetailsView is

bound with ObjectDataSource1 where as the DropDownList is bound with ObjectDataSource2.

Open the smart tags of ObjectDataSource1 and choose "Configure Data Source...". Doing so will start a wizard as shown below:



Select Customer class from the list and click Next. On the next Wizard step choose methods for selecting, updating, inserting and deleting data. In our case the methods are SelectSingle(), Update(), Insert() and Delete() respectively.



Once you specify the methods click on Next. The next step will ask you the source of the custid parameter of SelectSingle() method. In our case the source control is DropDownList1.

Configure Data Source - ObjectDataSource1

Define Parameters

The wizard has detected one or more parameters in your SELECT method. For each parameter in the SELECT method, choose a source for the parameter's value.

Parameters:

Name	Value
custid	DropDownList1.Select...

Parameter source: Control

ControlID: DropDownList1

DefaultValue:

[Show advanced properties](#)

Method signature:
SelectSingle(String custid), returns Customer

< Previous Next > Finish Cancel

Click on Finish to close the Wizard.

Similarly configure the ObjectDataSource2 to select all the Customers i.e. specify Select method as SelectAll().


We just finished configuring the object data source controls. Running the wizard sets the following important properties of the object data source controls.

Property	Description
TypeName	This property specifies the name of the class that is acting as a Business Object. In our case it is the Customer class.
DataObjectTypeName	This property indicates the object that is being passed as a parameter to Insert, Update and Delete methods. In our case it is the Customer class.
SelectMethod	The method from Business Object that is going to fetch data.
InsertMethod	The method from Business Object that is going to insert record in the underlying data store.
UpdateMethod	The method from Business Object that is going to update record in the underlying data store.
DeleteMethod	The method from Business Object that is going to delete record in the underlying data store.

Now open the smart tag of DetailsView and set its data source to ObjectDataSource1. Also, enable inserting, editing and deleting.

DetailsView Tasks	
Auto Format...	
Choose Data Source:	ObjectDataSource1
Configure Data Source...	
Refresh Schema	
Edit Fields...	
Add New Field...	
<input type="checkbox"/>	Enable Paging
<input checked="" type="checkbox"/>	Enable Inserting
<input checked="" type="checkbox"/>	Enable Editing
<input checked="" type="checkbox"/>	Enable Deleting
Edit Templates	

Similarly open smart tag of DropDownList1 and set its data source to ObjectDtaSource2. Specify DataTextField and DataValueField properties as CustomerID.

Data Source Configuration Wizard	
 Choose a Data Source	
Select a data source:	
ObjectDataSource2	
Select a data field to display in the DropDownList:	
CustomerID	
Select a data field for the value of the DropDownList:	
CustomerID	
Refresh Schema	
<div>OK</div> <div>Cancel</div>	

Finally enable AutoPostBack for the DropDownList.

DropDownList Tasks	
Choose Data Source...	
Configure Data Source...	
Refresh Schema	
Edit Items...	
<input checked="" type="checkbox"/>	Enable AutoPostBack

We want that whenever an item is added or deleted the DropDownList should immediately reflect the changes. In order to achieve this you need to handle ItemInserted and ItemDeleted events of DetailsView. These event handlers are shown below

```
Protected Sub DetailsView1_ItemInserted  
(ByVal sender As Object, ByVal e As DetailsViewInsertedEventArgs)  
Handles DetailsView1.ItemInserted  
DropDownList1.DataBind()  
End Sub
```

```
Protected Sub DetailsView1_ItemDeleted  
(ByVal sender As Object, ByVal e As DetailsViewDeletedEventArgs)  
Handles DetailsView1.ItemDeleted  
DropDownList1.DataBind()  
End Sub
```

The event handlers simply call the DataBind() method of the DropDownList which causes it to repopulate itself.

That's it. Run the web form and test editing and deleting functionality for different customers. The following figure shows the web form in action.



- LINQ to SQL

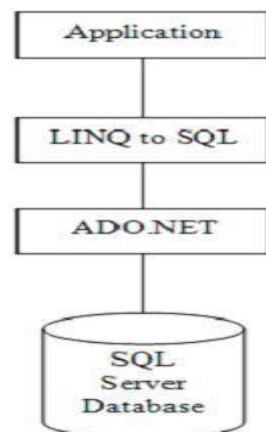
LINQ to SQL is a component of the .NET Framework version 3.5 that provides a run-time infrastructure for managing relational data as objects.

In LINQ to SQL, the data model of a relational database is mapped to an object model expressed in the programming language of the developer. When the application runs, LINQ to SQL translates the language-integrated queries in the object model into SQL and sends them to the database for execution. When the database returns the results, LINQ to SQL translates them back to objects that you can work with in your own programming language.

LINQ to SQL fully supports transactions, views, Stored Procedures, and user-defined functions. It also provides an easy way to integrate data validation and business logic rules into your data model, and supports single table inheritance in the object model.

LINQ to SQL is one of Microsoft's new ORM products to compete with many existing ORM products for the .NET platform on the market, like the Open Source products NHibernate, NPersist, and commercial products LLBLGen and WilsonORMapper. LINQ to SQL has many overlaps with other ORM products, but because it is designed and built specifically for .NET and SQL Server, it has many advantages over other ORM products. For example, it takes the advantages of all the LINQ features and it fully supports SQL Server Stored Procedures. You get all the relationships (foreign keys) for all tables, and the fields of each table just become properties of its corresponding object. You have even the intellisense popup when you type in an entity (table) name, which will list all of its fields in the database. Also, all of the fields and the query results are strongly typed, which means you will get a compiling error instead of a runtime error if you miss spell the query statement or cast the query result to a wrong type. In addition, because it is part of the .NET Framework, you don't need to install and maintain any third party ORM product in your production and development environments. Under the hood of LINQ to SQL, ADO.NET SqlClient adapters are used to communicate with real SQL Server databases. We will see how to capture the generated SQL statements at runtime later in this article.

Below is a diagram showing the usage of LINQ to SQL in a .NET application:



We will explore LINQ to SQL features in detail in this article and the following article.

Comparing LINQ to SQL with LINQ to Objects

In the previous article, we used LINQ to query in-memory objects. Before we dive further to the world of LINQ to SQL, we will first look at the relationships between LINQ to SQL and LINQ to Objects.

Followings are some key differences between LINQ to SQL and LINQ to Objects:

- LINQ to SQL needs a Data Context object. The Data Context object is the bridge between LINQ and the database. LINQ to Objects doesn't need any intermediate LINQ provider or API.
- LINQ to SQL returns data of type `IQueryable<T>` while LINQ to Objects returns data of type `IEnumerable<T>`.
- LINQ to SQL is translated to SQL by way of Expression Trees, which allow them to be evaluated as a single unit and translated to the appropriate and optimal SQL statements. LINQ to Objects does not need to be translated.

- LINQ to SQL is translated to SQL calls and executed on the specified database while LINQ to Objects is executed in the local machine memory.
The similarities shared between all aspects of LINQ are the syntax. They all use the same SQL like syntax and share the same groups of standard query operators. From a language syntax point, working with a database is the same as working with in-memory objects.

LINQ to SQL.

The table below lists some supported features by these two data access methodologies:

Features	LINQ to SQL	LINQ to Entities
Conceptual Data Model	No	Yes
Storage Schema	No	Yes
Mapping Schema	No	Yes
New Data Access Provider	No	Yes
Non-SQL Server Database Support	No	Yes
Direct Database Connection	Yes	No
Language Extensions Support	Yes	Yes
Stored Procedures	Yes	Yes
Single-table Inheritance	Yes	Yes
Multiple-table Inheritance	No	Yes
Single Entity from Multiple Tables	No	Yes
Lazy Loading Support	Yes	Yes

We will use LINQ to SQL in this article, because we will use it in the data access layer, and the data access layer is only one of the three layers for a WCF service. LINQ to SQL is much less complex than LINQ to Entities, so we can still cover it in the same article with WCF. However, once you have learned how to develop WCF services with LINQ to SQL through this article, and you have learned how to use LINQ to Entities through some other means, you can easily migrate your data access layer to using LINQ to Entities.

- **Web Site Navigation:**

Maintaining the menu of a large web site is difficult and time consuming.

In ASP.NET 2.0 the menu can be stored in a file to make it easier to maintain. This file is normally called **web.sitemap**, and is stored in the root directory of the web.

In addition, ASP.NET 2.0 has three new navigation controls:

- Dynamic menus
- TreeViews
- Site Map Path

The Sitemap File

The following sitemap file is used in this tutorial:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<siteMap>
```

```
<siteMapNode title="Home" url="/aspnet/w3home.aspx">
<siteMapNode title="Services" url="/aspnet/w3services.aspx">
  <siteMapNode title="Training" url="/aspnet/w3training.aspx"/>
  <siteMapNode title="Support" url="/aspnet/w3support.aspx"/>
</siteMapNode>
</siteMapNode>
</siteMap>
```

Rules for creating a sitemap file:

- The XML file must contain a <siteMap> tag surrounding the content
- The <siteMap> tag can only have one <siteMapNode> child node (the "home" page)
- Each <siteMapNode> can have several child nodes (web pages)
- Each <siteMapNode> has attributes defining page title and URL



Note: The sitemap file must be placed in the root directory of the web and the URL attributes must be relative to the root directory.

Dynamic Menu

The <asp:Menu> control displays a standard site navigation menu.

Code Example:

```
<asp:SiteMapDataSource id="nav1" runat="server" />
<form runat="server">
<asp:Menu runat="server" DataSourceId="nav1" />
</form>
```

The **<asp:Menu>** control in the example above is a placeholder for a server created navigation menu.

The data source of the control is defined by the **DataSourceId** attribute.

The **id="nav1"** connects it to the **<asp:SiteMapDataSource>** control.

The **<asp:SiteMapDataSource>** control automatically connects to the default sitemap file (**web.sitemap**).

[Click here to see a demo of Menu, TreeView, and SiteMapPath](#)

TreeView

The <asp:TreeView> control displays a multi level navigation menu.

The menu looks like a tree with branches that can be opened or closed with + or - symbol.

Code Example:

```
<asp:SiteMapDataSource id="nav1" runat="server" />
<form runat="server">
<asp:TreeView runat="server" DataSourceId="nav1" />
</form>
```

The **<asp:TreeView>** control in the example above is a placeholder for a server created navigation menu.

The data source of the control is defined by the **DataSourceId** attribute.

The **id="nav1"** connects it to the **<asp:SiteMapDataSource>** control.

The **<asp:SiteMapDataSource>** control automatically connects to the default sitemap file (**web.sitemap**).

[Click here to see a demo of Menu, TreeView, and SiteMapPath](#)

- **SiteMapPath**

The SiteMapPath control displays the trail (navigation path) to the current page. The path acts as clickable links to previous pages.

Unlike the TreeView and Menu control the SiteMapPath control does **NOT** use a SiteMapDataSource. The SiteMapPath control uses the web.sitemap file by default.

💡 Tips: If the SiteMapPath displays incorrectly, most likely there is an URL error (typo) in the web.sitemap file.

Code Example:

```
<form runat="server">
<asp:SiteMapPath runat="server" />
</form>
```

The **<asp:SiteMapPath>** control in the example above is a placeholder for a server created site path display.

- **Levels of state management**

1. Control level: In ASP.NET, by default controls provide state management automatically.
2. Variable or object level: In ASP.NET, member variables at page level are stateless and thus we need to maintain state explicitly.
3. Single or multiple page level: State management at single as well as multiple page level i.e., managing state between page requests.
4. User level: State should be preserved as long as a user is running the application.
5. Application level: State available for complete application irrespective of the user, i.e., should be available to all users.
6. Application to application level: State management between or among two or more applications.

Client side methods:

1. Hidden field

Hidden field is a control provided by ASP.NET which is used to store small amounts of data on the client. It store one value for the variable and it is a preferable way when a variable's value is changed frequently. Hidden field control is not rendered to the client (browser) and it is invisible on the browser. A hidden field travels with every request like a standard control's value.

Let us see with a simple example how to use a hidden field. These examples increase a value by 1 on every "No Action Button" click. The source of the hidden field control is.

[Hide](#) [Copy Code](#)

```
<asp:HiddenField ID="HiddenField1" runat="server" />
```

In the code-behind page:

[Hide](#) [Copy Code](#)

```
protected void Page_Load(object sender, EventArgs e)
{
    if (HiddenField1.Value != null)
    {
        int val= Convert.ToInt32(HiddenField1.Value) + 1;
        HiddenField1.Value = val.ToString();
        Label1.Text = val.ToString();
    }
}
```

```

    }
}
protected void Button1_Click(object sender, EventArgs e)
{
    //this is No Action Button Click
}

```

2. 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 deserilization 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.

[Hide](#) [Copy Code](#)

```

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();
}

```

3. 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:

Hide Copy Code

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

And the second one is:

Hide Copy Code

```
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:

Hide Copy Code

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

And the second way is:

Hide Copy Code

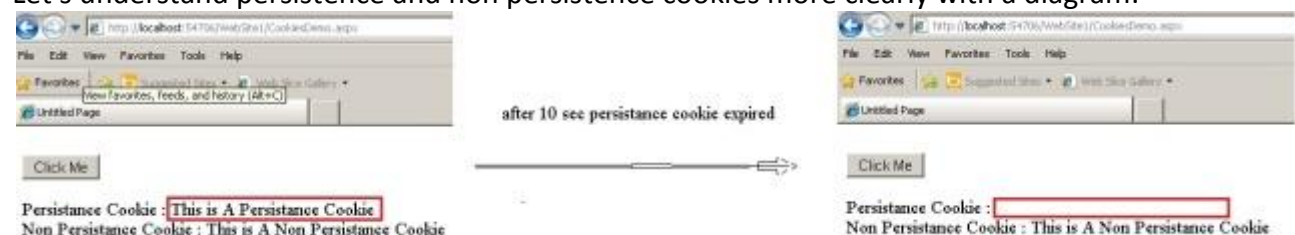
```
HttpCookie aCookieValNonPer = new HttpCookie("NonPersistence");
aCookieValNonPer.Value = "This is A Non Persistence Cookie";
Response.Cookies.Add(aCookieValNonPer);
```

How to read a cookie:

Hide Copy Code

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

Let's understand persistence and non persistence cookies more clearly with a diagram:



Limitation of cookies: The number of cookies allowed is limited and varies according to the browser. Most browsers allow 20 cookies per server in a client's hard disk folder and the size of a cookie is not more than 4096 bytes or 4 KB of data that also includes name and value data.

4. Control State

Control State is another client side state management technique. Whenever we develop a custom control and want to preserve some information, we can use view state but suppose view state is disabled explicitly by the user, the control will not work as expected. For expected results for the control we have to use Control State property. Control state is separate from view state.

How to use control state property: Control state implementation is simple. First override the OnInit() method of the control and add a call for the Page.RegisterRequiresControlState() method with the instance of the control to register. Then override LoadControlState and SaveControlState in order to save the required state information.

Server side

1. Session

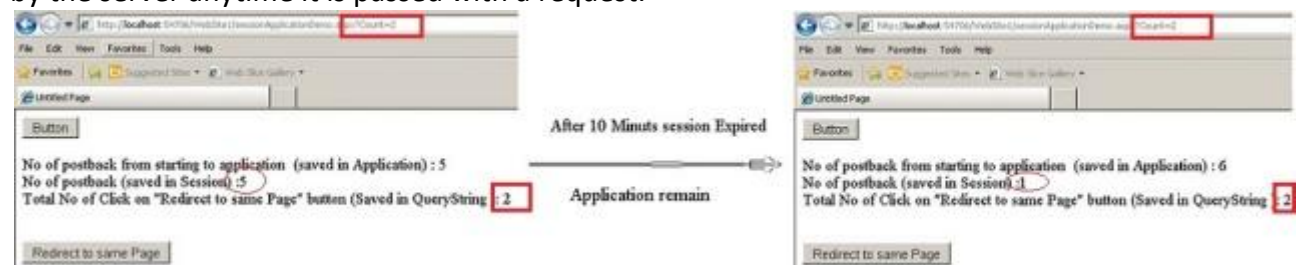
Session management is a very strong technique to maintain state. Generally session is used to store user's information and/or uniquely identify a user (or say browser). The server maintains the state of user information by using a session ID. When users makes a request without a session ID, ASP.NET creates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

How to get and set value in Session:

[Hide](#) [Copy Code](#)

```
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.

[Hide](#) [Copy Code](#)

```
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.

[Hide](#) [Copy Code](#)

```
<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.

[Hide](#) [Copy Code](#)

```
<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.

[Hide](#) [Copy Code](#)

```
<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.

Attributes	Description
Cookieless true/false	Indicates that the session is used with or without cookie. cookieless set to true indicates sessions without cookies is used and cookieless set to false indicates sessions with cookies is used. cookieless set to false is the default set.
timeout	Indicates the session will abound if it is idle before session is abounded explicitly (the default time is 20 min).
StateConnectionString	Indicates the session state is stored on the remote computer (server). This attribute is required when session mode is StateServer
SqlConnectionString	Indicates the session state is stored in the database. This attribute is required when session mode is SqlServer.

2. Application

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**:

Hide Copy Code

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

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

Hide Copy Code

```
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.

[Hide](#) [Copy Code](#)

```
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.

Navigation controls are very important for websites. Navigation controls are basically used to navigate the user through webpage. It is more helpful for making the navigation of pages easier. There are three controls in ASP.NET, which are used for Navigation on the webpage.

1. TreeView control
2. Menu Control
3. SiteMapPath control

There are some **Namespaces**, which are used for above Navigation controls which are given below:

`Using System.Web.UI.WebControls.TreeView ;`

`Using System.Web.UI.WebControls.Menu ;`

`Using System.Web.UI.WebControls.SiteMapPath ;`

In this tutorial, I will show you how to add **navigation control** on the web page. I will also give you real example of each control. Please read each control very carefully and use it on ASP.NET website. You can download each **control** application from bottom and implement on your system.

1.) The TreeView Control:-

The TreeView control is used for logically displaying the data in a hierarchical structure. We can use this navigation control for displaying the files and folders on the webpage. We can easily display the XML document, Web.SiteMap files and Database records in a tree structure.

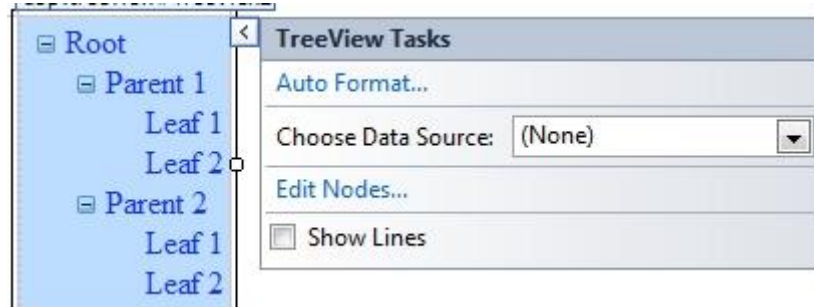
There are some types to generate navigation on webpage through **TreeView** control.

1. TreeView Node Editor dialog box
2. Generate TreeView based on XML Data
3. Generate TreeView based on Web.SiteMap data
4. Generate TreeView from Database.

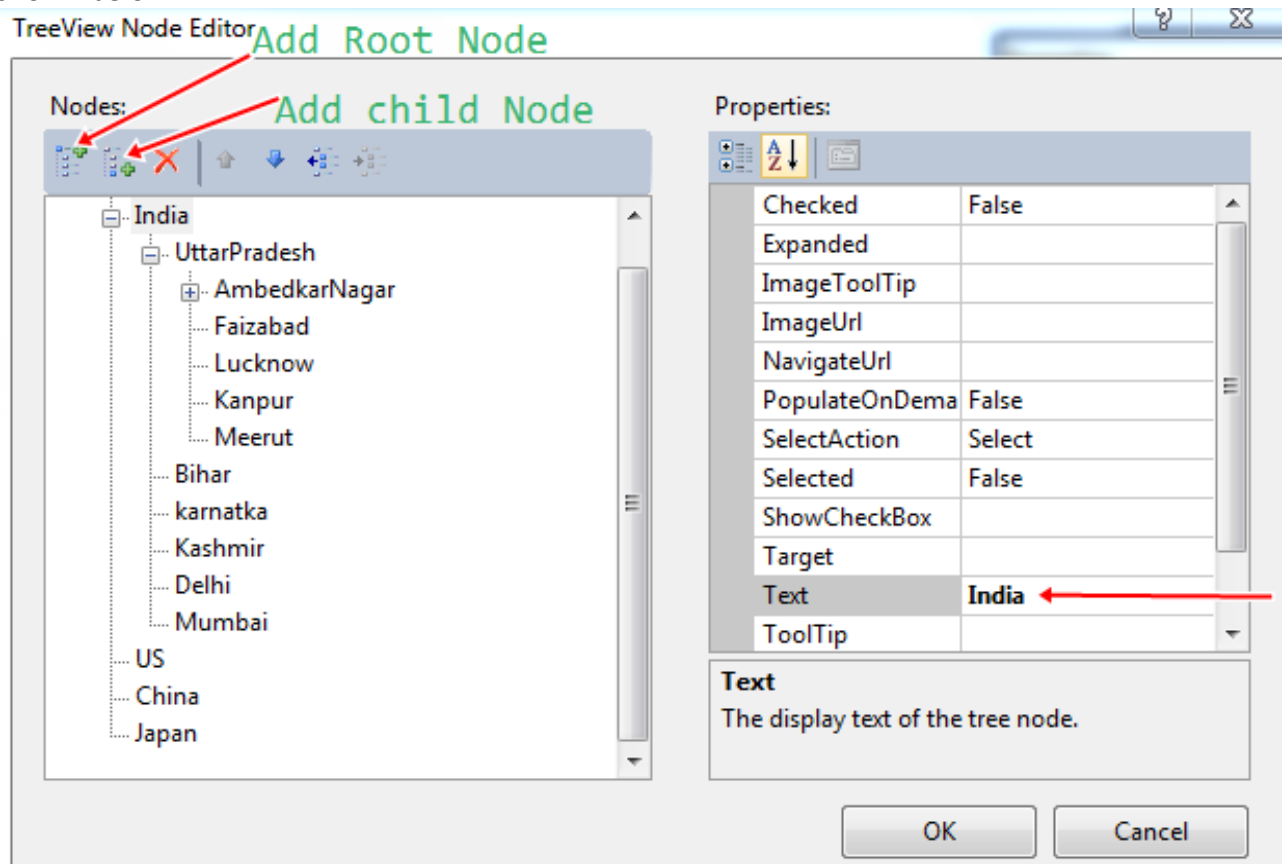
1.1) TreeView Node Editor Dialog Box:-

There are some steps to generate the Tree structure on the page, which are given below:

Step 1: First open your visual studio-->File-->New-->Website-->Select ASP.NET Empty Website -->OK-->open solution explorer-->Add New Web Form-->Drag and Drop TreeView control from Toolbox as shown below:



Step 2: Now go **properties** of TreeView control-->Click **Nodes**-->Add Root and child Node as shown below:



Step 3: Now Run the Program(press F5).

output:



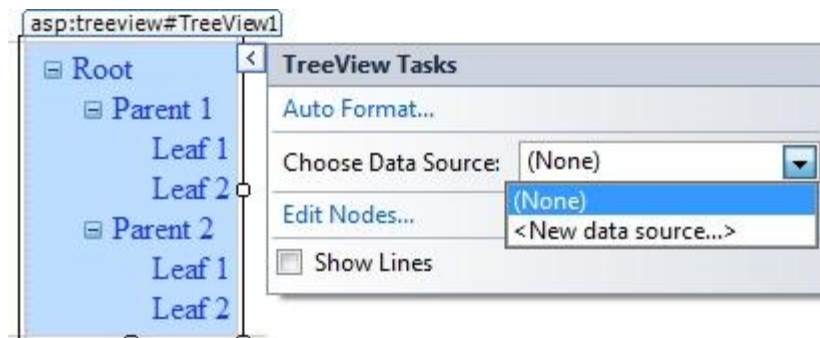
1.2) Generate TreeView Based On XML Data:-

There Are Some Steps To Implement This Concepts On The Webpage,Which Are Given Below:

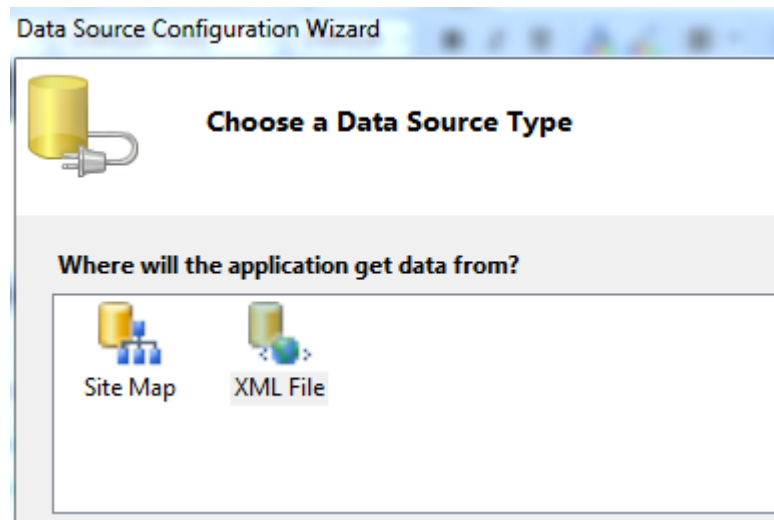
Step 1: Now First Add A Web Form And A XML File In Your Solution Explorer-->Now Open The XML File And Write The Following Codes As Shown Below-->Now Click Save.

```
<?xml version="1.0" encoding="utf-8" ?>
<application>
  <homepage title="Country" value="default.aspx">
    <page title ="INDIA" value="default.aspx">
      <subpage title ="up" value="default.aspx"/>
      <subpage title ="delhi" value="default.aspx"/>
      <subpage title ="mumbai" value="default.aspx"/>
      <subpage title ="kolkata" value="default.aspx"/>
    </page>
    <page title ="US" value="default.aspx"/>
    <page title ="CHNIA" value="default.aspx"/>
    <page title ="JAPAN" value="default.aspx"/>
  </homepage>
</application>
```

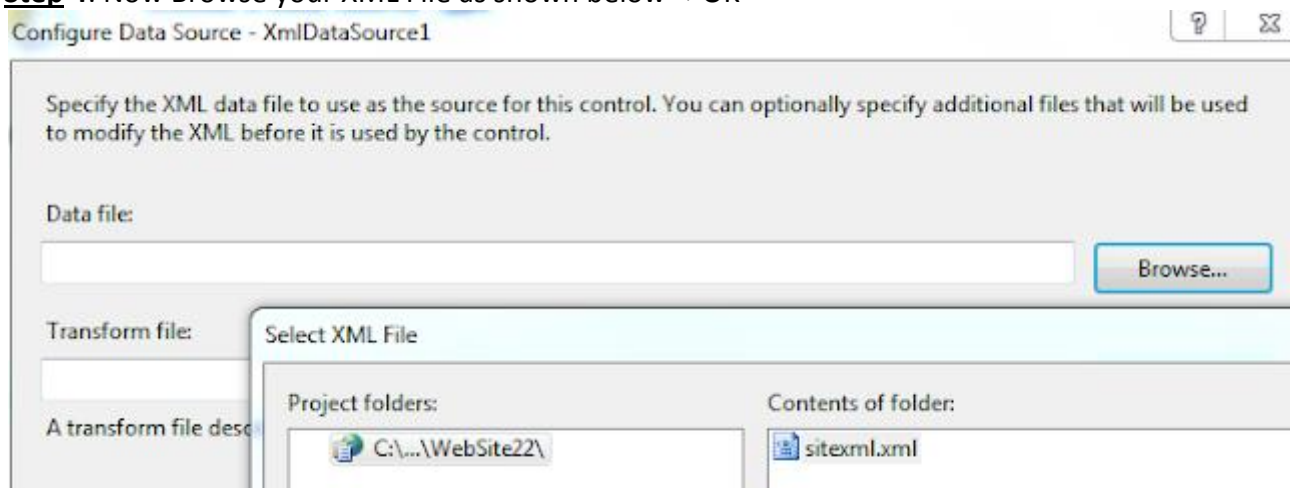
Step 2:Now Drag and drop TreeView control on the Web Form --> Now Choose Data Source from TreeView control-->Select **New data source** as shown below:



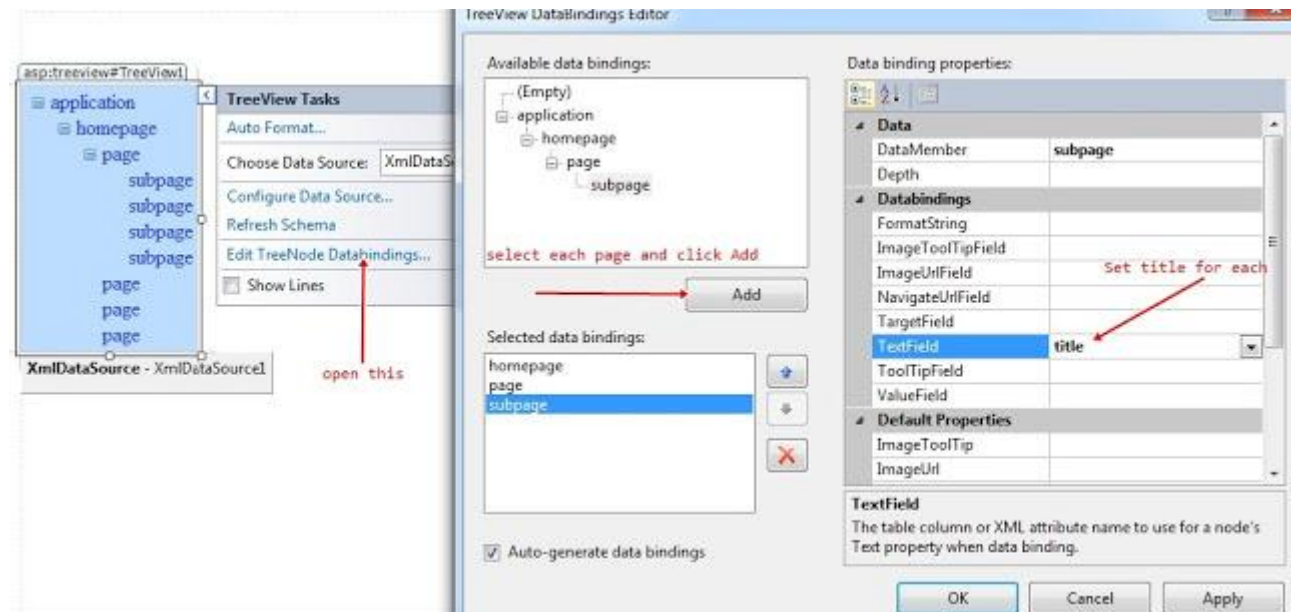
Step 3: Now select XML File as shown below:-->OK.



Step 4: Now Browse your XML File as shown below-->OK

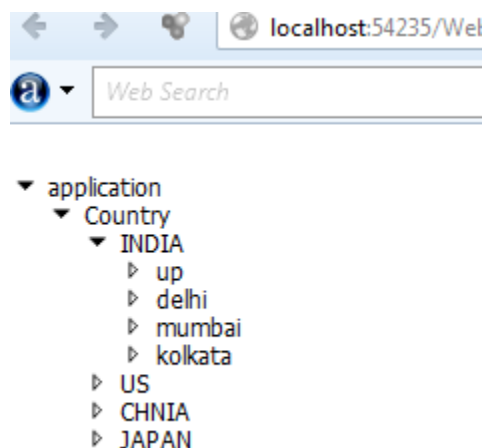


Step 5: Now click **Edit TreeNode DataBindings...**-->Select each page one by one -->and click **Add** button -->set **TextField =title** from right side for each page-->click **Apply** as shown below:



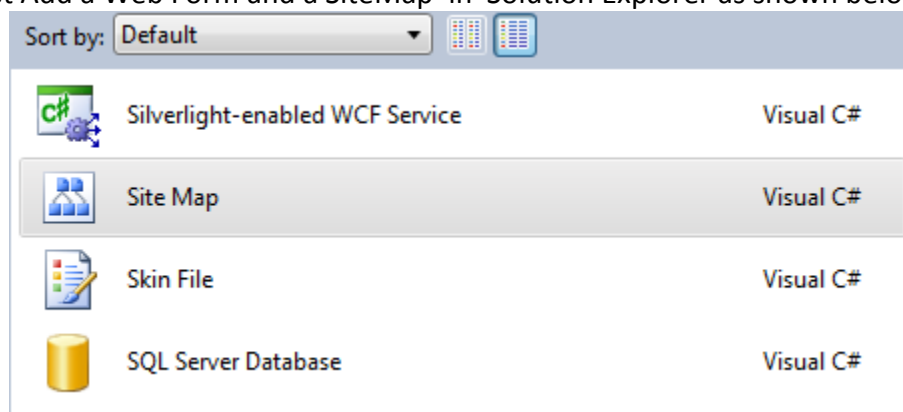
Step 6: Now Run the program(press F5).

Output:



1.3) Generate TreeView Based On Sitemap Data:-

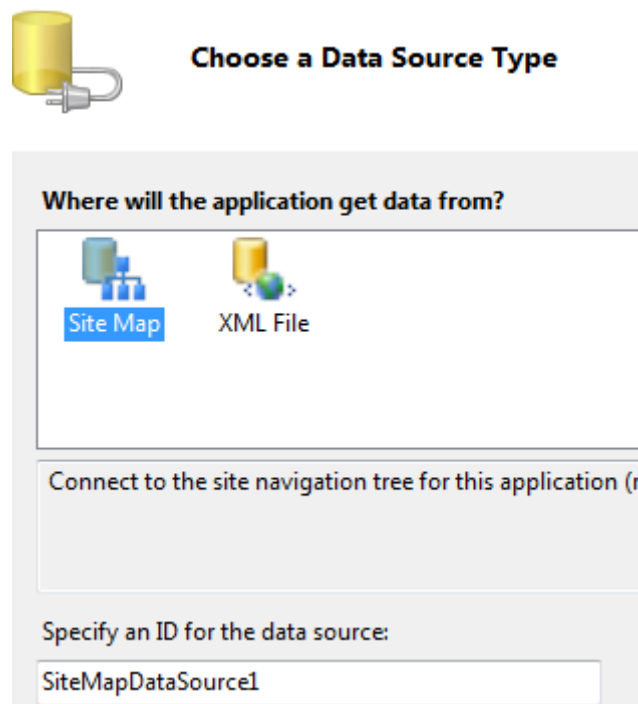
Step 1: First Add a Web Form and a SiteMap in Solution Explorer as shown below-->Add



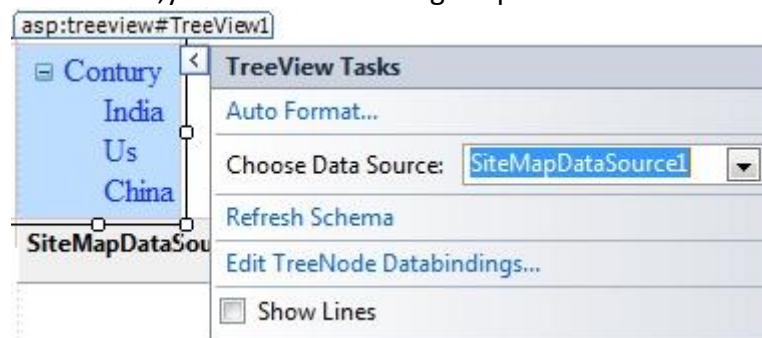
Step 2: Open **web.sitemap** file and write the following codes.which are given below-->**Save**

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="default.aspx" title="Contury" description="">
    <siteMapNode url="treeview1.aspx" title="India" description="" />
    <siteMapNode url="menu.aspx" title="Us" description="" />
    <siteMapNode url="menu1.aspx" title="China" description="" />
  </siteMapNode>
</siteMap>
```

Step 3: Now drag and drop TreeView control on the Form-->Now **choose Data Source**-->select **New data source**-->Select **SiteMap** as sown below:

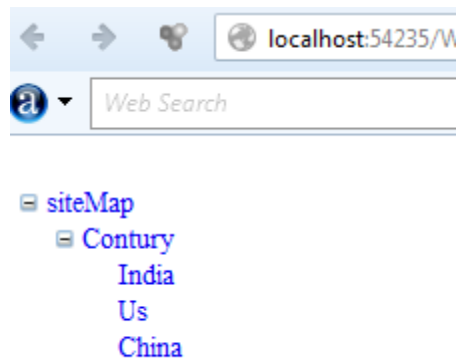


Step 4:Now click OK Button ,you will see following output.



Step 5: Now Run the program(press F5).

OUTPUT:



2.) The Menu Control:-

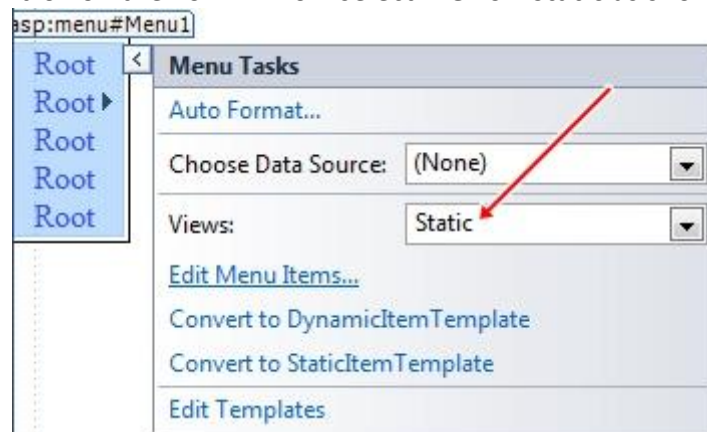
The menu control is a Navigation control, which is used to display the site navigation information. This can be used to display the site data structure vertically and horizontally. It can be used as a binding control as TreeView control. Means we can easily bind the XML and SiteMap data in menu control.

The menu control can be used as two types.

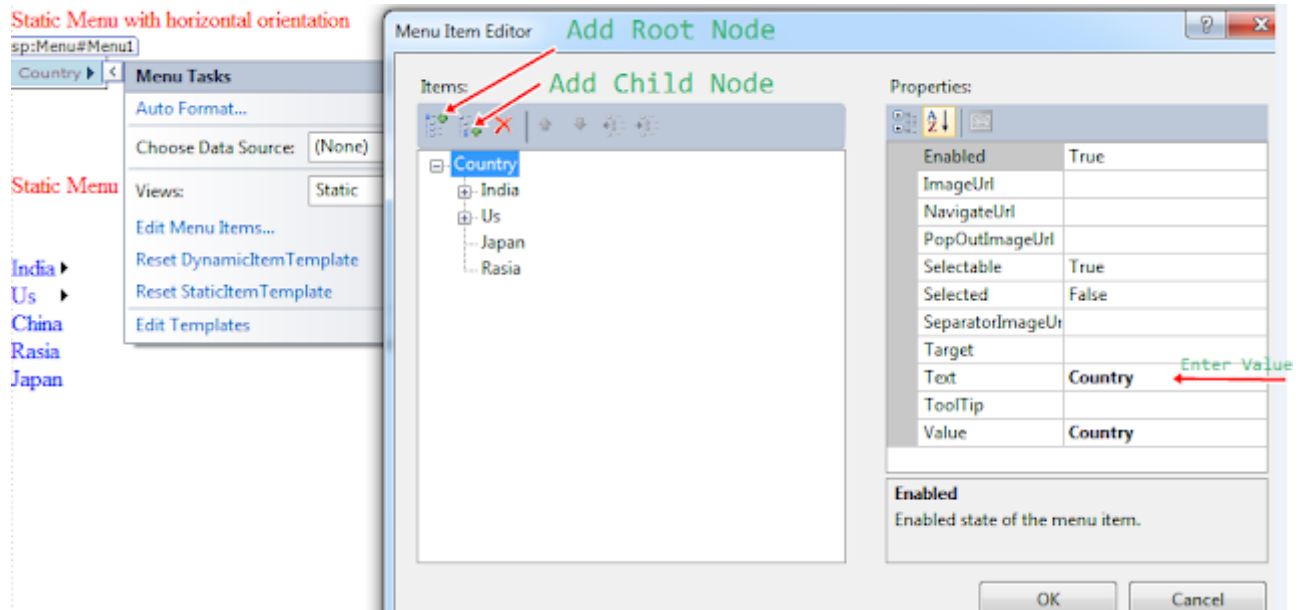
- **Static menu:-** It is used to display the parent menu items and their sub menu items on the page. Means it is used to display the entire structure of the static menu.
- **Dynamic menu:-** It is used to display the static menu as well as dynamic menu on the site. It means when user passes the mouse over the menu then it will appear on the site.

2.1) Static Menu:-

We can display site structure vertically as well as horizontally through static menu control on the site. There are some steps to implement the static menu control on the Web Form. Which are given below: **Step 1:** First Add a New Web Form in solution Explorer --> drag and drop **menu** control on the Form --> now select **Views = static** as shown below:



Step 2: Now click **Edit Menu Items...** --> Add **parent** and **child** nodes as shown below:



Step 3: Now Run the program(press F5).

Output:-



Note:- You can implement the vertical orientation as horizontal orientation.

2.2) Dynamic Menu:-

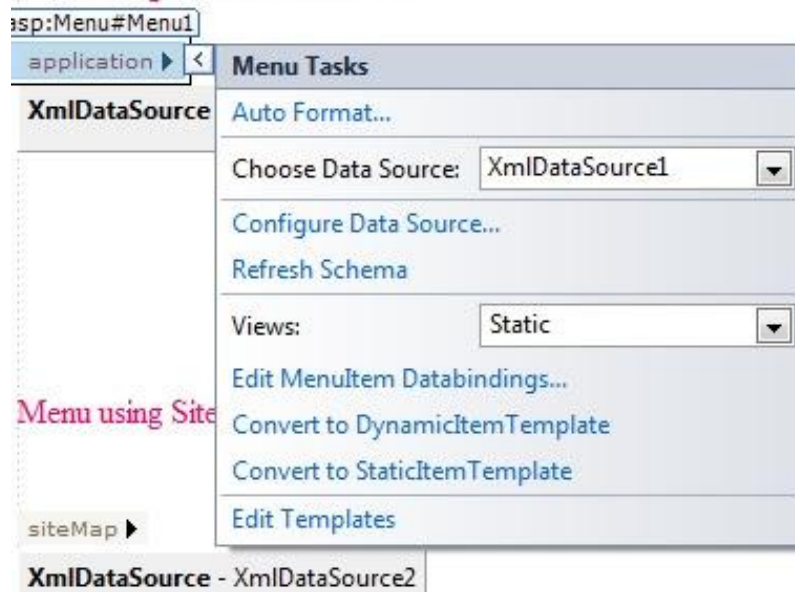
When user passes the mouse over the control ,the data will automatically appear on the site.we can generate dynamic menu control in two ways:

- Generate menu control using Xml Data source
- Generate menu control using SiteMap Data source

There are some steps to implement this concepts on the site.which are given below:-

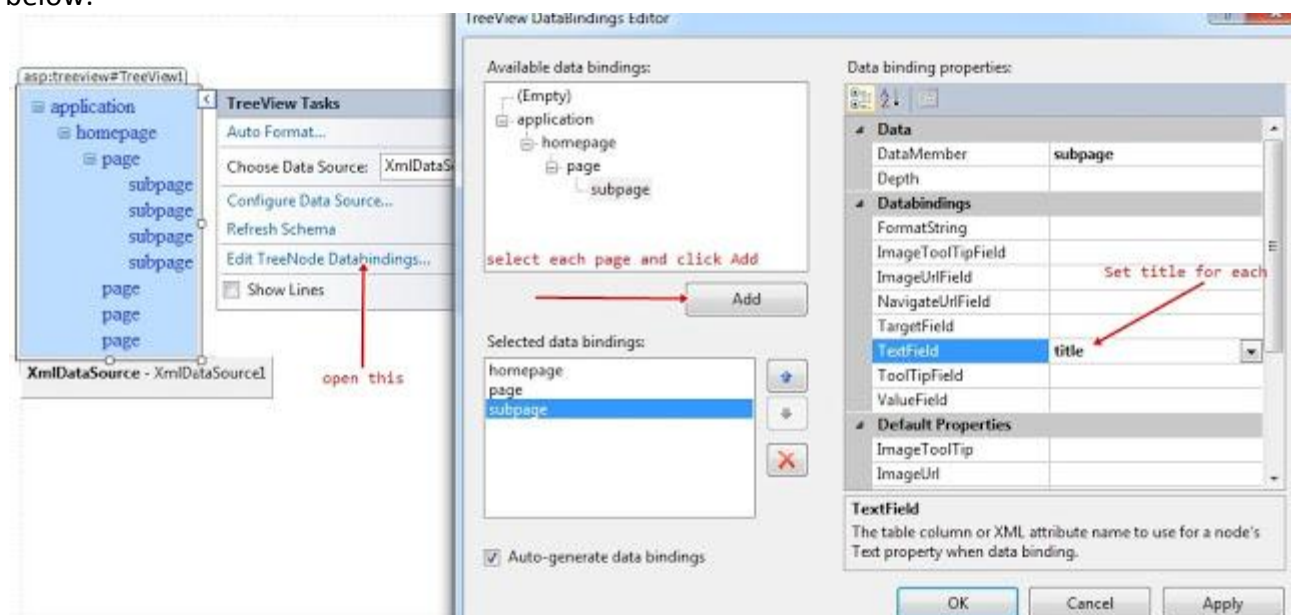
Step 1: First Add a **Web Form** in the solution Explorer-->Drag and drop menu control on the Form -->**choose Data Source** -->Select **XML File**-->OK-->**Browse XML File**-->OK.

Menu using XMLDataSource



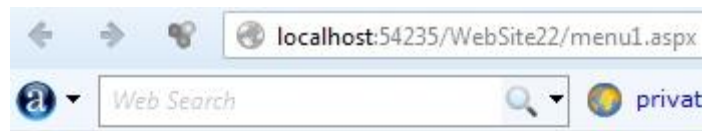
Menu using Site

Step 2: Now click **Edit TreeNode Databindings...**-->Select each page one by one -->and click **Add** button -->set **TextField =title** from right side for each page-->click **Apply** as shown below:



Step 3: Now Run the program(press F5).

Output:-



Menu using XMLDataSource

application	Country	INDIA	up
		US	delhi
		CHNIA	mumbai
		JAPAN	kolkata

Menu using SiteMapDataSource

siteMap ▶

Note:- We can use same process for SiteMap File also.

3.) The SiteMapPath Control:

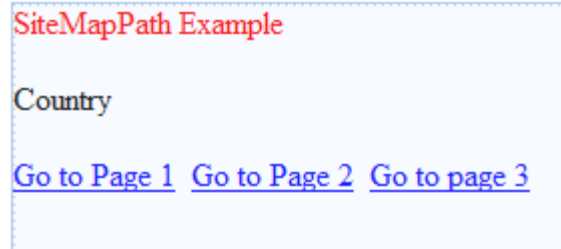
The SiteMapPath control is also used to display the Navigation information on the site. It display the current page's context within the entire structure of a website.

There are some steps to implement the SiteMapPath control on the web page. Which are given below:

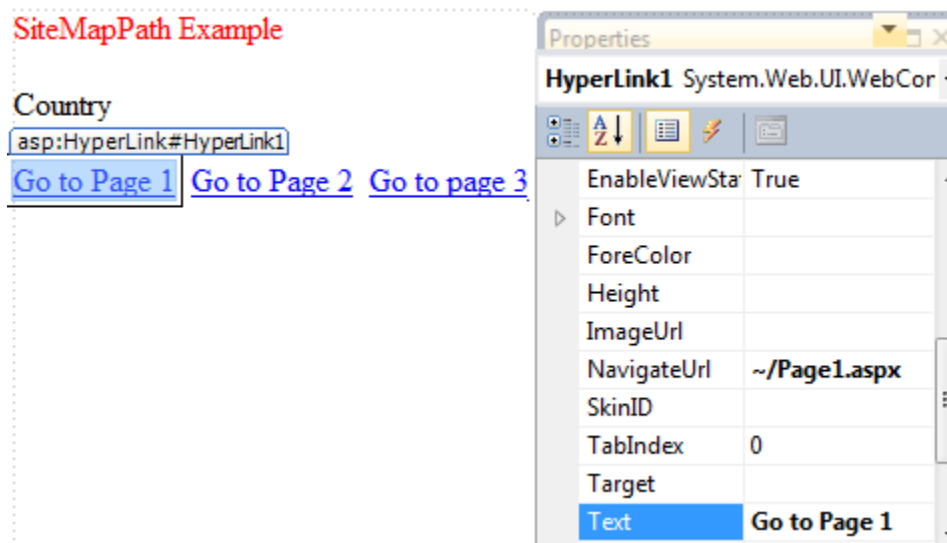
Step 1: First open your visual studio -->File -->New-->Website-->Select ASP.NET Empty Website-->Open **solution Explorer**-->Add a **Web Form** (SiteMap.aspx)-->Now again Add **Site Map** File in solution Explorer-->open **web.sitemap** file-->write the following codes , which are given below:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="SiteMap.aspx" title="Country" description="">
    <siteMapNode url="page1.aspx" title="India" description="" />
    <siteMapNode url="page2.aspx" title="China" description="" />
    <siteMapNode url="page3.aspx" title="US" description="" />
  </siteMapNode>
</siteMap>
```

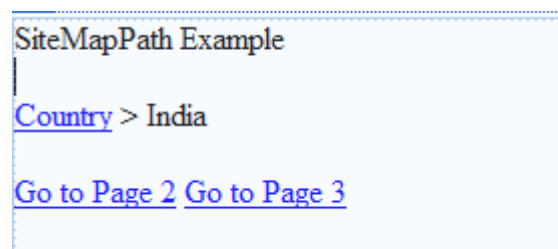
Step 2: Now drag and drop **SiteMapPath** control on the web Form (SiteMap.aspx)-->Now drag and drop **HyperLink** control on the Form as shown below:



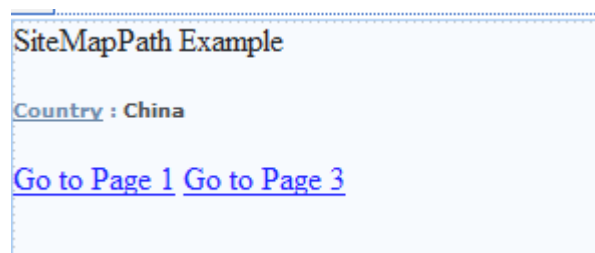
Step 3: Now Add three more **Web Form** (page1.aspx ,page2.aspx, page3.aspx) in Solution Explorer-->Go**properties** of **HyperLink Button** control -->set **NavigateUrl**-->Write **Text** Information ,as shown below:



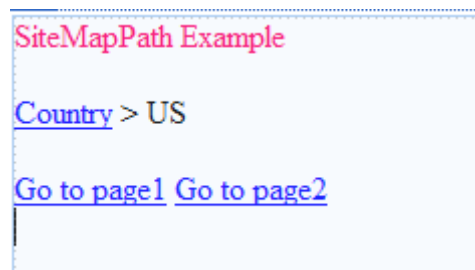
Step 4: Now Go **page1.aspx** -->drag and drop **SiteMapPath** control and HyperLink control on the Form as shown below-->Set the **NavigateUrl** of each **HyperLink** control as previous i have done.



Step 5: Now Go **page2.aspx** -->Same steps perform as step 4.

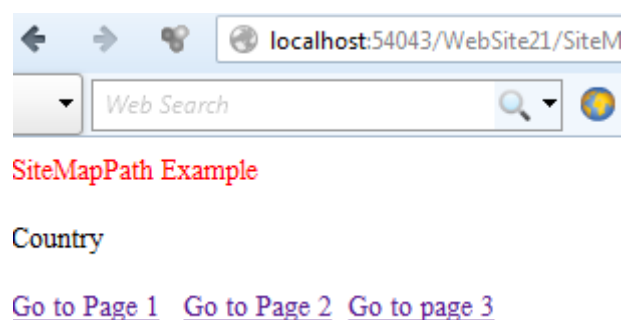


Step 6: Now Go **page3.aspx** -->Same steps perform as step 4 and step 5.



Step 7: Now Run the program(press F5).

Output:-



- **Site Maps In ASP.NET:**

Creating high quality content for web site requires hard work. Because of that, webmasters usually want to be sure that all produced content is visible to web spiders so it can be indexed and possibly attract visits from popular search engines, like Google, Live or Yahoo. Webmasters often solved this problem by adding one more page, named site map page. That was common HTML page which contains hyperlinks to all other pages on web site, so visitor or web spider could find all content if classic navigation through menus didn't work properly.

Google recognized this problem and introduced site maps formatted as XML files. Site maps are accepted later by Microsoft and Yahoo. ASP.NET 2.0 goes one step beyond and provides new .sitemap XML files that work with Menu, TreeView and SiteMapPath controls to enable easier navigation. Although Google site map and ASP.NET .sitemap are both XML files, they don't use same schema.

- **Introduction to Membership:**

ASP.NET membership gives you a built-in way to validate and store user credentials. ASP.NET membership therefore helps you manage user authentication in your Web sites. You can use ASP.NET membership with ASP.NET forms authentication by using with the ASP.NET login controls to create a complete system for authenticating users.

ASP.NET membership supports facilities for:

- Creating new users and passwords.
- Storing membership information (user names, passwords, and supporting data) in Microsoft SQL Server, Active Directory, or an alternative data store.

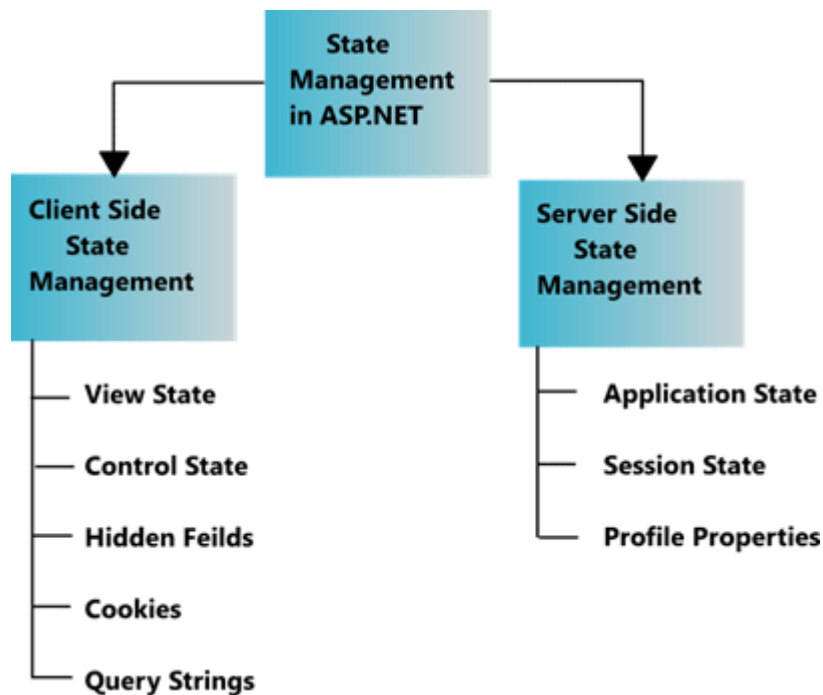
- Authenticating users who visit your site. You can authenticate users programmatically, or you can use the ASP.NET login controls to create a complete authentication system that requires little or no code.
- Managing passwords, which includes creating, changing, and resetting them . Depending on membership options you choose, the membership system can also provide an automated password-reset system that takes a user-supplied question and response.
- Exposing a unique identification for authenticated users that you can use in your own applications and that also integrates with the ASP.NET personalization and role-management (authorization) systems.

To use membership, you must first configure it for your site. The following are the basic steps you follow in order to configure membership:

1. Specify membership options as part of your Web site configuration. By default, membership is enabled. You can also specify what membership provider you want to use. The default provider stores membership information in a Microsoft SQL Server database. However, you can use other providers as well, such as a provider for [Windows Live ID](#). You can also choose to use Active Directory to store membership information, or you can specify a custom provider. For information about membership configuration options that can be specified in the Web.config file for your ASP.NET application, see [Configuring an ASP.NET Application to Use Membership](#).
2. Configure your application to use forms authentication (as distinct from Windows or [Windows Live ID](#) authentication).
You can also specify that some pages or folders in your application are protected and are accessible only to authenticated users. For more information, see [Walkthrough: Managing Web Site Users with Roles](#).
3. Define user accounts for membership. You can do this in a variety of ways. You can use the Web Site Administration Tool, which provides a wizard-like interface for creating new users. Alternatively, you can use an ASP.NET Web page where you collect a user name and password (and optionally an email address), and then use the [CreateUser](#) membership method to create a new user in the membership system.

- **State Management Techniques**

They are classified into the following 2 categories:



Now I am explaining what View State is.

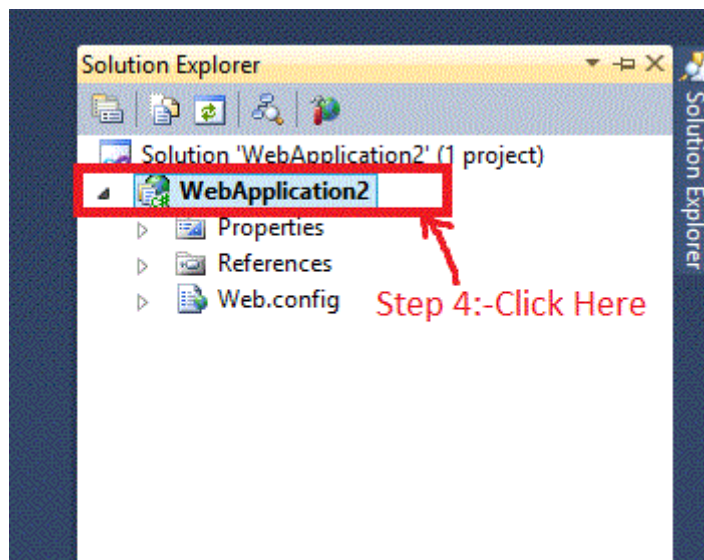
View State

View State is the method to preserve the Value of the Page and Controls between round trips. It is a Page-Level State Management technique. View State is turned on by default and normally serializes the data in every control on the page regardless of whether it is actually used during a post-back.

Now I am showing you an example of what the problem is when we don't use view state.

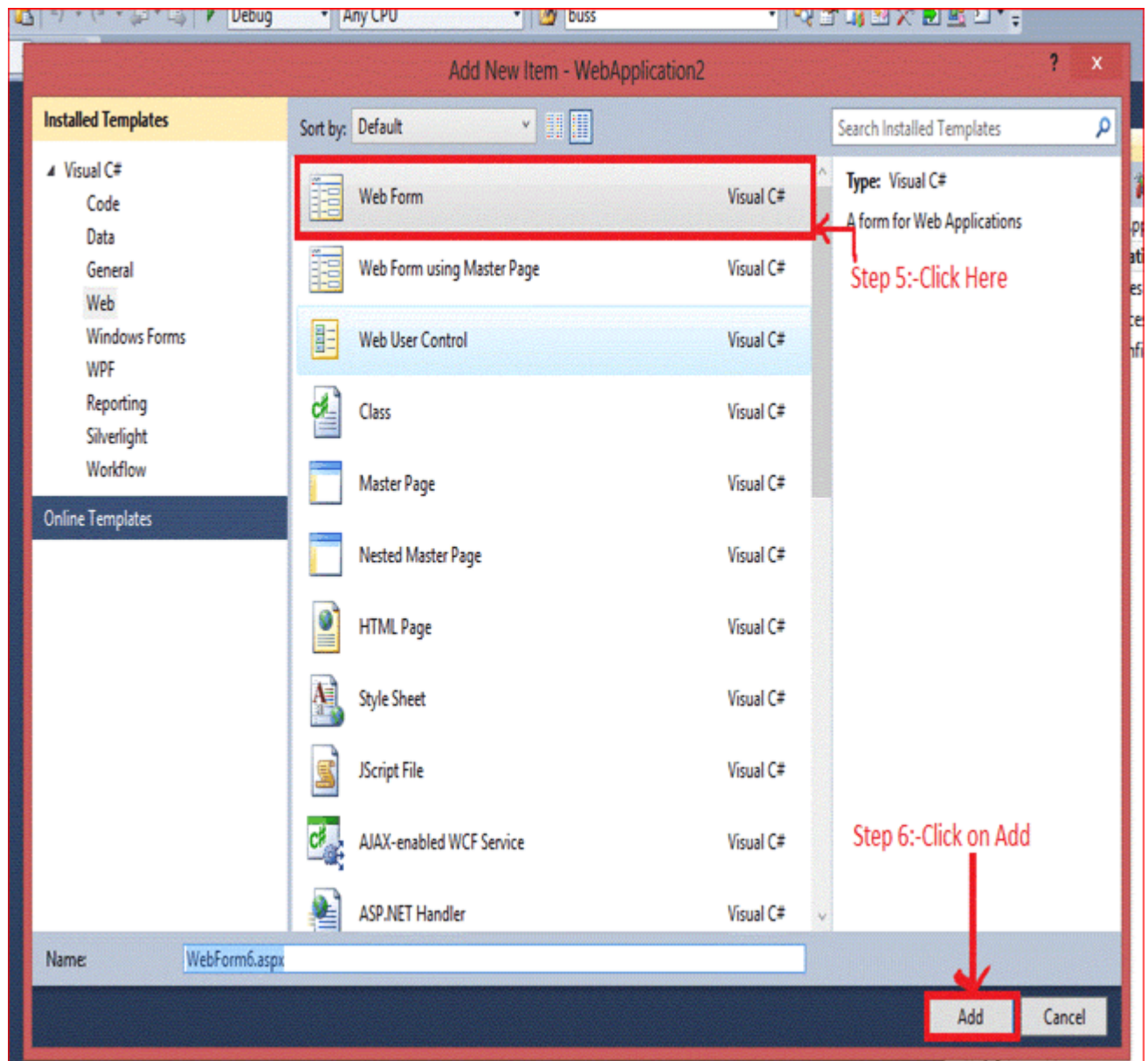
Step 1

Open Visual Studio 2010.



Step 4

Now right-click on the "ADD" > "New Item" > "Web Form" and **add** the name of the Web Form just like I did in WebForm6.aspx.



Step 5

After adding the WebForm6.aspx you will see the following code:

1. `<%@ page language="C#" autoeventwireup="true" codebehind="WebForm6.aspx.cs" inherits="view_state.WebForm6" %>`
- 2.
3. `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`
4. `<html xmlns="http://www.w3.org/1999/xhtml">`
5. `<head runat="server">`
6. `<title></title>`
7. `</head>`
8. `<body>`

```

9.   <form id="form1" runat="server">
10.  <div>
11.      User Name:-<asp:textbox id="TextBox1" runat="server"></asp:textbox>
12.      <br />
13.      Password :-<asp:textbox id="TextBox2" runat="server"></asp:textbox>
14.      <br />
15.      <asp:button id="Button1" runat="server" onclick="Button1_Click" text="Submit"
        />
16.      <asp:button id="Button3" runat="server" onclick="Button3_Click" text="Restore
        " />
17.  </div>
18. </form>
19.</body>
20.</html>

```

Now write the code as in the following:

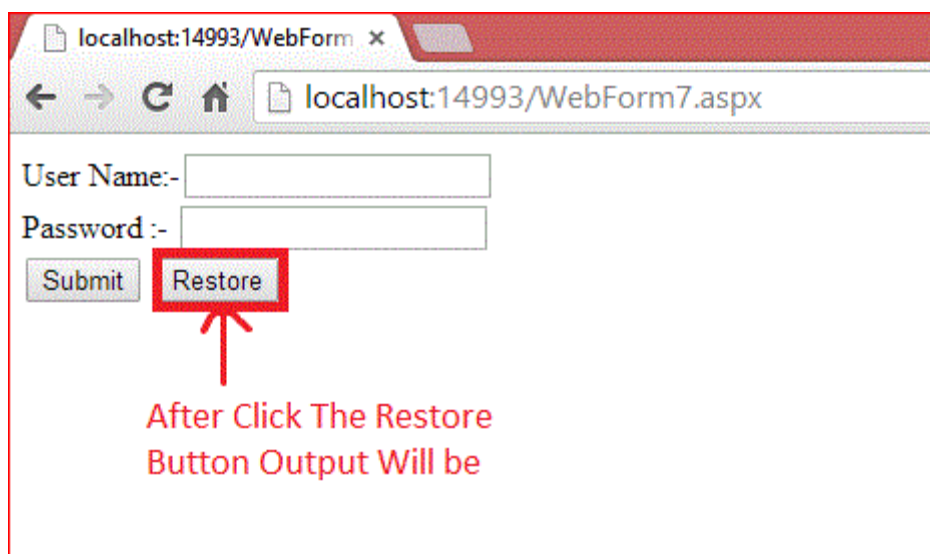
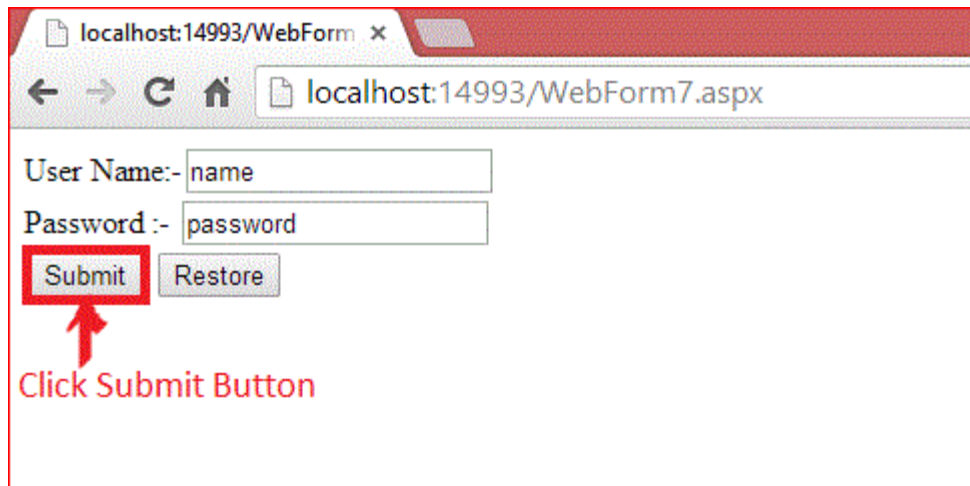
```

1. //Declaration of a and b
2. public string a, b;
3. protected void Button1_Click(object sender, EventArgs e)
4. {
5.     //TextBox1 and TextBox2 Value is Assigning on the variable a and b
6.     a = TextBox1.Text;
7.     b = TextBox2.Text;
8.     //after clicking on Button TextBox value Will be Cleared
9.     TextBox1.Text = TextBox2.Text = string.Empty;
10. }
11.
12. protected void Button3_Click(object sender, EventArgs e)
13. {
14.     //value of variable a and b is assigning on TextBox1 and Textbox2
15.     TextBox1.Text = a;
16.     TextBox2.Text = b;
17. }

```

Output

Now the output is:



It only happens because all the controls are classes and on the server all the Control Objects are created and then after the round trip the Page is returned to the client's browser in HTML format and the objects are destroyed at the server.

After the Submit button is clicked the value of user name and password is submitted to the server. We cannot restore the value again because after the postback the instance of the control is destroyed and on clicking of the Restore Button the server takes a new request and the server cannot restore the value of the TextBox.

Features Of View State

These are the main features of view state:

1. Retains the value of the Control after post-back without using a session.
2. Stores the value of Pages and Control Properties defined in the page.
3. Creates a custom View State Provider that lets you store View State Information in a SQL Server Database or in another data store.

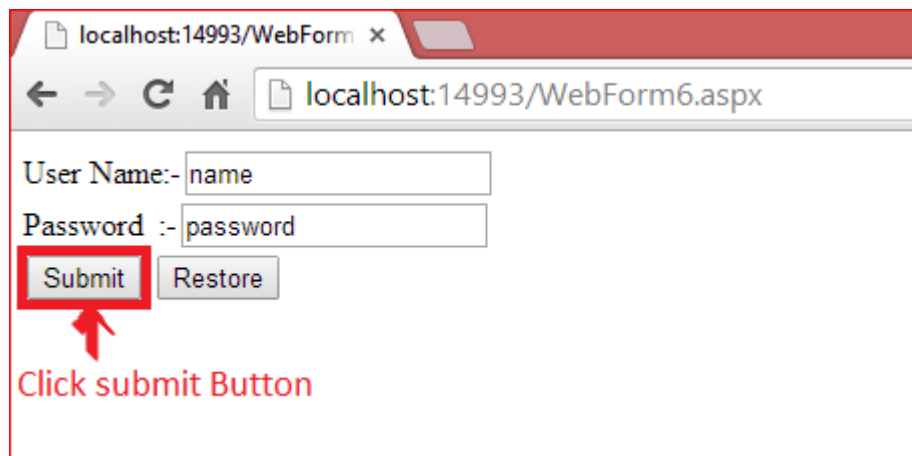
And now I am explaining the stored value in the View State and the remaining steps are the same as the previous.

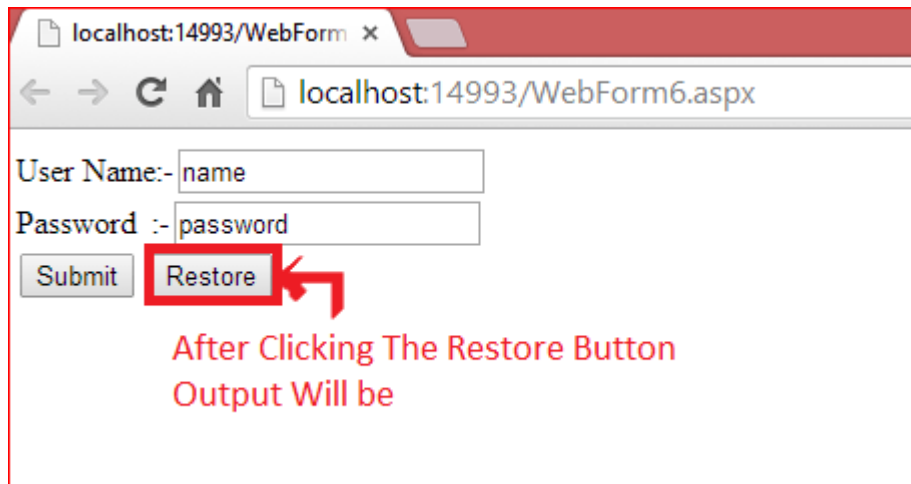
Now write this code,

```
1. protected void Button1_Click(object sender, EventArgs e)
2. {
3.     //Value of Textbox1 and TextBox2 is assign on the ViewState
4.     ViewState["name"] = TextBox1.Text;
5.     ViewState["password"] = TextBox2.Text;
6.     //after clicking on Button TextBox value Will be Cleared
7.     TextBox1.Text = TextBox2.Text = string.Empty;
8. }
9. protected void Button3_Click(object sender, EventArgs e)
10. {
11.     //If ViewState Value is not Null then Value of View State is Assign to TextBox
12.     if (ViewState["name"] != null)
13.     {
14.         TextBox1.Text = ViewState["name"].ToString();
15.     }
16.     if (ViewState["password"] != null)
17.     {
18.         TextBox2.Text = ViewState["password"].ToString();
19.     }
20. }
```

Output

Now the output is,





After clicking on the Submit Button the value of user name and password is submitted in View State and the View State stores the value of user name and password during post-back.

After click on the Restore Button we can get the value again. The Value must be retained during post-back and the values are stored into a base 64 encoded string and this information is then put into the View State Hidden Field.

Data Objects That Can be Stored in View state

1. String
2. Boolean Value
3. Array Object
4. Array List Object
5. Hash Table
6. Custom type Converters

Advantages of View State

1. Easy to Implement.
2. No server resources are required: The View State is contained in a structure within the page load.
3. Enhanced security features: It can be encoded and compressed or Unicode implementation.

Disadvantages of View State

1. Security Risk: The Information of View State can be seen in the page output source directly. You can manually encrypt and decrypt the contents of a Hidden Field, but It requires extra coding. If security is a concern then consider using a Server-Based state Mechanism so that no sensitive information is sent to the client.
2. Performance: Performance is not good if we use a large amount of data because View State is stored in the page itself and storing a large value can cause the page to be slow.
3. Device limitation: Mobile Devices might not have the memory capacity to store a large amount of View State data.
4. It can store values for the same page only.