Unit IV

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

ASP.NET directives are instructions to specify optional settings, such as registering a custom control and page language. These settings describe how the web forms (.aspx) or user controls (.ascx) pages are processed by the .Net framework.

The syntax for declaring a directive is:

```
<%@  directive_name attribute=value  [attribute=value]  %>
```

The Application Directive

The Application directive defines application-specific attributes. It is provided at the top of the global.aspx file.

The basic syntax of Application directive is:

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

The attributes of the Application directive are:

Attributes	Description
Inherits	The name of the class from which to inherit.
Description	The text description of the application. Parsers and compilers ignore this.

Language	The language used in code blocks.

The Assembly Directive

The Assembly directive links an assembly to the page or the application at parse time. This could appear either in the global.asax file for application-wide linking, in the page file, a user control file for linking to a page or user control.

The basic syntax of Assembly directive is:

<%@ Assembly Name ="myassembly" %>

The attributes of the Assembly directive are:

Attributes	Description
Name	The name of the assembly to be linked.
Src	The path to the source file to be linked and compiled dynamically.

The Control Directive

The control directive is used with the user controls and appears in the user control (.ascx) files.

The basic syntax of Control directive is:

<%@ Control Language="C#" EnableViewState="false" %>

The attributes of the Control directive are:

Attributes	Description
AutoEventWireup	The Boolean value that enables or disables automatic association of events to handlers.
ClassName	The file name for the control.
Debug	The Boolean value that enables or disables compiling with debug symbols.
Description	The text description of the control page, ignored by compiler.
EnableViewState	The Boolean value that indicates whether view state is maintained across page requests.

Explicit	For VB language, tells the compiler to use option explicit mode.
Inherits	The class from which the control page inherits.
Language	The language for code and script.
Src	The filename for the code-behind class.
Strict	For VB language, tells the compiler to use the option strict mode.

The Implements Directive

The Implement directive indicates that the web page, master page or user control page must implement the specified .Net framework interface.

The basic syntax for implements directive is:

<%@ Implements Interface="interface_name" %>

The Import Directive

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

The basic syntax for import directive is:

<%@ namespace="System.Drawing" %>

The Master Directive

The Master directive specifies a page file as being the mater page.

The basic syntax of sample MasterPage directive is:

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

The MasterType Directive

The MasterType directive assigns a class name to the Master property of a page, to make it strongly typed.

The basic syntax of MasterType directive is:

<%@ MasterType attribute="value"[attribute="value" ...] %>

The OutputCache Directive

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

The basic syntax of OutputCache directive is:

```
<%@ OutputCache Duration="15" VaryByParam="None"  %>
```

The Page Directive

The Page directive defines the attributes specific to the page file for the page parser and the compiler.

The basic syntax of Page directive is:

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

The attributes of the Page directive are:

Attributes	Description
AutoEventWireup	The Boolean value that enables or disables page events that are being automatically bound to methods; for example, Page_Load.
Buffer	The Boolean value that enables or disables HTTP response buffering.
ClassName	The class name for the page.
ClientTarget	The browser for which the server controls should render content.
CodeFile	The name of the code behind file.
Debug	The Boolean value that enables or disables compilation with debug symbols.
Description	The text description of the page, ignored by the parser.
EnableSessionState	It enables, disables, or makes session state read-only.
EnableViewState	The Boolean value that enables or disables view state across page requests.

ErrorPage	URL for redirection if an unhandled page exception occurs.
Inherits	The name of the code behind or other class.
Language	The programming language for code.
Src	The file name of the code behind class.
Trace	It enables or disables tracing.
TraceMode	It indicates how trace messages are displayed, and sorted by time or category.
Transaction	It indicates if transactions are supported.
ValidateRequest	The Boolean value that indicates whether all input data is validated against a hardcoded list of values.

The PreviousPageType Directive

The PreviousPageType directive assigns a class to a page, so that the page is strongly typed.

The basic syntax for a sample PreviousPagetype directive is:

<%@ PreviousPageType attribute="value"[attribute="value" ...]   %>

The Reference Directive

The Reference directive indicates that another page or user control should be compiled and linked to the current page.

The basic syntax of Reference directive is:

<%@ Reference Page ="somepage.aspx" %>

The Register Directive

The Register derivative is used for registering the custom server controls and user controls.

The basic syntax of Register directive is:

<%@ Register Src="~/footer.ascx" TagName="footer" TagPrefix="Tfooter" %>

Code Render Blocks

If you've had experience with traditional ASP, you might recognize these blocks. You can use code render blocks to define inline code or expressions that will execute when a page is rendered. Code within a code render block is executed immediately when it is encountered—usually when the page is loaded or rendered. On the other hand, code within a code *declaration* block (within <script> tags) is executed only when it is called or triggered by user or page interactions. There are two types of code render blocks—inline code, and inline expressions—both of which are typically written within the body of the ASP.NET page.

Inline code render blocks execute one or more statements, and are placed directly inside a page's HTML between <% and %> delimiters. In our example, the following is a code render block:

In VB.Net

<% Dim Title As String = "This is generated by a code render " & _ "block." %>

In C#.Net

<% string Title = "This is generated by a code render block."; %>

ASP.NET Server Controls

At the heart of any ASP.NET page lie server controls, which represent dynamic elements with which your users can interact. There are three basic types of server control: ASP.NET controls, HTML controls, and web user controls.

Usually, an ASP.NET control must reside within a <form runat="server"> tag in order to function correctly. Controls offer the following advantages to ASP.NET developers:

They give us the ability to access HTML elements easily from within our code: we can change these elements' characteristics, check their values, or even update them dynamically from our server-side programming language of choice.

ASP.NET controls retain their properties thanks to a mechanism called **view state**. We'll be covering view state later in this chapter. For now, you need to know that view state prevents users from losing the data they've entered into a form once that form has been sent to the server for processing. When the response comes back to the client, text box entries, drop-down list selections, and so on, are all retained through view state.

With ASP.NET controls, developers are able to separate a page's presentational elements (everything the user sees) from its application logic (the dynamic portions of the ASP.NET page), so that each can be considered separately.

Many ASP.NET controls can be "bound" to the data sources from which they will extract data for display with minimal (if any) coding effort.

Server-side Comments

Server-side comments allow you to include within the page comments or notes that will not be processed by ASP.NET. Traditional HTML uses the <!-- and - -> character sequences to delimit comments; any information included between these tags will not be displayed to the user. ASP.NET comments look very similar, but use the sequences <%-- and --%>.

Our ASP.NET example contains the following server-side comment block:

File: Hello.aspx (excerpt)

<%-- Declare the title as string and set it --%>

The difference between ASP.NET comments and HTML comments is that ASP.NET comments are not sent to the client at all; HTML comments are, so they're not suited to commenting out ASP.NET code. Consider the following example:

C#

```
<!--

<% string Title = "This is generated by a code render block."; %> <%= Title %>

-->
```

Here, it looks as if a developer has attempted to use an HTML comment to stop a code render block from being executed. Unfortunately, HTML comments will only hide information from the browser, not the ASP.NET runtime. So, in this case, while we won't see anything in the browser that represents these two lines, they will be processed by ASP.NET, and the value of the variable Title will be sent to the browser inside an HTML comment, as shown here:

```
<!–-

This code generated by a code render block. -->
```

The code could be modified to use server-side comments very simply:

Literal Text and HTML Tags

The final elements of an ASP.NET page are plain old text and HTML. Generally, you can't do without these elements—after all, HTML allows the display of the information in your ASP.NET controls and code in a way that's suitable for users. Let's take a look at the literal text and HTML tags that were used to produce the display in Figure 2.2:

In VB

Visual Basic File: **Hello.aspx (excerpt)**

<%@ Page Language="VB" %>

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

<html>

<head>

```
<title>Sample Page</title>
<script runat="server"> Sub Page_Load()
messageLabel.Text = "Hello World!" End Sub
</script>
</head>
<body>
<form runat="server">
<p>
<asp:Label id="messageLabel" runat="server" />
</p>
<p>
<%-- Declare the title as string and set it --%>
<% Dim Title As String = "This is generated by a " & _ "code render block." %>
<%= Title %>
</p>
</form>
</body>
</html>
```

In C#

C# File: Hello.aspx (excerpt)

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<title>Sample Page</title>
<script runat="server"> void Page_Load()
{
messageLabel.Text = "Hello World";
}
```

```
</script>
</head>
<body>
<form runat="server">
<p>
<asp:Label id="messageLabel" runat="server" />
</p>
<p>
<%-- Declare the title as string and set it --%>
<% string Title = "This is generated by a code render " + "block."; %>
<%= Title %>
</p>
</form>
</body>
</html>
```

The bold code above highlights the fact that literal text and HTML tags provide the structure for presenting our dynamic data. Without these elements, this page would have no format, and the browser would be unable to understand it.

ASP.NET Page Life Cycle Events

At each stage of the page life cycle, the page raises some events, which could be coded. An event handler is basically a function or subroutine, bound to the event, using declarative attributes such as Onclick or handle.

Following are the page life cycle events:

- **PreInit** - PreInit is the first event in page life cycle. It checks the IsPostBack property and determines whether the page is a postback. It sets the themes and master pages, creates dynamic controls, and gets and sets profile property values. This event can be handled by overloading the OnPreInit method or creating a Page_PreInit handler.
- **Init** - Init event initializes the control property and the control tree is built. This event can be handled by overloading the OnInit method or creating a Page_Init handler.
- **InitComplete** - InitComplete event allows tracking of view state. All the controls turn on view-state tracking.
- **LoadViewState** - LoadViewState event allows loading view state information into the controls.

- **LoadPostData -** During this phase, the contents of all the input fields are defined with the <form> tag are processed.
- **PreLoad -** PreLoad occurs before the post back data is loaded in the controls. This event can be handled by overloading the OnPreLoad method or creating a Page_PreLoad handler.
- **Load -** The Load event is raised for the page first and then recursively for all child controls. The controls in the control tree are created. This event can be handled by overloading the OnLoad method or creating a Page_Load handler.
- **LoadComplete -** The loading process is completed, control event handlers are run, and page validation takes place. This event can be handled by overloading the OnLoadComplete method or creating a Page_LoadComplete handler
- **PreRender** - The PreRender event occurs just before the output is rendered. By handling this event, pages and controls can perform any updates before the output is rendered.
- **PreRenderComplete** - As the PreRender event is recursively fired for all child controls, this event ensures the completion of the pre-rendering phase.
- **SaveStateComplete** - State of control on the page is saved. Personalization, control state and view state information is saved. The HTML markup is generated. This stage can be handled by overriding the Render method or creating a Page_Render handler.
- **UnLoad** - The UnLoad phase is the last phase of the page life cycle. It raises the UnLoad event for all controls recursively and lastly for the page itself. Final cleanup is done and all resources and references, such as database connections, are freed. This event can be handled by modifying the OnUnLoad method or creating a Page_UnLoad handler.

Validation controls

ASP.NET validation controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored.

ASP.NET provides the following validation controls:

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

BaseValidator Class

The validation control classes are inherited from the BaseValidator class hence they inherit its properties and methods. Therefore, it would help to take a look at the properties and the methods of this base class, which are common for all the validation controls:

Members	Description
ControlToValidate	Indicates the input control to validate.
Display	Indicates how the error message is shown.
EnableClientScript	Indicates whether client side validation will take.
Enabled	Enables or disables the validator.
ErrorMessage	Indicates error string.
Text	Error text to be shown if validation fails.
IsValid	Indicates whether the value of the control is valid.
SetFocusOnError	It indicates whether in case of an invalid control, the focus should switch to the related input control.
ValidationGroup	The logical group of multiple validators, where this control belongs.
Validate()	This method revalidates the control and updates the IsValid property.

RequiredFieldValidator Control

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

The syntax of the control is as given:

```
<asp:RequiredFieldValidator ID="rfvcandidate"  runat="server" ControlToValidate ="ddlcandidate"
ErrorMessage="Please choose a candidate" InitialValue="Please choose a candidate">
</asp:RequiredFieldValidator>
```

RangeValidator Control

The RangeValidator control verifies that the input value falls within a predetermined range.

It has three specific properties:

Properties	Description
Type	It defines the type of the data. The available values are: Currency, Date, Double, Integer, and String.
MinimumValue	It specifies the minimum value of the range.
MaximumValue	It specifies the maximum value of the range.

The syntax of the control is as given:

```
<asp:RangeValidator ID="rvclass" runat="server" ControlToValidate="txtclass"
   ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"
   MinimumValue="6" Type="Integer">
   </asp:RangeValidator>
```

CompareValidator Control

The CompareValidator control compares a value in one control with a fixed value or a value in another control.

It has the following specific properties:

Properties	Description
Type	It specifies the data type.
ControlToCompare	It specifies the value of the input control to compare with.
ValueToCompare	It specifies the constant value to compare with.
Operator	It specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and DataTypeCheck.

The basic syntax of the control is as follows:

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
  ErrorMessage="CompareValidator"></asp:CompareValidator>
```

RegularExpressionValidator

The RegularExpressionValidator allows validating the input text by matching against a pattern of a regular expression. The regular expression is set in the ValidationExpression property.

The following table summarizes the commonly used syntax constructs for regular expressions:

Character Escapes	**Description**
\b	Matches a backspace.
\t	Matches a tab.
\r	Matches a carriage return.
\v	Matches a vertical tab.
\f	Matches a form feed.
\n	Matches a new line.
\	Escape character.

Apart from single character match, a class of characters could be specified that can be matched, called the metacharacters.

Metacharacters	**Description**
.	Matches any character except \n.
[abcd]	Matches any character in the set.
[^abcd]	Excludes any character in the set.
[2-7a-mA-M]	Matches any character specified in the range.
\w	Matches any alphanumeric character and underscore.

\W	Matches any non-word character.
\s	Matches whitespace characters like, space, tab, new line etc.
\S	Matches any non-whitespace character.
\d	Matches any decimal character.
\D	Matches any non-decimal character.

Quantifiers could be added to specify number of times a character could appear.

Quantifier	**Description**
*	Zero or more matches.
+	One or more matches.
?	Zero or one matches.
{N}	N matches.
{N,}	N or more matches.
{N,M}	Between N and M matches.

The syntax of the control is as given:

```
<asp:RegularExpressionValidator ID="string" runat="server" ErrorMessage="string"
   ValidationExpression="string" ValidationGroup="string">
</asp:RegularExpressionValidator>
```

CustomValidator

The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation.

The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control's ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

The basic syntax for the control is as given:

```
<asp:CustomValidator ID="CustomValidator1" runat="server"
   ClientValidationFunction=.cvf_func. ErrorMessage="CustomValidator">
</asp:CustomValidator>
```

ValidationSummary

The ValidationSummary control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

The following two mutually inclusive properties list out the error message:

- **ShowSummary** : shows the error messages in specified format.
- **ShowMessageBox** : shows the error messages in a separate window.

The syntax for the control is as given:

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
   DisplayMode = "BulletList" ShowSummary = "true" HeaderText="Errors:" />
```

Validation Groups

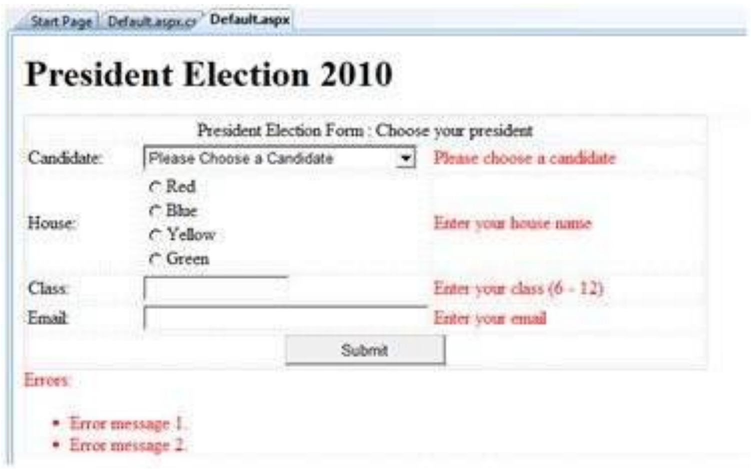
Complex pages have different groups of information provided in different panels. In such situation, a need might arise for performing validation separately for separate group. This kind of situation is handled using validation groups.

To create a validation group, you should put the input controls and the validation controls into the same logical group by setting their *ValidationGroup* property.

Example

The following example describes a form to be filled up by all the students of a school, divided into four houses, for electing the school president. Here, we use the validation controls to validate the user input.

This is the form in design view:



President Election 2010

President Election Form : Choose your president		
Candidate:	Please Choose a Candidate	Please choose a candidate
House:	○ Red ○ Blue ○ Yellow ○ Green	Enter your house name
Class:		Enter your class (6 - 12)
Email:		Enter your email
	Submit	

Errors:

- Error message 1.
- Error message 2.

The content file code is as given:

```
<form id="form1" runat="server">
  <table style="width: 66%;">
    <tr>
      <td class="style1" colspan="3" align="center">
      <asp:Label ID="lblmsg"
        Text="President Election Form : Choose your president"
        runat="server" />
      </td>
    </tr>
    <tr>
      <td class="style3">
        Candidate:
      </td>
      <td class="style2">
        <asp:DropDownList ID="ddlcandidate" runat="server"  style="width:239px">
          <asp:ListItem>Please Choose a Candidate</asp:ListItem>
          <asp:ListItem>M H Kabir</asp:ListItem>
          <asp:ListItem>Steve Taylor</asp:ListItem>
          <asp:ListItem>John Abraham</asp:ListItem>
          <asp:ListItem>Venus Williams</asp:ListItem>
        </asp:DropDownList>
      </td>
      <td>
        <asp:RequiredFieldValidator ID="rfvcandidate"
          runat="server" ControlToValidate ="ddlcandidate"
          ErrorMessage="Please choose a candidate"
          InitialValue="Please choose a candidate">
        </asp:RequiredFieldValidator>
      </td>
    </tr>

    <tr>
      <td class="style3">
        House:
      </td>

      <td class="style2">
        <asp:RadioButtonList ID="rblhouse" runat="server" RepeatLayout="Flow">
          <asp:ListItem>Red</asp:ListItem>
          <asp:ListItem>Blue</asp:ListItem>
          <asp:ListItem>Yellow</asp:ListItem>
          <asp:ListItem>Green</asp:ListItem>
        </asp:RadioButtonList>
      </td>

      <td>
        <asp:RequiredFieldValidator ID="rfvhouse" runat="server"
          ControlToValidate="rblhouse" ErrorMessage="Enter your house name" >
        </asp:RequiredFieldValidator>
        <br />
      </td>
    </tr>

    <tr>
      <td class="style3">
        Class:
      </td>

      <td class="style2">
        <asp:TextBox ID="txtclass" runat="server"></asp:TextBox>
      </td>

      <td>
        <asp:RangeValidator ID="rvclass"
          runat="server" ControlToValidate="txtclass"
          ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"
          MinimumValue="6" Type="Integer">
        </asp:RangeValidator>
      </td>
    </tr>

    <tr>
      <td class="style3">
        Email:
      </td>

      <td class="style2">
        <asp:TextBox ID="txtemail" runat="server" style="width:250px">
        </asp:TextBox>
```

```
      </td>

      <td>
        <asp:RegularExpressionValidator ID="remail" runat="server"
          ControlToValidate="txtemail" ErrorMessage="Enter your email"
          ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
        </asp:RegularExpressionValidator>
      </td>
    </tr>

    <tr>
      <td class="style3" align="center" colspan="3">
        <asp:Button ID="btnsubmit" runat="server" onclick="btnsubmit_Click"
          style="text-align: center" Text="Submit" style="width:140px" />
      </td>
    </tr>
  </table>
  <asp:ValidationSummary ID="ValidationSummary1" runat="server"
    DisplayMode ="BulletList" ShowSummary ="true" HeaderText="Errors:" />
</form>
```

The code behind the submit button:

```
protected void btnsubmit_Click(object sender, EventArgs e)
{
  if (Page.IsValid)
  {
    lblmsg.Text = "Thank You";
  }
  else
  {
    lblmsg.Text = "Fill up all the fields";
  }
}
```

Rich Controls

ASP.NET provides large set of controls. These controls are divided into different categories, depends upon their functionalities. The followings control comes under the rich controls category.

- FileUpload control
- Calendar control
- AdRotator control
- MultiView control
- Wizard control

FileUpload control

The FileUpload control allows the user to browse for and select the file to be uploaded, providing a browse button and a text box for entering the filename.

Once, the user has entered the filename in the text box by typing the name or browsing, the SaveAs method of the FileUpload control can be called to save the file to the disk.

The basic syntax of FileUpload is:

<asp:FileUpload ID= "Uploader" runat = "server" />

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

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

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

The HttpPostedFile class has the following frequently used properties:

Properties	Description
ContentLength	Returns the size of the uploaded file in bytes.
ContentType	Returns the MIME type of the uploaded file.
FileName	Returns the full filename.

InputStream	Returns a stream object pointing to the uploaded file.

The AdRotator control

The AdRotator control randomly selects banner graphics from a list, which is specified in an external XML schedule file. This external XML schedule file is called the advertisement file.

The AdRotator control allows you to specify the advertisement file and the type of window that the link should follow in the AdvertisementFile and the Target property respectively.

The basic syntax of adding an AdRotator is as follows:

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

Before going into the details of the AdRotator control and its properties, let us look into the construction of the advertisement file.

The Advertisement File

The advertisement file is an XML file, which contains the information about the advertisements to be displayed.

Extensible Markup Language (XML) is a W3C standard for text document markup. It is a text-based markup language that enables you to store data in a structured format by using meaningful tags. The term 'extensible' implies that you can extend your ability to describe a document by defining meaningful tags for the application.

There are the following standard XML elements that are commonly used in the advertisement file:

Element	Description
Advertisements	Encloses the advertisement file.
Ad	Delineates separate ad.
ImageUrl	The path of image that will be displayed.
NavigateUrl	The link that will be followed when the user clicks the ad.
AlternateText	The text that will be displayed instead of the picture if it cannot be displayed.
Keyword	Keyword identifying a group of advertisements. This is used for filtering.

Impressions	The number indicating how often an advertisement will appear.
Height	Height of the image to be displayed.
Width	Width of the image to be displayed.

Apart from these tags, customs tags with custom attributes could also be included. The following code illustrates an advertisement file ads.xml:

```
<Advertisements>
  <Ad>
    <ImageUrl>rose1.jpg</ImageUrl>
    <NavigateUrl>http://www.1800flowers.com</NavigateUrl>
    <AlternateText>
      Order flowers, roses, gifts and more
    </AlternateText>
    <Impressions>20</Impressions>
    <Keyword>flowers</Keyword>
  </Ad>

  <Ad>
    <ImageUrl>rose2.jpg</ImageUrl>
    <NavigateUrl>http://www.babybouquets.com.au</NavigateUrl>
    <AlternateText>Order roses and flowers</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>gifts</Keyword>
  </Ad>

  <Ad>
    <ImageUrl>rose3.jpg</ImageUrl>
    <NavigateUrl>http://www.flowers2moscow.com</NavigateUrl>
    <AlternateText>Send flowers to Russia</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>russia</Keyword>
  </Ad>
  <Ad>
    <ImageUrl>rose4.jpg</ImageUrl>
    <NavigateUrl>http://www.edibleblooms.com</NavigateUrl>
    <AlternateText>Edible Blooms</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>gifts</Keyword>
  </Ad>
</Advertisements>
```

Properties and Events of the AdRotator Class

The AdRotator class is derived from the WebControl class and inherits its properties. Apart from those, the AdRotator class has the following properties:

Properties	Description
AdvertisementFile	The path to the advertisement file.
AlternateTextFeild	The element name of the field where alternate text is provided. The default value is AlternateText.
DataMember	The name of the specific list of data to be bound when advertisement file is not used.
DataSource	Control from where it would retrieve data.
DataSourceID	Id of the control from where it would retrieve data.
Font	Specifies the font properties associated with the advertisement banner control.
ImageUrlField	The element name of the field where the URL for the image is provided. The default value is ImageUrl.
KeywordFilter	For displaying the keyword based ads only.
NavigateUrlField	The element name of the field where the URL to navigate to is provided. The default value is NavigateUrl.

Target	The browser window or frame that displays the content of the page linked.
UniqueID	Obtains the unique, hierarchically qualified identifier for the AdRotator control.

Following are the important events of the AdRotator class:

Events	Description
AdCreated	It is raised once per round trip to the server after creation of the control, but before the page is rendered
DataBinding	Occurs when the server control binds to a data source.
DataBound	Occurs after the server control binds to a data source.
Disposed	Occurs when a server control is released from memory, which is the last stage of the server control lifecycle when an ASP.NET page is requested
Init	Occurs when the server control is initialized, which is the first step in its lifecycle.
Load	Occurs when the server control is loaded into the Page object.
PreRender	Occurs after the Control object is loaded but prior to rendering.
Unload	Occurs when the server control is unloaded from memory.

Working with AdRotator Control

Create a new web page and place an AdRotator control on it.

```
<form id="form1" runat="server">
   <div>
      <asp:AdRotator ID="AdRotator1" runat="server" AdvertisementFile  ="~/ads.xml"
onadcreated="AdRotator1_AdCreated" />
   </div>
```

</form>

calendar control

calendar control is a functionally rich web control, which provides the following capabilities:

- Displaying one month at a time
- Selecting a day, a week or a month
- Selecting a range of days
- Moving from month to month
- Controlling the display of the days programmatically

The basic syntax of a calendar control is:

```
<asp:Calender ID = "Calendar1" runat = "server"></asp:Calender>
```

Properties and Events of the Calendar Control

The calendar control has many properties and events, using which you can customize the actions and display of the control. The following table provides some important properties of the Calendar control:

Properties	Description
Caption	Gets or sets the caption for the calendar control.
CaptionAlign	Gets or sets the alignment for the caption.
CellPadding	Gets or sets the number of spaces between the data and the cell border.
CellSpacing	Gets or sets the space between cells.
DayHeaderStyle	Gets the style properties for the section that displays the day of the week.
DayNameFormat	Gets or sets format of days of the week.
DayStyle	Gets the style properties for the days in the displayed month.
FirstDayOfWeek	Gets or sets the day of week to display in the first column.
NextMonthText	Gets or sets the text for next month navigation control. The default value is >.

NextPrevFormat	Gets or sets the format of the next and previous month navigation control.
OtherMonthDayStyle	Gets the style properties for the days on the Calendar control that are not in the displayed month.
PrevMonthText	Gets or sets the text for previous month navigation control. The default value is <.
SelectedDate	Gets or sets the selected date.
SelectedDates	Gets a collection of DateTime objects representing the selected dates.
SelectedDayStyle	Gets the style properties for the selected dates.
SelectionMode	Gets or sets the selection mode that specifies whether the user can select a single day, a week or an entire month.
SelectMonthText	Gets or sets the text for the month selection element in the selector column.
SelectorStyle	Gets the style properties for the week and month selector column.
SelectWeekText	Gets or sets the text displayed for the week selection element in the selector column.
ShowDayHeader	Gets or sets the value indicating whether the heading for the days of the week is displayed.
ShowGridLines	Gets or sets the value indicating whether the gridlines would be shown.
ShowNextPrevMonth	Gets or sets a value indicating whether next and previous month navigation elements are shown in the title section.
ShowTitle	Gets or sets a value indicating whether the title section is displayed.
TitleFormat	Gets or sets the format for the title section.
Titlestyle	Get the style properties of the title heading for the Calendar control.

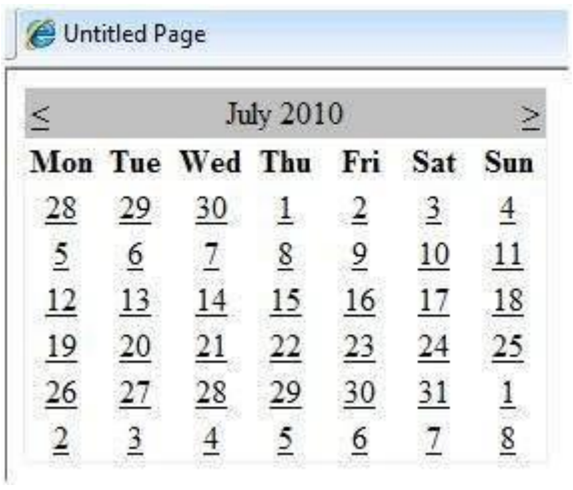
TodayDayStyle	Gets the style properties for today's date on the Calendar control.
TodaysDate	Gets or sets the value for today's date.
UseAccessibleHeader	Gets or sets a value that indicates whether to render the table header <th> HTML element for the day headers instead of the table data <td> HTML element.
VisibleDate	Gets or sets the date that specifies the month to display.
WeekendDayStyle	Gets the style properties for the weekend dates on the Calendar control.

The Calendar control has the following three most important events that allow the developers to program the calendar control. They are:

Events	**Description**
SelectionChanged	It is raised when a day, a week or an entire month is selected.
DayRender	It is raised when each data cell of the calendar control is rendered.
VisibleMonthChanged	It is raised when user changes a month.

Working with the Calendar Control

Putting a bare-bone calendar control without any code behind file provides a workable calendar to a site, which shows the months and days of the year. It also allows navigation to next and previous months.



<			July 2010			>
Mon	Tue	Wed	Thu	Fri	Sat	Sun
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Calendar controls allow the users to select a single day, a week, or an entire month. This is done by using the SelectionMode property. This property has the following values:

Properties	Description
Day	To select a single day.
DayWeek	To select a single day or an entire week.
DayWeekMonth	To select a single day, a week, or an entire month.
None	Nothing can be selected.

The syntax for selecting days:

```
<asp:Calender ID = "Calendar1" runat = "server" SelectionMode="DayWeekMonth">
</asp:Calender>
```

MultiView and View controls

MultiView and View controls allow you to divide the content of a page into different groups, displaying only one group at a time. Each View control manages one group of content and all the View controls are held together in a MultiView control.

The MultiView control is responsible for displaying one View control at a time. The View displayed is called the active view.

The syntax of MultiView control is:

```
<asp:MultView ID= "MultiView1" runat= "server"></asp:MultiView>
```

The syntax of View control is:

```
<asp:View ID= "View1" runat= "server"></asp:View>
```

However, the View control cannot exist on its own. It would render error if you try to use it stand-alone. It is always used with a Multiview control as:

```
<asp:MultView ID= "MultiView1" runat= "server">
   <asp:View ID= "View1" runat= "server"> </asp:View>
</asp:MultiView>
```

Properties of View and MultiView Controls

Both View and MultiView controls are derived from Control class and inherit all its properties, methods, and events. The most important property of the View control is Visible property of type Boolean, which sets the visibility of a view.

The MultiView control has the following important properties:

Properties	**Description**
Views	Collection of View controls within the MultiView.
ActiveViewIndex	A zero based index that denotes the active view. If no view is active, then the index is -1.

The CommandName attribute of the button control associated with the navigation of the MultiView control are associated with some related field of the MultiView control.

For example, if a button control with CommandName value as NextView is associated with the navigation of the multiview, it automatically navigates to the next view when the button is clicked.

The following table shows the default command names of the above properties:

Properties	**Description**
NextViewCommandName	NextView
PreviousViewCommandName	PrevView
SwitchViewByIDCommandName	SwitchViewByID
SwitchViewByIndexCommandName	SwitchViewByIndex

The important methods of the multiview control are:

Methods	**Description**

SetActiveview	Sets the active view
GetActiveview	Retrieves the active view

Every time a view is changed, the page is posted back to the server and a number of events are raised. Some important events are:

Events	**Description**
ActiveViewChanged	Raised when a view is changed
Activate	Raised by the active view
Deactivate	Raised by the inactive view

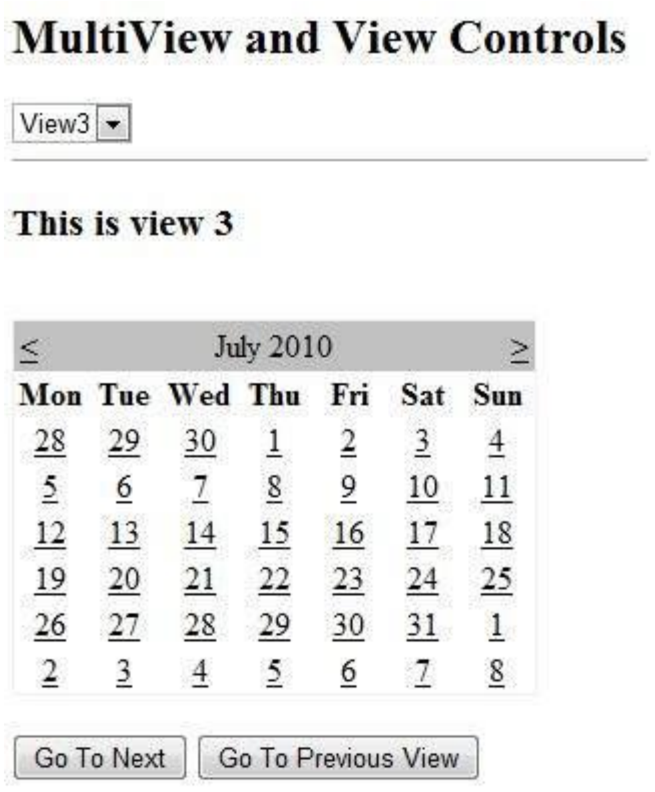
Apart from the above mentioned properties, methods and events, multiview control inherits the members of the control and object class.

Example

The example page has three views. Each view has two button for navigating through the views.

The content file code is as follows:

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

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

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

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

      <div>
        <h2>MultiView and View Controls</h2>
        <asp:DropDownList ID="DropDownList1" runat="server" 
onselectedindexchanged="DropDownList1_SelectedIndexChanged">
        </asp:DropDownList>

        <hr />

        <asp:MultiView ID="MultiView1" runat="server" ActiveViewIndex="2" 
onactiveviewchanged="MultiView1_ActiveViewChanged" >
          <asp:View ID="View1" runat="server">
            <h3>This is view 1</h3>
            <br />
            <asp:Button CommandName="NextView" ID="btnnext1" runat="server" Text = "Go To 
Next" />
            <asp:Button CommandArgument="View3" CommandName="SwitchViewByID" 
ID="btnlast" runat="server" Text  ="Go To Last" />
          </asp:View>

          <asp:View ID="View2" runat="server">
            <h3>This is view 2</h3>
            <asp:Button CommandName="NextView" ID="btnnext2" runat="server" Text = "Go To 
Next" />
            <asp:Button CommandName="PrevView" ID="btnprevious2" runat="server" Text = 
"Go To Previous View" />
          </asp:View>

          <asp:View ID="View3" runat="server">
            <h3> This is view 3</h3>
            <br />
            <asp:Calendar ID="Calender1" runat="server"></asp:Calendar>
            <br />
            <asp:Button  CommandArgument="0" CommandName="SwitchViewByIndex" 
ID="btnfirst"   runat="server" Text = "Go To Next" />
            <asp:Button CommandName="PrevView" ID="btnprevious" runat="server" Text = "Go 
To Previous View" />
          </asp:View>

        </asp:MultiView>
      </div>

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


MultiView and View Controls

View3

This is view 3

<			July 2010			>
Mon	Tue	Wed	Thu	Fri	Sat	Sun
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Wizard Control

This control is same as MultiView control but the main difference is that, it has inbuilt navigation buttons.

The wizard control enables you to design a long form in such a way that you can work in multiple sub form. You can perform the task in a step by step process. It reduces the work of developers to design multiple forms. It enables you to create multi step user interface. Wizard control provides with built-in previous/next functionality.

The Wizard control can contains one or more WizardStep as child controls. Only one WizardStep is displayed at a time. WizardStep control has an important property called as StepType. The StepType property determines the type of navigation buttons that will be displayed for that step. The possible values are:

The StepType associated with each WizardStep determines the type of navigation buttons that will be displayed for that step. **The StepTypes are:**

- Start:
- Step:
- Finish:
- Complete:
- Auto:

Drag the Wizard control on the web page from toolbox, you will get the following code.

```
<asp:Wizard ID="Wizard1" runat="server" Height="75px" Width="140px">
        <WizardSteps>
              <asp:WizardStep runat="server" title="Step 1">
              </asp:WizardStep>
              <asp:WizardStep runat="server" title="Step 2">
              </asp:WizardStep>
        </WizardSteps>
</asp:Wizard>
```

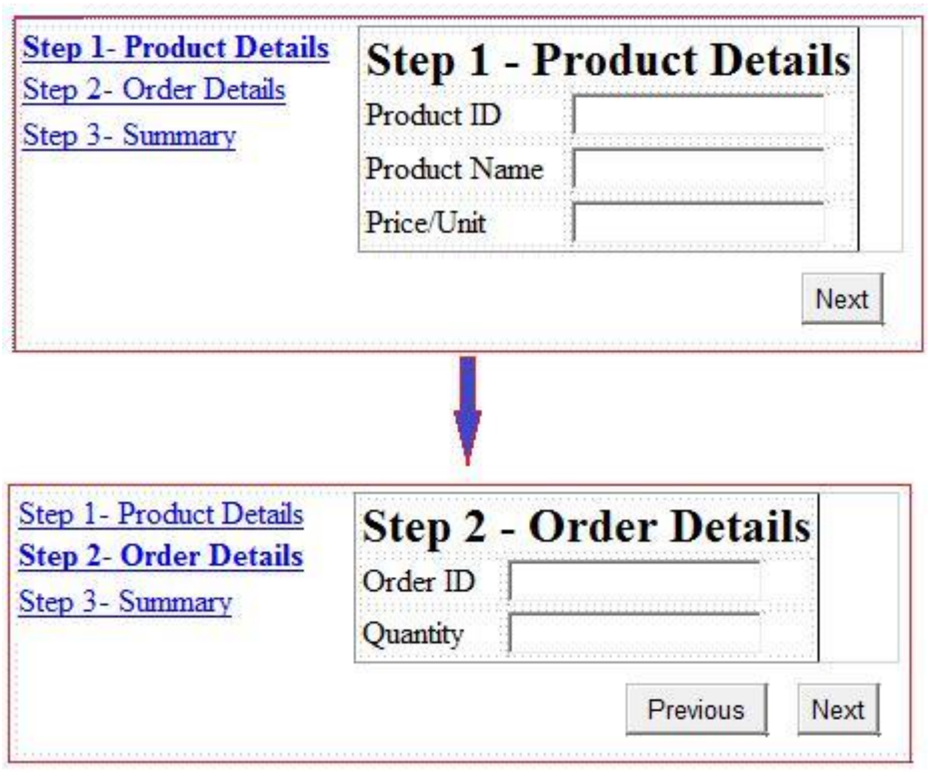
You can put WizardStep according to application need.

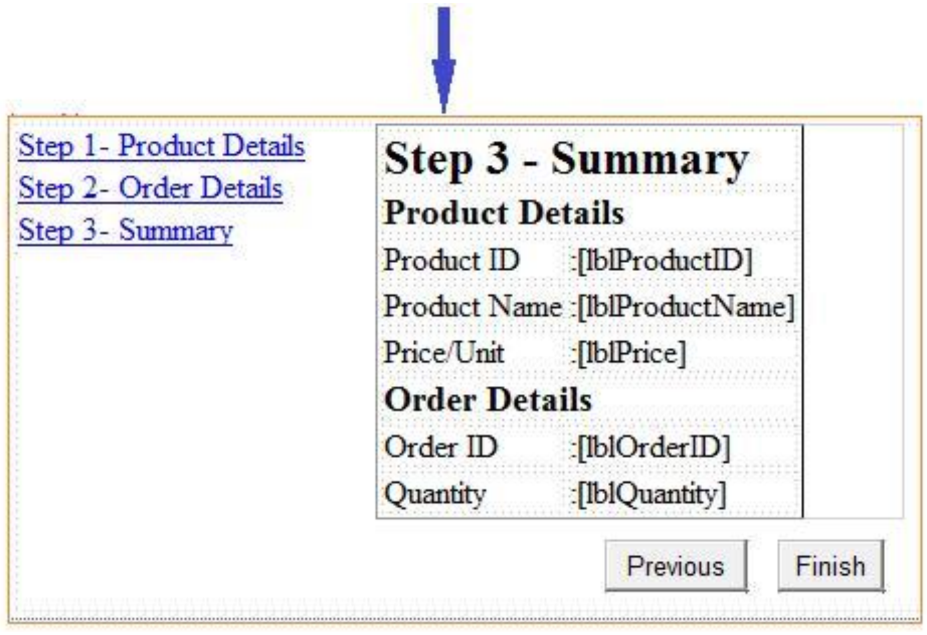
Important events of Wizard control are as follows:

- ActiveStepChanged:
- CancelButtonClick:
- FinishButtonClick:
- NextButtonClick:
- PreviousButtonClick:

Now we will create an application as we had done with MultiView control. We will create three different WizardStep in Wizard control.

1. In First step we will design to capture Product details
2. In Second step we will design to capture Order details
3. Next we will show summary for confirmation.





Step 1- Product Details
Step 2- Order Details
Step 3- Summary

Step 3 - Summary

Product Details

Product ID	:[lblProductID]
Product Name	:[lblProductName]
Price/Unit	:[lblPrice]

Order Details

Order ID	:[lblOrderID]
Quantity	:[lblQuantity]

WizardControlDemo.aspx.cs file

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

public partial class WizardControl : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
   }
   protected void Wizard1_FinishButtonClick(object sender, WizardNavigationEventArgs e)
   {
      Response.Redirect("SaveData.aspx");
   }
   protected void Wizard1_NextButtonClick(object sender, WizardNavigationEventArgs e)
   {
      if (e.NextStepIndex == 2)
      {
         lblProductID.Text = txtProductID.Text;
         lblProductName.Text = txtProductName.Text;
         lblPrice.Text = txtProductPrice.Text;
         lblOrderID.Text = txtOrderID.Text;
         lblQuantity.Text = txtQuantity.Text;
      }
   }
}
```

Register the event for Next button by using property window with event tab. In the given example, for going third step from second we have to set e.NextStepIndex == 2. Here StepIndex is zero based.

FinishButtonClick event performs the final WizardStep with a summary of the answers entered in the previous WizardStep controls.

ASP.NET folder structure

The asp.net application folder is contains list of specified folder that you can use of specific type of files or content in an each folder. The root folder structure is as following

- BIN
- App_Code
- App_GlobalResources
- App_LocalResources
- App_WebReferences
- App_Data
- App_Browsers
- App_Themes

Bin Directory

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

App_Code Directory

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

App_GlobalResources Directory

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

App_LocalResources Directory

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

App_WebReferences Directory

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

App_Data Directory

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

App_Browsers Directory

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

App_Themes Directory

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

Comparison between HTML controls and server controls

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.

Enter Name :
 	
`<input type="text"  ID="txtName">`
	Enter Name :
	
	`<asp:TextBox Id="txtName" runat="server">`
	
	`</asp:TextBox>`

ASP.NET - Data Sources

A data source control interacts with the data-bound controls and hides the complex data binding processes. These are the tools that provide data to the data bound controls and support execution of operations like insertions, deletions, sorting, and updates.

Each data source control wraps a particular data provider-relational databases, XML documents, or custom classes and helps in:

- Managing connection
- Selecting data
- Managing presentation aspects like paging, caching, etc.
- Manipulating data

There are many data source controls available in ASP.NET for accessing data from SQL Server, from ODBC or OLE DB servers, from XML files, and from business objects.

Based on type of data, these controls could be divided into two categories:

- Hierarchical data source controls
- Table-based data source controls

The data source controls used for hierarchical data are:

- **XMLDataSource** - It allows binding to XML files and strings with or without schema information.
- **SiteMapDataSource** - It allows binding to a provider that supplies site map information.

The data source controls used for tabular data are:

Data source controls	Description
SqlDataSource	It represents a connection to an ADO.NET data provider that returns SQL data, including data sources accessible via OLEDB and ODBC.
ObjectDataSource	It allows binding to a custom .Net business object that returns data.
LinqdataSource	It allows binding to the results of a Linq-to-SQL query (supported by ASP.NET 3.5 only).
AccessDataSource	It represents connection to a Microsoft Access database.

The SqlDataSource Control

The SqlDataSource control represents a connection to a relational database such as SQL Server or Oracle database, or data accessible through OLEDB or Open Database Connectivity (ODBC). Connection to data is made through two important properties ConnectionString and ProviderName.

The following code snippet provides the basic syntax of the control:

```
<asp:SqlDataSource runat="server" ID="MySqlSource"
   ProviderName='<%$ ConnectionStrings:LocalNWind.ProviderName  %>'
   ConnectionString='<%$ ConnectionStrings:LocalNWind %>'
   SelectionCommand= "SELECT * FROM EMPLOYEES" />
<asp:GridView ID="GridView1" runat="server" DataSourceID="MySqlSource" />
```

Configuring various data operations on the underlying data depends upon the various properties (property groups) of the data source control.

The following table provides the related sets of properties of the SqlDataSource control, which provides the programming interface of the control:

Property Group	Description
DeleteCommand, DeleteParameters, DeleteCommandType	Gets or sets the SQL statement, parameters, and type for deleting rows in the underlying data.
FilterExpression, FilterParameters	Gets or sets the data filtering string and parameters.
InsertCommand, InsertParameters, InsertCommandType	Gets or sets the SQL statement, parameters, and type for inserting rows in the underlying database.
SelectCommand, SelectParameters, SelectCommandType	Gets or sets the SQL statement, parameters, and type for retrieving rows from the underlying database.
SortParameterName	Gets or sets the name of an input parameter that the command's stored procedure will use to sort data.

UpdateCommand, UpdateParameters, UpdateCommandType	Gets or sets the SQL statement, parameters, and type for updating rows in the underlying data store.

The following code snippet shows a data source control enabled for data manipulation:

```
<asp:SqlDataSource runat="server" ID= "MySqlSource"
   ProviderName='<%$ ConnectionStrings:LocalNWind.ProviderName  %>'
   ConnectionString=' <%$ ConnectionStrings:LocalNWind %>'
   SelectCommand= "SELECT * FROM EMPLOYEES"
   UpdateCommand= "UPDATE EMPLOYEES SET LASTNAME=@lame"
   DeleteCommand= "DELETE FROM EMPLOYEES WHERE EMPLOYEEID=@eid"
   FilterExpression= "EMPLOYEEID > 10">
   .....
   .....
</asp:SqlDataSource>
```

The ObjectDataSource Control

The ObjectDataSource Control enables user-defined classes to associate the output of their methods to data bound controls. The programming interface of this class is almost same as the SqlDataSource control.

Following are two important aspects of binding business objects:

- The bindable class should have a default constructor, it should be stateless, and have methods that can be mapped to select, update, insert, and delete semantics.
- The object must update one item at a time, batch operations are not supported.

Let us go directly to an example to work with this control. The student class is the class to be used with an object data source. This class has three properties: a student id, name, and city. It has a default constructor and a GetStudents method for retrieving data.

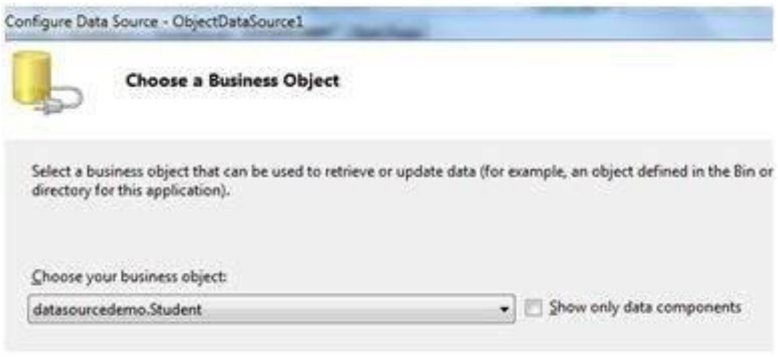
The student class:

```
public class Student
{
   public int StudentID { get; set; }
   public string Name { get; set; }
   public string City { get; set; }

   public Student()
   { }

   public DataSet GetStudents()
```

```
  {
    DataSet ds = new DataSet();
    DataTable dt = new DataTable("Students");

    dt.Columns.Add("StudentID", typeof(System.Int32));
    dt.Columns.Add("StudentName", typeof(System.String));
    dt.Columns.Add("StudentCity", typeof(System.String));
    dt.Rows.Add(new object[] { 1, "M. H. Kabir", "Calcutta" });
    dt.Rows.Add(new object[] { 2, "Ayan J. Sarkar", "Calcutta" });
    ds.Tables.Add(dt);

    return ds;
  }
}
```

Take the following steps to bind the object with an object data source and retrieve data:

- Create a new web site.
- Add a class (Students.cs) to it by right clicking the project from the Solution Explorer, adding a class template, and placing the above code in it.
- Build the solution so that the application can use the reference to the class.
- Place an object data source control in the web form.
- Configure the data source by selecting the object.

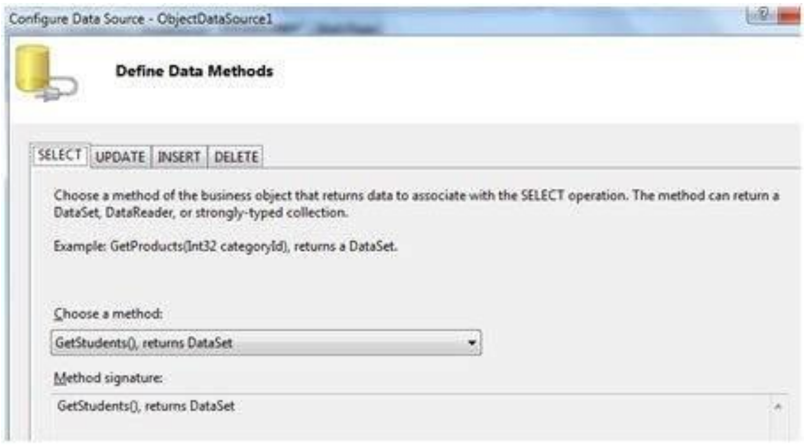

Configure Data Source - ObjectDataSource1

Choose a Business Object

Select a business object that can be used to retrieve or update data (for example, an object defined in the Bin or directory for this application).

Choose your business object:
datasourcedemo.Student

- Select a data method(s) for different operations on data. In this example, there is only one method.



Configure Data Source - ObjectDataSource1

Define Data Methods

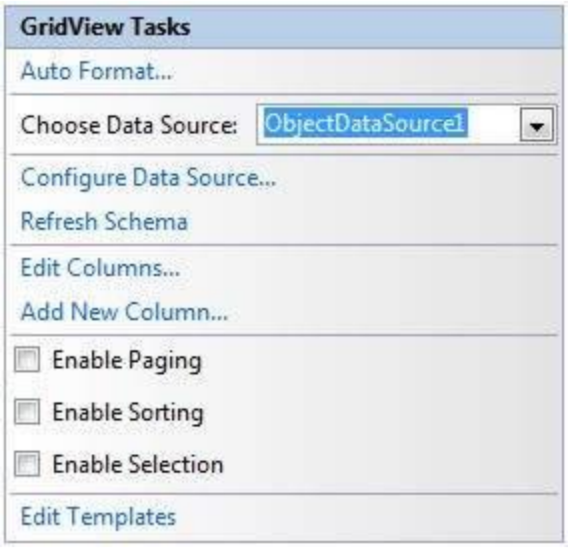
Choose a method of the business object that returns data to associate with the SELECT operation. The method can return a DataSet, DataReader, or strongly-typed collection.

Example: GetProducts(Int32 categoryId), returns a DataSet.

Choose a method: GetStudents(), returns DataSet

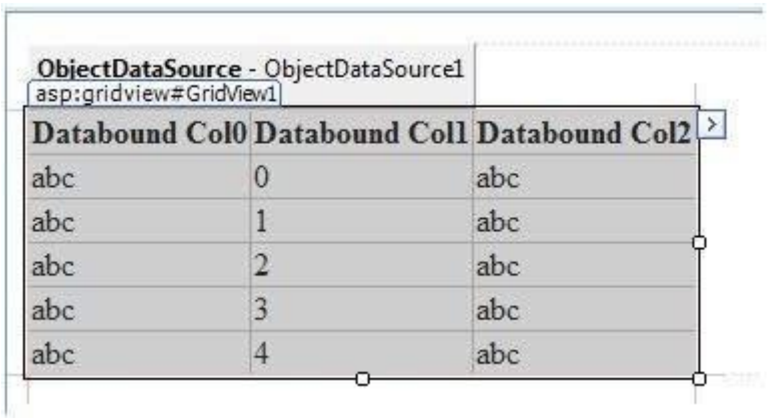
Method signature: GetStudents(), returns DataSet

- Place a data bound control such as grid view on the page and select the object data source as its underlying data source.



Choose Data Source: ObjectDataSource1

- At this stage, the design view should look like the following:

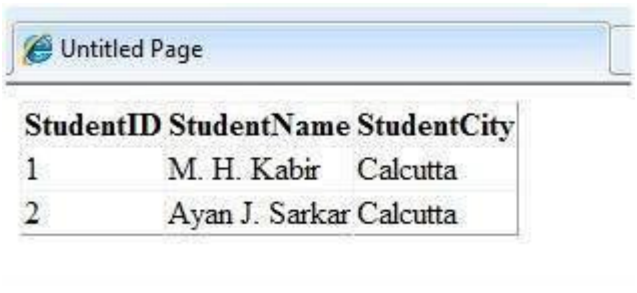


ObjectDataSource - ObjectDataSource1

asp:gridview#GridView1

Databound Col0	Databound Col1	Databound Col2
abc	0	abc
abc	1	abc
abc	2	abc
abc	3	abc
abc	4	abc

- Run the project, it retrieves the hard coded tuples from the students class.



StudentID	StudentName	StudentCity
1	M. H. Kabir	Calcutta
2	Ayan J. Sarkar	Calcutta

The AccessDataSource Control

The AccessDataSource control represents a connection to an Access database. It is based on the SqlDataSource control and provides simpler programming interface. The following code snippet provides the basic syntax for the data source:

```
<asp:AccessDataSource ID="AccessDataSource1 runat="server"
   DataFile="~/App_Data/ASPDotNetStepByStep.mdb" SelectCommand="SELECT * FROM
[DotNetReferences]">
</asp:AccessDataSource>
```

The AccessDataSource control opens the database in read-only mode. However, it can also be used for performing insert, update, or delete operations. This is done using the ADO.NET commands and parameter collection.

Updates are problematic for Access databases from within an ASP.NET application because an Access database is a plain file and the default account of the ASP.NET application might not have the permission to write to the database file.

Introduction to Web Site Navigation

A web site has many web pages which provide information to the user. Users have to navigate across web pages easily and quickly. Navigation in a web site is essential and important. It provides a means for allowing the user to access the required information quickly in the web application. Hyperlinks act as a basic mechanism of web site navigation. The Hyperlinks act as static links. The navigation mechanisms used may be simple or complex, depending on the web site.

The Navigation Controls

ASP.NET provides a set of advanced navigational features with the help of Navigation Controls. The ASP.NET navigational controls provide rich and consistent navigational functionality. The ASP.NET navigational controls are SiteMapPath, TreeView and Menu.These controls can be used to provide navigation to the web application, without coding or little coding. However, you can also write code for these controls to manage them at run-time.

The Site Map

The Site Map provides a logical structure of the pages in a website. The Site Map information is stored in a file named Web.sitemap. The Web.sitemap file is stored in the root directory of the web application. ASP.NET has the XmlSiteMapProvider which is used to access the information from the Web.sitemap file. It automatically reads data in the Web.sitemap file present in the root folder of the web application and stores it in a SiteMap object. Let us now consider the banking scenario where Banking can be of two types, namely Personal Banking and Corporate Banking.

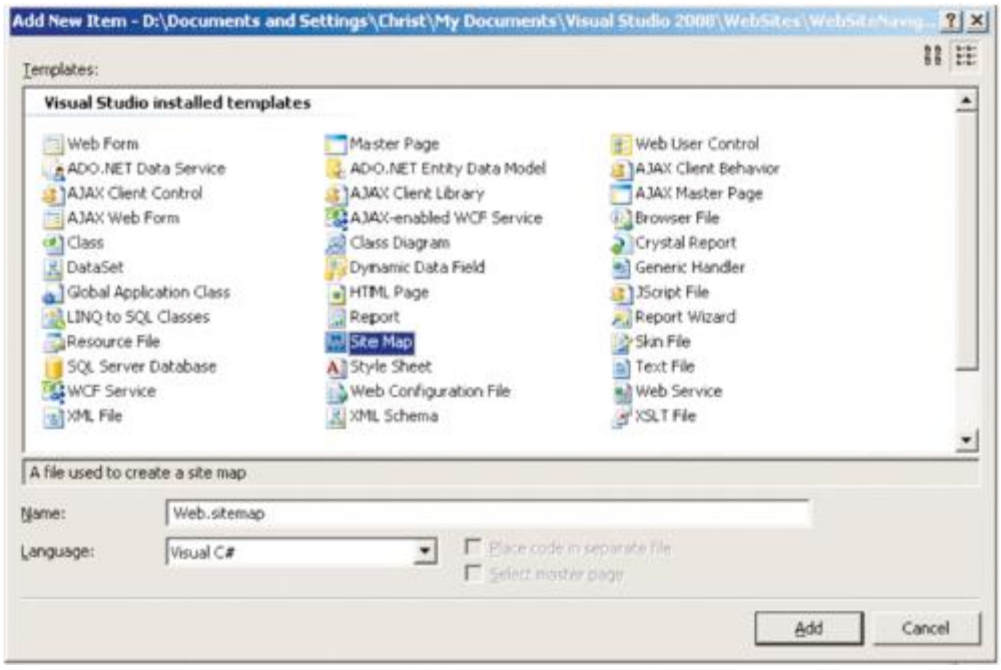
Further more personal banking has Accounts, Loans and Insurance.Similarly, Corporate Banking has Credit and Capital Market. The Web.sitemap file can be created with the required information.

Demonstration: Creating the Web.sitemap file

Steps to create the Web.sitemap file.

1) Create a New Web Site named “WebSiteNavigation”.

2) Right-click the project in the Solution Explorer and Select Add New Item and select Site Map as shown below and click the Add button.



Add New Item - D:\Documents and Settings\Christ\My Documents\Visual Studio 2008\WebSites\WebSiteNavig...

A file used to create a site map

Name: Web.sitemap

Language: Visual C#

3) Enter the following content below in the Web.sitemap file.

```
< ?xml version="1.0" encoding="utf-8" ? >
< siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
< siteMapNode url="" title="Banking" description="Banking" >
< siteMapNode url="" title="Personal Banking" description="Personal
Banking Information" >
< siteMapNode url="Accounts.aspx" title="Accounts"
description="Accounts Information" / >
< siteMapNode url="Loans.aspx" title="Loans" description="Loans
Information" / >
< siteMapNode url="Insurance.aspx" title="Insurance"
description="Insurance Information" / >
< /siteMapNode >

< siteMapNode url="Credit.aspx" title="Credit" description="Credit
Information" > < /siteMapNode >
< siteMapNode url="CapitalMarket.aspx" title="Capital Market"
description="Capital Market Information" > < /siteMapNode >
< /siteMapNode >
< /siteMapNode >
< /siteMap >
```

Exploring the Web.sitemap file:

The Web.sitemap file contains < siteMap > as the root element. The < siteMap > contains a child element named < siteMapNode >. The < siteMap > element can have only one < siteMapNode > element inside it. The < siteMapNode > element inside the < siteMap > element can contain any number of < siteMapNode > elements. The < siteMapNode > contains many properties such as title, description and url etc. The value set to the title property of the < siteMapNode > is displayed as text or links by controls such as the tree view. The value set to the description property is displayed as the Tooltip text when the mouse is placed over the node. The url property stores the url to navigate when the user clicks the node.

Note : The values set to URL property in the SiteMap should be unique.

The SiteMapDataSource Control

The SiteMapDataSource control obtains the site map information from the SiteMap object.The Navigation controls can display the data from the SiteMapDataSource control. For example, you can use a tree view control to display the information stored in the Web.sitemap file by using the SiteMapDataSource control. It is easy to use the SiteMapDataSource control.The SiteMapDataSource control can be created by dragging the control available in the Data tab of the Toolbox into the Design View of the Web Form. It can also be created by entering the code below in the Source View of the Web form.

```
< asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" / >
```

The TreeView Control

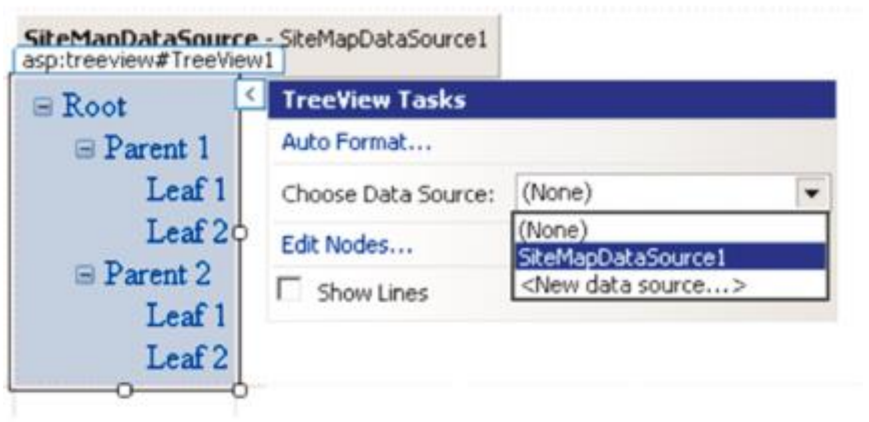
The TreeView control can be used to display a complex hierarchical structure of items. TreeView can be used fo r purposes, such as to display the Windows File System Directory Structure, or any structure with a hierarchy. The TreeView can be expanded to many levels.The TreeView is an advanced server control. It can display the details contained in the Web.sitemap file by using the SiteMapDataSource control. The DataSourceID property of the TreeView control is set to the ID of the SiteMapDataSource. Below is the tag for TreeView Control.

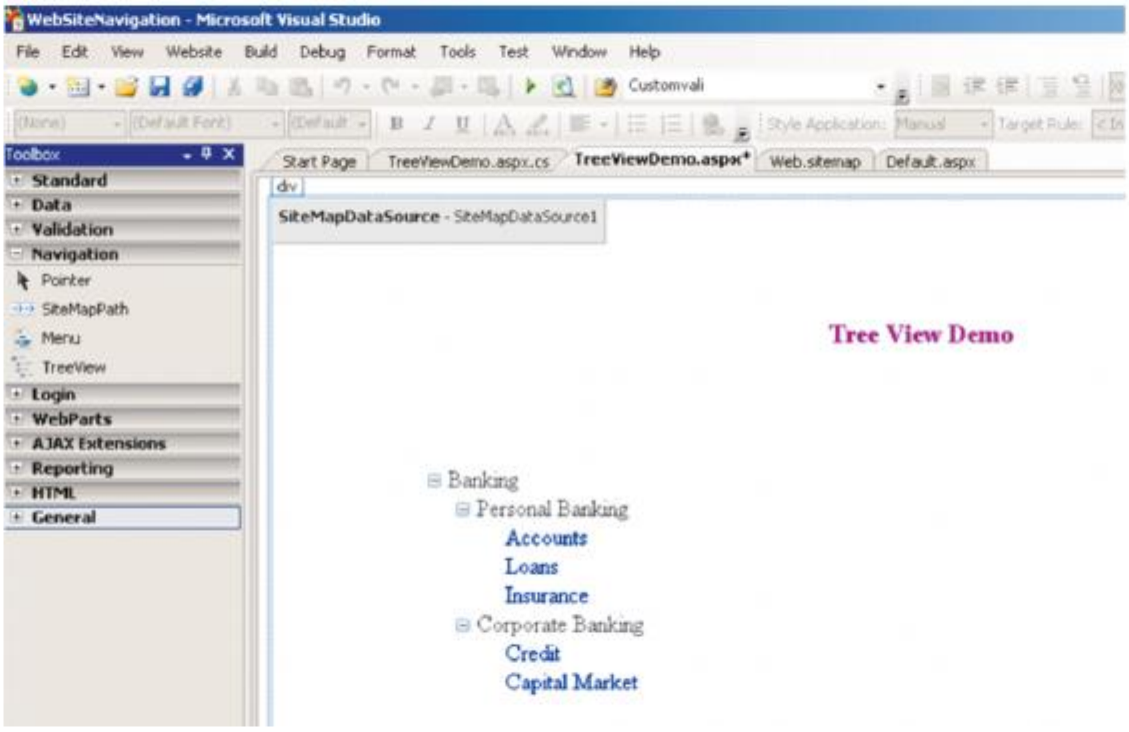
```
< asp:TreeView
ID="TreeView1"
runat="server"
DataSourceID="SiteMapDataSource1" >
< /asp:TreeView >
```

Demonstration: Creating a TreeView Control to display the information stored in the Web.sitemap file using the SiteMapDataSource Control.

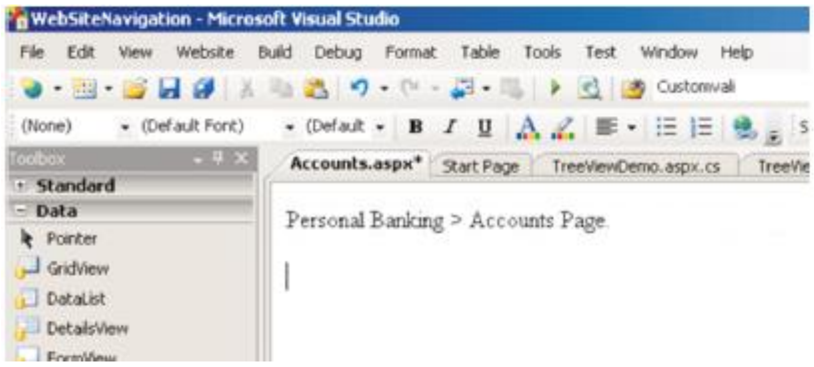
Steps to create the Web Page.
1) Create a new web form named TreeViewDemo.aspx in the WebSiteNavigation Project.
2) Add a Label and set its Text as “Tree View Demo”
3) Drag a SiteMapDataSource Control from the Data Tab of the Toolbox into the Web Form.
4) Drag a TreeView Control from the Navigation Tab of the Toolbox into the Web Form.

5) Click on the Smart Tag of the TreeView Control and in the Choose Data Source Drop Down, select SiteMapDataSource1.

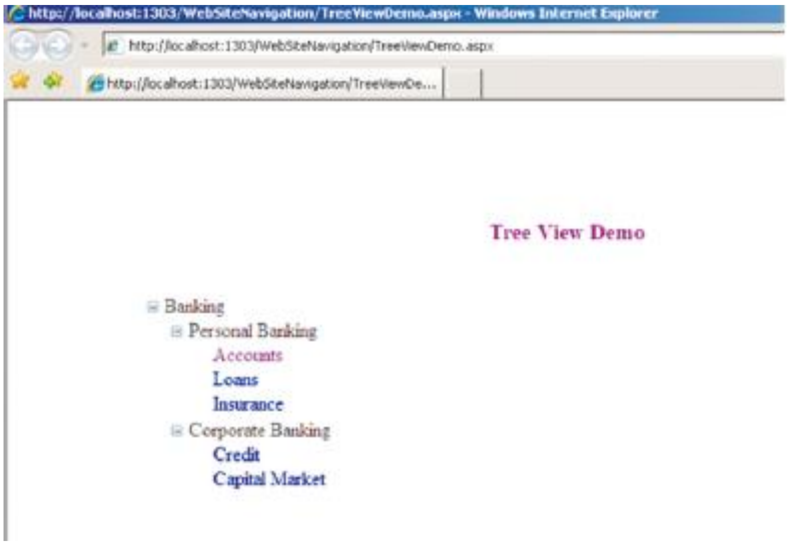




7) Create a New Web Form named Accounts.aspx and enter “Personal Banking > Accounts Page” inside the page. The Accounts page design should look as below:


WebSiteNavigation - Microsoft Visual Studio
File Edit View Website Build Debug Format Table Tools Test Window Help
Toolbox
Standard
Data
Pointer
GridView
DataList
DetailsView
Accounts.aspx*
Start Page
TreeViewDemo.aspx.cs
Personal Banking > Accounts Page

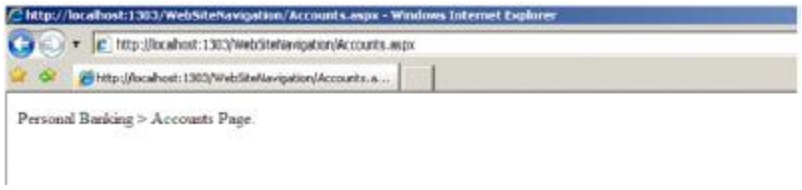
8) Right-Click the TreeViewDemo.aspx file and select “Set As Start Page” and execute the project.
9) View the Output 1 below :


http://localhost:1303/WebSiteNavigation/TreeViewDemo.aspx - Windows Internet Explorer
http://localhost:1303/WebSiteNavigation/TreeViewDemo.aspx

Tree View Demo

Banking
Personal Banking
Accounts
Loans
Insurance
Corporate Banking
Credit
Capital Market

10) Click on the Accounts Link under Banking > Personal Banking.
11) View the Output 2 below :


http://localhost:1303/WebSiteNavigation/Accounts.aspx - Windows Internet Explorer
http://localhost:1303/WebSiteNavigation/Accounts.aspx

Personal Banking > Accounts Page

12) Similarly create web pages for Loans, Insurance, Credit and Capital Market.

The Menu Control

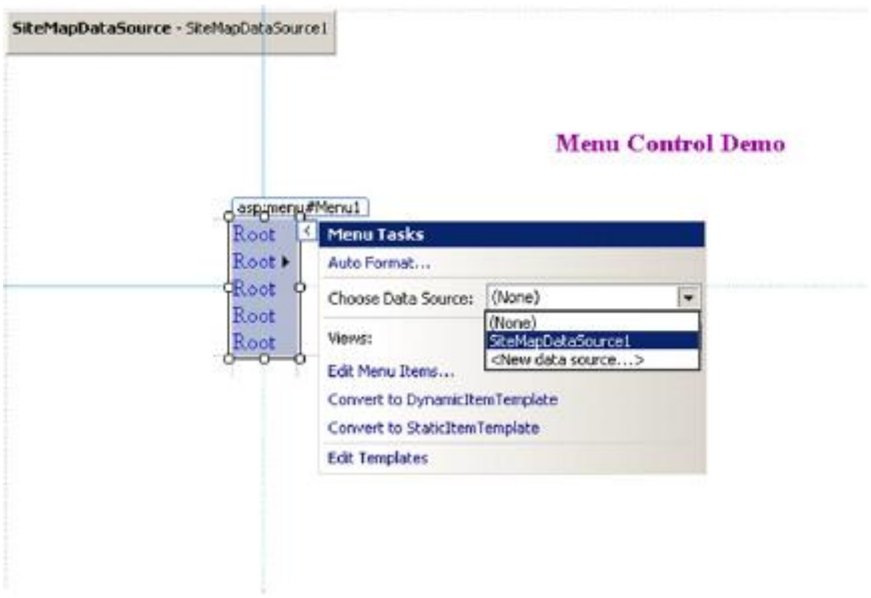
The Menu Control can be used to display hierarchical data. It is very easy to navigate through the Menu Control when there are many options in the hierarchy. Menus occupy less space in the web page. Menus can be expanded at various levels or collapsed as required by the user.The Menu control can be bound to a SiteMapDataSource to display the information available in the Web.sitemap file. The DataSourceID property of the Menu Control is set to the ID of the SiteMapDataSource control. Below is the tag for the Menu Control.

```
< asp:Menu ID="Menu1"
runat="server"
DataSourceID="SiteMapDataSource1" >
< /asp:Menu >
```

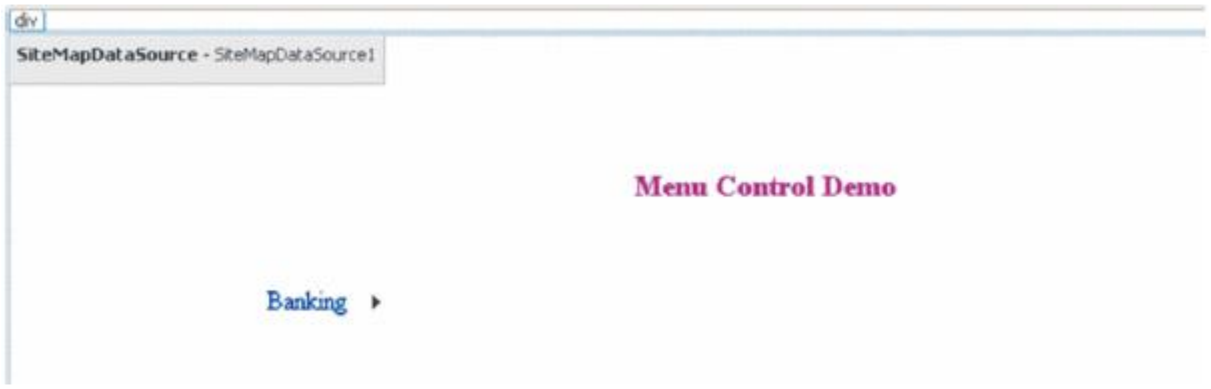
Demonstration: Creating a Menu Control to display the information stored in the Web.sitemap file using the SiteMapDataSource Control.

Steps to create the Web Page.
1) Create a new web form named MenuControlDemo.aspx in the WebSiteNavigation Project.
2) Add a Label and set its Text as “Menu Control Demo”
3) Drag a SiteMapDataSource Control from the Data Tab of the Toolbox into the Web Form.

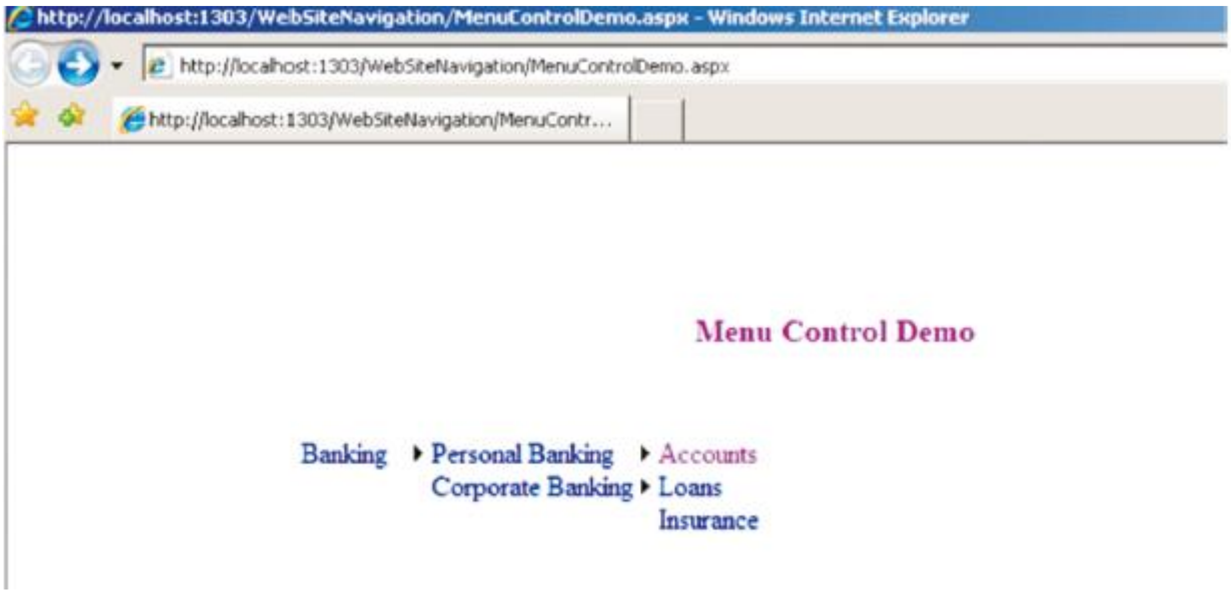
4) Drag a Menu Control from the Navigation Tab of the Toolbox into the Web Form.
5) Click on the Smart Tag of the Menu Control and in the Choose Data Source Drop Down, select SiteMapDataSource1.



6) The design should look as below:



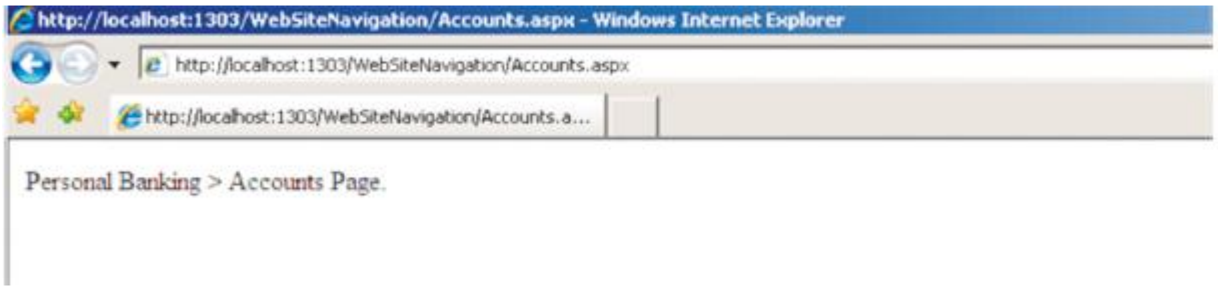
7) Right-Click the MenuControlDemo.aspx file and select “Set As Start Page” and execute the project.
8) View the Output 1 below :



http://localhost:1303/WebSiteNavigation/MenuControlDemo.aspx - Windows Internet Explorer

Menu Control Demo

9) Click on the Accounts Link under Banking > Personal Banking.
10) View the Output 2 below :



http://localhost:1303/WebSiteNavigation/Accounts.aspx - Windows Internet Explorer

Personal Banking > Accounts Page.

11) Similarly you can navigate to the other web pages by clicking the Loans, Insurance, Credit and Capital Market links.

The ASP.NET login controls

The ASP.NET login controls provide a robust login solution for ASP.NET Web applications without requiring programming. By default, login controls integrate with ASP.NET membership and forms authentication to help automate user authentication for a Web site. It provides you with a ready-to-use user interface that queries the user name and password from the user and offers a Log In button for login. It validate user credentials against the membership API and encapsulating the basic froms authentication functionality like redirecting back to the original requested page in a restricted area of you application after the successful login.

The Login control displays a user interface for user authentication. The Login control contains text boxes for the user name and password and a check box that allows users to indicate whether they want the server to store their identity using ASP.NET membership and automatically be authenticated the next time they visit the site.

The Login control has properties for customized display, for customized messages, and for links to other pages where users can change their password or recover a forgotten password. The Login control can be used as a standalone control on a main or home page, or you can use it on a dedicated login page. If you use the Login control with ASP.NET membership, you do not need to write code to perform authentication. However, if you want to create your own authentication logic, you can handle the Login control's Authenticate event and add custom authentication code.

These controls include the Login, LoginView, PasswordRecovery, LoginStatus, LoginName, ChangePassword, and CreateUserWizard controls. Here's a summary of what each control does.

Login-The Login control is the simplest login control and supports the most common login scenario-signing in using a user name and password. The control includes user name and password text boxes and a check box for users who want to compromise password security by saving their passwords on the machine. The control exposes properties through which you can change the text and appearance of the control. You may also add links to manage registration or password recovery. The Login control interacts with the ASP.NET membership component for authentication by default. If you want to manage authentication yourself, you may do so by handling the control's Authenticate event.

LoginView-The LoginView control is very like the optional login page mentioned earlier. It's useful for managing the content you display for authenticated versus nonauthenticated users. The LoginView displays the login status via the display templates AnonymousTemplate and LoggedInTemplate. The control renders a different template depending on the status of the user. The LoginView also lets you manage text and links within each template.

PasswordRecovery-The PasswordRecovery control supports Web sites that send user passwords to clients when they forget their passwords. The control collects the user's account name, and then follows up with a security question (provided that functionality is set up correctly). The control either e-mails the current password to the user or creates a new one.

LoginStatus-The LoginStatus control displays whether or not the current user is logged on. Nonlogged-in users are prompted to log in, while logged-in users are prompted to log out.

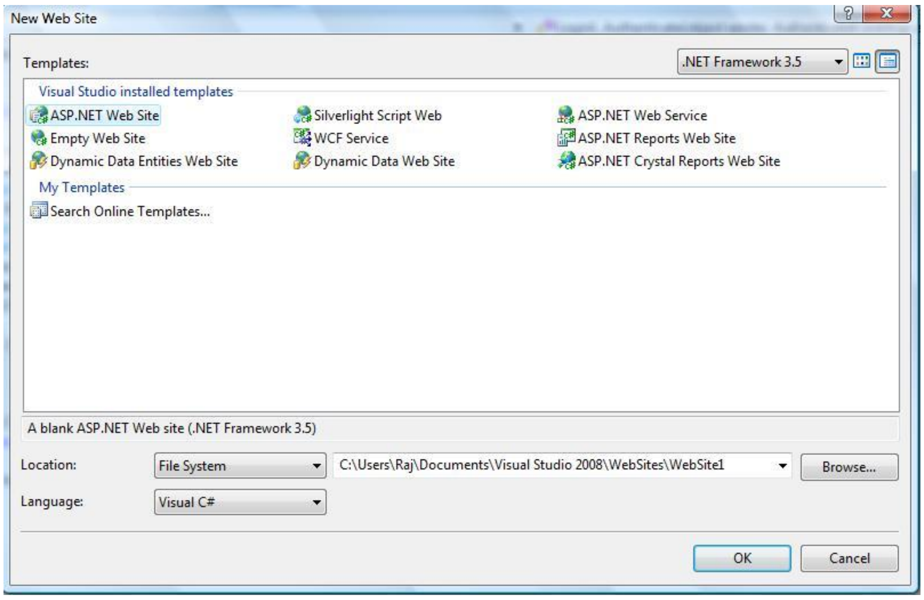
LoginName-The LoginName control displays the user's login name.

ChangePassword-The ChangePassword control gives users a chance to change their passwords. An authenticated user may change his or her password by supplying the original password and a new password (along with a confirmation of the new password).

CreateUserWizard-The CreateUserWizard control collects information from users so it can set up an ASP.NET membership account for each user. Out of the box, the control gathers a user name, a password, an e-mail address, a security question, and a security answer. The CreateUserWizard will collect different information from users, depending on the membership provider used by your application.

Note - Login controls might not function correctly if the Method of the ASP.NET Web page is changed from POST (the default) to GET.

1. Start Microsoft Visual Studio 2008
2. Create a new ASP.NET WebSite, Like this:


New Web Site

Templates: .NET Framework 3.5

Visual Studio installed templates

- ASP.NET Web Site
- Empty Web Site
- Dynamic Data Entities Web Site
- Silverlight Script Web
- WCF Service
- Dynamic Data Web Site
- ASP.NET Web Service
- ASP.NET Reports Web Site
- ASP.NET Crystal Reports Web Site

My Templates

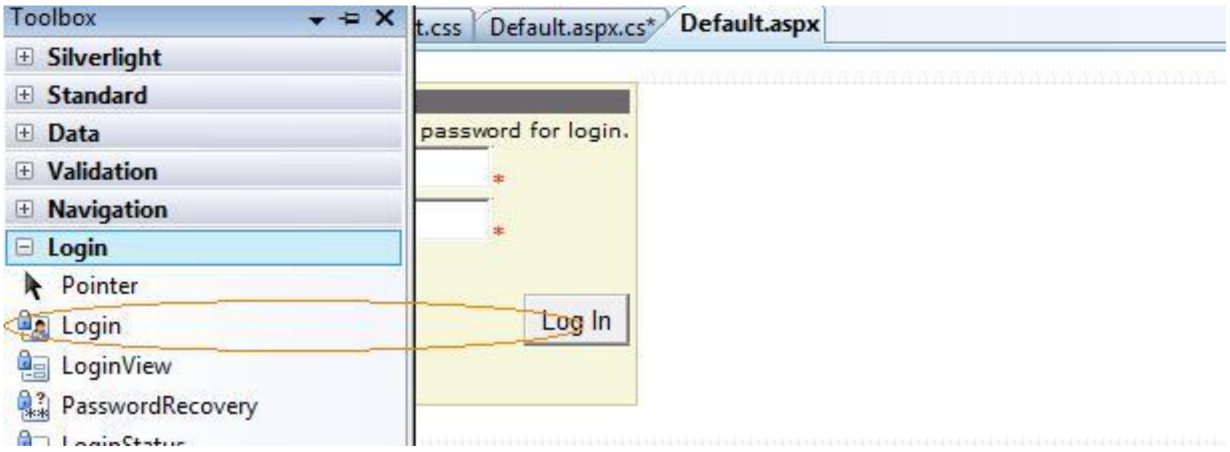
- Search Online Templates...

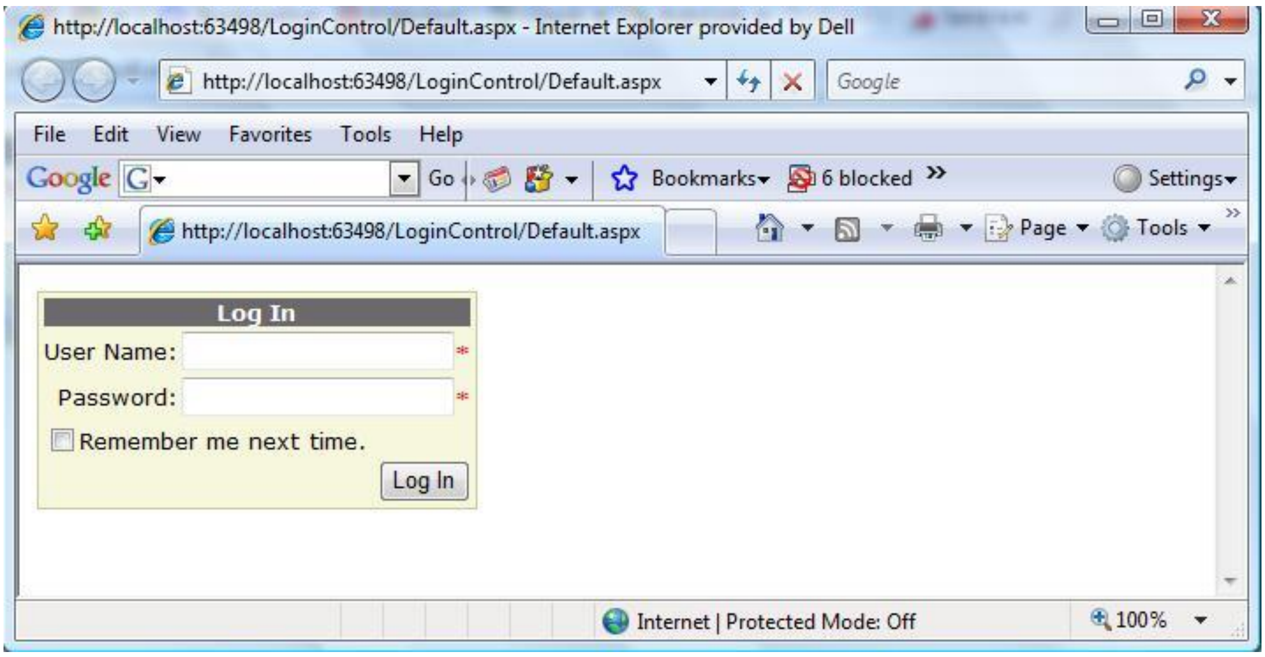
A blank ASP.NET Web site (.NET Framework 3.5)

Location: File System C:\Users\Raj\Documents\Visual Studio 2008\WebSites\WebSite1

Language: Visual C#

3. Drag and drop Login control on page from ToolBox.




http://localhost:63498/LoginControl/Default.aspx - Internet Explorer provided by Dell
File Edit View Favorites Tools Help
Internet | Protected Mode: Off
100%

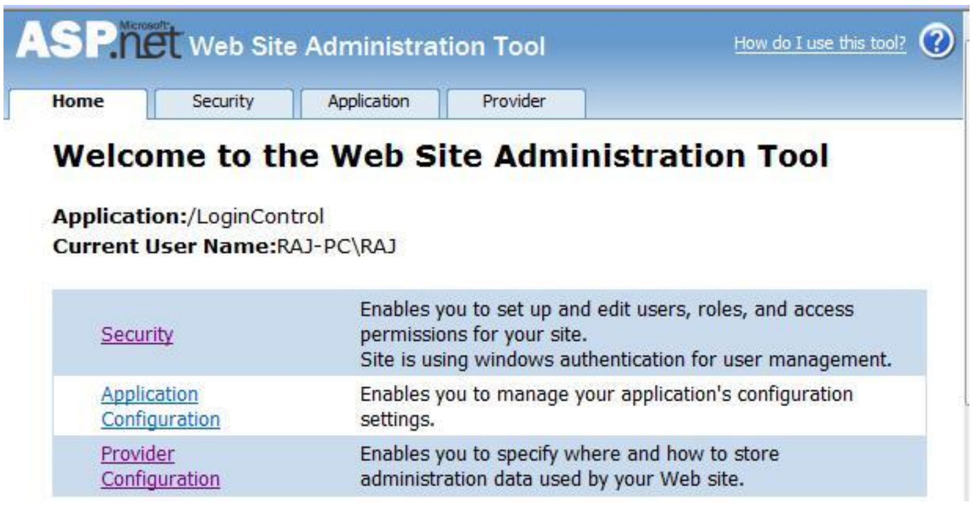
Log In
User Name: *
Password: *
Remember me next time.
Log In

Figure 1.

Whenever user hits the Log In button, the control automatically validates the user name and password using the membership API function Membership.ValidateUse() and then calls FormAuthentication.redirectFromLoginPage() if the validation was successful. All options on the UI of the LoginControl affect the input delivered by the control to these methods. For Example, if you click the "Remember me next time" check box, it passes the value true to the createPresistentCookie parameter of the RedirectFromLoginPage() method. Therefore, the FormAuthenticateModule creates a persistent cookie.

There are three Login Tasks by default.

- Auto Format - you can select default schemes.
- Convert To Template - You can edit content of Login Control.
- Administer Website - You can configure Web Site Administration Tools, Like Security, Application, Provider.



Welcome to the Web Site Administration Tool

Application:/LoginControl
Current User Name:RAJ-PC\RAJ

Security	Enables you to set up and edit users, roles, and access permissions for your site. Site is using windows authentication for user management.
Application Configuration	Enables you to manage your application's configuration settings.
Provider Configuration	Enables you to specify where and how to store administration data used by your Web site.

Figure 2.

```
<form id="form1" runat="server">
<div>
<asp:Login ID="Login1" runat="server" BackColor="#F7F7DE" BorderColor="#CCCC99"
BorderStyle="Solid" BorderWidth="1px" Font-Names="Verdana" Font-Size="10pt">
<TitleTextStyle BackColor="#6B696B" Font-Bold="True" ForeColor="#FFFFFF" />
</asp:Login>
</div>
</form>
```

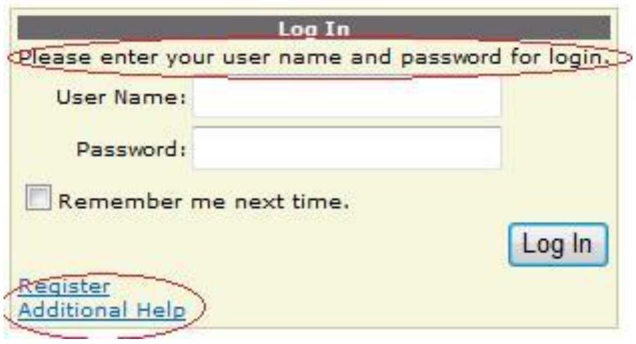
And now apply css to control:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>Login Control</title>
   <link href="StyleSheet.css" type="text/css" rel="Stylesheet" />
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <asp:Login ID="Login1" runat="server" CssClass="LoginControl">
         <TitleTextStyle BackColor="#6B696B" Font-Bold="True" ForeColor="#FFFFFF" />
      </asp:Login>
   </div>
   </form>
</body>
</html>
```

You can add several hyperlinks to your Login control, such as hyperlink to a help text page, or a hyperlink to to a registration page.

```
<asp:Login ID="Login1" runat="server" CssClass="LoginControl"
CreateUserText="Register"
CreateUserUrl="~/Register.aspx"
HelpPageText="Additional Help" HelpPageUrl="~/Help.aspx"
InstructionText="Please enter your user name and password for login.">
<TitleTextStyle BackColor="#6B696B" Font-Bold="True" ForeColor="#FFFFFF" />
</asp:Login>
```

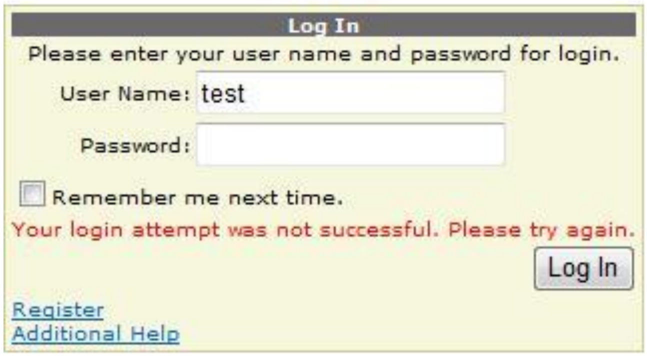
Looks like this :



Here is .CS Code: using System;

```
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data.SqlClient;
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!this.IsPostBack)
            ViewState["LoginErrors"] = 0;
    }
   protected void Login1_Authenticate(object sender, AuthenticateEventArgs e)
    {
        if (YourValidationFunction(Login1.UserName, Login1.Password))
        {
           // e.Authenticated = true;
            Login1.Visible = false;
            MessageLabel.Text = "Successfully Logged In";
        }
        else
        {
            e.Authenticated = false;
        }
    }
    protected void Login1_LoginError(object sender, EventArgs e)
    {
        if (ViewState["LoginErrors"] == null)
            ViewState["LoginErrors"] = 0;
        int ErrorCount = (int)ViewState["LoginErrors"] + 1;
        ViewState["LoginErrors"] = ErrorCount;
        if ((ErrorCount > 3) && (Login1.PasswordRecoveryUrl != string.Empty))
            Response.Redirect(Login1.PasswordRecoveryUrl);
    }
    private bool YourValidationFunction(string UserName, string Password)
    {
        bool boolReturnValue = false;
        string strConnection = "server=.;database=Vendor;uid=sa;pwd=wintellect;";
        SqlConnection sqlConnection = new SqlConnection(strConnection);
        String SQLQuery = "SELECT UserName, Password FROM Login";
        SqlCommand command = new SqlCommand(SQLQuery, sqlConnection);
        SqlDataReader Dr;
        sqlConnection.Open();
        Dr = command.ExecuteReader();
        while (Dr.Read())
        {
            if ((UserName == Dr["UserName"].ToString()) & (Password ==
Dr["Password"].ToString()))
            {
               boolReturnValue = true;
            }
            Dr.Close();
            return boolReturnValue;
        }
        return boolReturnValue;
    }
}
```

If you insert wrong username and password then message will show like this:



Log In

Please enter your user name and password for login.

User Name: test

Password:

Remember me next time.

Your login attempt was not successful. Please try again.

Log In

Register

Additional Help

If you insert right usename, password then redirect your page whereever you want or you can show message in ErrorLabel, Like this:

Successfully Logged In

I am attaching my database with application in App_Data folder, if u want use my database then attach my .MDF file.

ASP.NET - Configuration

The behavior of an ASP.NET application is affected by different settings in the configuration files:

- machine.config
- web.config

The machine.config file contains default and the machine-specific value for all supported settings. The machine settings are controlled by the system administrator and applications are generally not given access to this file.

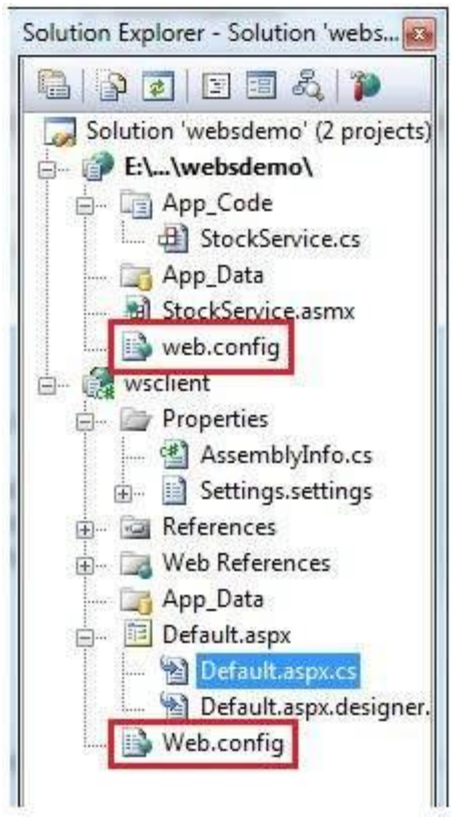
An application however, can override the default values by creating web.config files in its roots folder. The web.config file is a subset of the machine.config file.

If the application contains child directories, it can define a web.config file for each folder. Scope of each configuration file is determined in a hierarchical top-down manner.

Any web.config file can locally extend, restrict, or override any settings defined on the upper level.

Visual Studio generates a default web.config file for each project. An application can execute without a web.config file, however, you cannot debug an application without a web.config file.

The following figure shows the Solution Explorer for the sample example used in the web services tutorial:



In this application, there are two web.config files for two projects i.e., the web service and the web site calling the web service.

The web.config file has the configuration element as the root node. Information inside this element is grouped into two main areas: the configuration section-handler declaration area, and the configuration section settings area.

The following code snippet shows the basic syntax of a configuration file:

```
<configuration>
  <!-- Configuration section-handler declaration area. -->
    <configSections>
      <section name="section1" type="section1Handler" />
      <section name="section2" type="section2Handler" />
    </configSections>
  <!-- Configuration section settings area. -->
  <section1>
    <s1Setting1 attribute1="attr1" />
  </section1>

  <section2>
    <s2Setting1 attribute1="attr1" />
  </section2>

  <system.web>
    <authentication mode="Windows" />
  </system.web>

</configuration>
```

Configuration Section Handler declarations

The configuration section handlers are contained within the <configSections> tags. Each configuration handler specifies name of a configuration section, contained within the file, which provides some configuration data. It has the following basic syntax:

```
<configSections>
  <section />
  <sectionGroup />
  <remove />
  <clear/>
</configSections>
```

It has the following elements:

- **Clear** - It removes all references to inherited sections and section groups.
- **Remove** - It removes a reference to an inherited section and section group.
- **Section** - It defines an association between a configuration section handler and a configuration element.
- **Section group** - It defines an association between a configuration section handler and a configuration section.

Application Settings

The application settings allow storing application-wide name-value pairs for read-only access. For example, you can define a custom application setting as:

```
<configuration>
  <appSettings>
    <add key="Application Name" value="MyApplication" />
  </appSettings>
</configuration>
```

For example, you can also store the name of a book and its ISBN number:

```
<configuration>
  <appSettings>
    <add key="appISBN" value="0-273-68726-3" />
    <add key="appBook" value="Corporate Finance" />
  </appSettings>
</configuration>
```

Connection Strings

The connection strings show which database connection strings are available to the website. For example:

```
<connectionStrings>
  <add name="ASPDotNetStepByStepConnectionString"
    connectionString="Provider=Microsoft.Jet.OLEDB.4.0;
    Data Source=E:\\projects\datacaching\ /
    datacaching\App_Data\ASPDotNetStepByStep.mdb"
    providerName="System.Data.OleDb" />

  <add name="booksConnectionString"
    connectionString="Provider=Microsoft.Jet.OLEDB.4.0;
    Data Source=C:\ \databinding\App_Data\books.mdb"
    providerName="System.Data.OleDb" />
</connectionStrings>
```

System.Web Element

The system.web element specifies the root element for the ASP.NET configuration section and contains configuration elements that configure ASP.NET Web applications and control how the applications behave.

It holds most of the configuration elements needed to be adjusted in common applications. The basic syntax for the element is as given:

```
<system.web>
  <anonymousIdentification>
  <authentication>
  <authorization>
  <browserCaps>
  <caching>
  <clientTarget>
  <compilation>
  <customErrors>
  <deployment>
  <deviceFilters>
  <globalization>
  <healthMonitoring>
  <hostingEnvironment>
  <httpCookies>
  <httpHandlers>
  <httpModules>
  <httpRuntime>
  <identity>
  <machineKey>
  <membership>
```

```
   <mobileControls>
   <pages>
   <processModel>
   <profile>
   <roleManager>
   <securityPolicy>
   <sessionPageState>
   <sessionState>
   <siteMap>
   <trace>
   <trust>
   <urlMappings>
   <webControls>
   <webParts>
   <webServices>
   <xhtmlConformance>
</system.web>
```

The following table provides brief description of some of common sub elements of the **system.web** element:

AnonymousIdentification

This is required to identify users who are not authenticated when authorization is required.

Authentication

It configures the authentication support. The basic syntax is as given:

```
<authentication mode="[Windows|Forms|Passport|None]">

   <forms>...</forms>

   <passport/>

</authentication>
```

Authorization

It configures the authorization support. The basic syntax is as given:

```
<authorization>

   <allow .../>

   <deny .../>

</authorization>
```

Caching

It Configures the cache settings. The basic syntax is as given:

```
<caching>
   <cache>...</cache>
   <outputCache>...</outputCache>
   <outputCacheSettings>...</outputCacheSettings>
   <sqlCacheDependency>...</sqlCacheDependency>
</caching>
```

CustomErrors

It defines custom error messages. The basic syntax is as given:

```
<customErrors defaultRedirect="url" mode="On|Off|RemoteOnly">

   <error. . ./>

</customErrors>
```

Deployment

It defines configuration settings used for deployment. The basic syntax is as follows:

```
<deployment retail="true|false" />
```

HostingEnvironment

It defines configuration settings for hosting environment. The basic syntax is as follows:

```
<hostingEnvironment idleTimeout="HH:MM:SS" shadowCopyBinAssemblies="true|false"

   shutdownTimeout="number" urlMetadataSlidingExpiration="HH:MM:SS" />
```

Identity

It configures the identity of the application. The basic syntax is as given:

```
<identity impersonate="true|false" userName="domain\username"

   password="<secure password>"/>
```

MachineKey

It configures keys to use for encryption and decryption of Forms authentication cookie data.

It also allows configuring a validation key that performs message authentication checks on view-state data and forms authentication tickets. The basic syntax is:

```
<machineKey validationKey="AutoGenerate,IsolateApps" [String]
   decryptionKey="AutoGenerate,IsolateApps" [String]
   validation="HMACSHA256" [SHA1 | MD5 | 3DES | AES | HMACSHA256 |
   HMACSHA384 | HMACSHA512 | alg:algorithm_name]
   decryption="Auto" [Auto | DES | 3DES | AES | alg:algorithm_name]
/>
```

Membership

This configures parameters of managing and authenticating user accounts. The basic syntax is:

```
<membership defaultProvider="provider name"
   userIsOnlineTimeWindow="number of minutes" hashAlgorithmType="SHA1">
   <providers>...</providers>
```

```
</membership>
```

Pages

It provides page-specific configurations. The basic syntax is:

```
<pages asyncTimeout="number" autoEventWireup="[True|False]"
    buffer="[True|False]" clientIDMode="[AutoID|Predictable|Static]"
    compilationMode="[Always|Auto|Never]"
    controlRenderingCompatibilityVersion="[3.5|4.0]"
    enableEventValidation="[True|False]"
    enableSessionState="[True|False|ReadOnly]"
    enableViewState="[True|False]"
    enableViewStateMac="[True|False]"
    maintainScrollPositionOnPostBack="[True|False]"
    masterPageFile="file path"
    maxPageStateFieldLength="number"
    pageBaseType="typename, assembly"
    pageParserFilterType="string"
    smartNavigation="[True|False]"
    styleSheetTheme="string"
    theme="string"
    userControlBaseType="typename"
    validateRequest="[True|False]"
    viewStateEncryptionMode="[Always|Auto|Never]" >

  <controls>...</controls>
  <namespaces>...</namespaces>
  <tagMapping>...</tagMapping>
  <ignoreDeviceFilters>...</ignoreDeviceFilters>
</pages>
```

Profile

It configures user profile parameters. The basic syntax is:

```
<profile enabled="true|false" inherits="fully qualified type reference"
  automaticSaveEnabled="true|false" defaultProvider="provider name">
  <properties>...</properties>
  <providers>...</providers>
</profile>
```

RoleManager

It configures settings for user roles. The basic syntax is:

```
<roleManager cacheRolesInCookie="true|false" cookieName="name"
  cookiePath="/" cookieProtection="All|Encryption|Validation|None"
  cookieRequireSSL="true|false " cookieSlidingExpiration="true|false "
  cookieTimeout="number of minutes" createPersistentCookie="true|false"
  defaultProvider="provider name" domain="cookie domain">
  enabled="true|false"
  maxCachedResults="maximum number of role names cached"

  <providers>...</providers>
</roleManager>
```

SecurityPolicy

It configures the security policy. The basic syntax is:

```
<securityPolicy>

  <trustLevel />

</securityPolicy>
```

UrlMappings

It defines mappings to hide the original URL and provide a more user friendly URL. The basic syntax is:

```
<urlMappings enabled="true|false">

  <add.../>

  <clear />

  <remove.../>

</urlMappings>
```

WebControls

It provides the name of shared location for client scripts. The basic syntax is:

```
<webControls clientScriptsLocation="String" />
```

WebServices

This configures the web services.

ASP.NET - Web Services

A web service is a web-based functionality accessed using the protocols of the web to be used by the web applications. There are three aspects of web service development:

- Creating the web service
- Creating a proxy
- Consuming the web service

Creating a Web Service

A web service is a web application which is basically a class consisting of methods that could be used by other applications. It also follows a code-behind architecture such as the ASP.NET web pages, although it does not have a user interface.

To understand the concept let us create a web service to provide stock price information. The clients can query about the name and price of a stock based on the stock symbol. To keep this example simple, the values are hardcoded in a two-dimensional array. This web service has three methods:

- A default HelloWorld method
- A GetName Method
- A GetPrice Method

Take the following steps to create the web service:

Step (1) : Select File -> New -> Web Site in Visual Studio, and then select ASP.NET Web Service.

Step (2) : A web service file called Service.asmx and its code behind file, Service.cs is created in the App_Code directory of the project.

Step (3) : Change the names of the files to StockService.asmx and StockService.cs.

Step (4) : The .asmx file has simply a WebService directive on it:

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/StockService.cs"
Class="StockService" %>
```

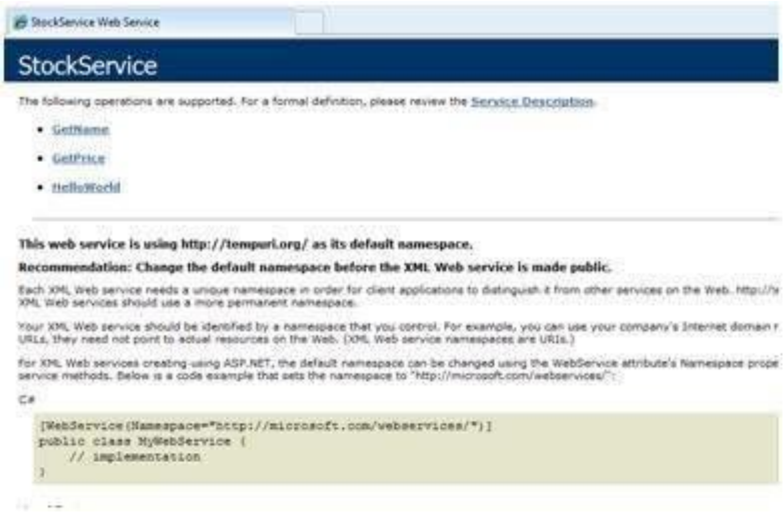
Step (5) : Open the StockService.cs file, the code generated in it is the basic Hello World service. The default web service code behind file looks like the following:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Linq;

using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

using System.Xml.Linq;

namespace StockService
{
  // <summary>
  // Summary description for Service1
  // <summary>

  [WebService(Namespace = "http://tempuri.org/")]
  [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
  [ToolboxItem(false)]

  // To allow this Web Service to be called from script,
  // using ASP.NET AJAX, uncomment the following line.
  // [System.Web.Script.Services.ScriptService]

  public class Service1 : System.Web.Services.WebService
  {
    [WebMethod]

    public string HelloWorld()
    {
      return "Hello World";
    }
  }
}
```

Step (6) : Change the code behind file to add the two dimensional array of strings for stock symbol, name and price and two web methods for getting the stock information.

```
using System;
using System.Linq;

using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

using System.Xml.Linq;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]

// To allow this Web Service to be called from script,
// using ASP.NET AJAX, uncomment the following line.
// [System.Web.Script.Services.ScriptService]

public class StockService : System.Web.Services.WebService
{
  public StockService () {
    //Uncomment the following if using designed components
    //InitializeComponent();
  }

  string[,] stocks =
  {
    {"RELIND", "Reliance Industries", "1060.15"},
    {"ICICI", "ICICI Bank", "911.55"},
    {"JSW", "JSW Steel", "1201.25"},
    {"WIPRO", "Wipro Limited", "1194.65"},
    {"SATYAM", "Satyam Computers", "91.10"}
  };
  [WebMethod]
  public string HelloWorld() {
    return "Hello World";
  }

  [WebMethod]
  public double GetPrice(string symbol)
  {
    //it takes the symbol as parameter and returns price
    for (int i = 0; i < stocks.GetLength(0); i++)
    {
      if (String.Compare(symbol, stocks[i, 0], true) == 0)
      return Convert.ToDouble(stocks[i, 2]);
    }

    return 0;
  }

  [WebMethod]
  public string GetName(string symbol)
  {
    // It takes the symbol as parameter and
    // returns name of the stock
    for (int i = 0; i < stocks.GetLength(0); i++)
    {
      if (String.Compare(symbol, stocks[i, 0], true) == 0)
      return stocks[i, 1];
    }

    return "Stock Not Found";
  }
}
```

Step (7) : Running the web service application gives a web service test page, which allows testing the service methods.



StockService Web Service

StockService

The following operations are supported. For a formal definition, please review the Service Description.

- GetName
- GetPrice
- HelloWorld

This web service is using http://tempuri.org/ as its default namespace.

Recommendation: Change the default namespace before the XML Web service is made public.

Each XML Web service needs a unique namespace in order for client applications to distinguish it from other services on the Web. http://t
XML Web services should use a more permanent namespace.

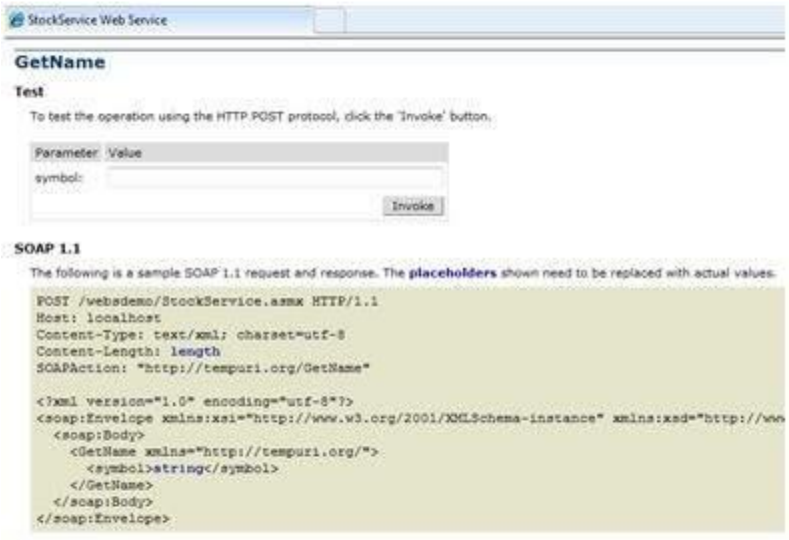
Your XML Web service should be identified by a namespace that you control. For example, you can use your company's Internet domain n
URLs, they need not point to actual resources on the Web. (XML Web service namespaces are URIs.)

For XML Web services creating using ASP.NET, the default namespace can be changed using the WebService attribute's Namespace prope
service methods. Below is a code example that sets the namespace to "http://microsoft.com/webservices/":

C#

```
[WebService(Namespace="http://microsoft.com/webservices/")]
public class MyWebService {
    // implementation
}
```

Step (8) : Click on a method name, and check whether it runs properly.



StockService Web Service

GetName

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

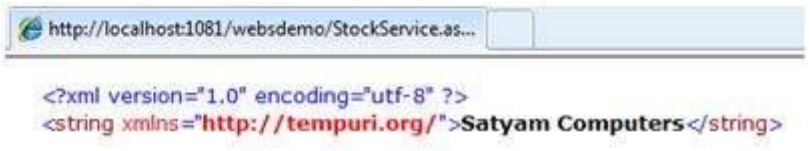
Parameter	Value
symbol:	

SOAP 1.1

The following is a sample SOAP 1.1 request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /websdemo/StockService.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/GetName"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://ww
  <soap:Body>
    <GetName xmlns="http://tempuri.org/">
      <symbol>string</symbol>
    </GetName>
  </soap:Body>
</soap:Envelope>
```

Step (9) : For testing the GetName method, provide one of the stock symbols, which are hard coded, it returns the name of the stock



http://localhost:1081/websdemo/StockService.as...

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">Satyam Computers</string>
```

Consuming the Web Service

For using the web service, create a web site under the same solution. This could be done by right clicking on the Solution name in the Solution Explorer. The web page calling the web service should have a label control to display the returned results and two button controls one for post back and another for calling the service.

The content file for the web application is as follows:

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

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>
  <body>
    <form id="form1" runat="server">
      <div>
        <h3>Using the Stock Service</h3>
        <br /> <br />
        <asp:Label ID="lblmessage" runat="server"></asp:Label>
        <br /> <br />
        <asp:Button ID="btnpostback" runat="server" onclick="Button1_Click" Text="Post Back"
style="width:132px" />
        <asp:Button ID="btnservice" runat="server" onclick="btnservice_Click"  Text="Get Stock"
style="width:99px" />
      </div>
    </form>
  </body>
</html>
```

The code behind file for the web application is as follows:

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;

using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

using System.Xml.Linq;

//this is the proxy
using localhost;

namespace wsclient
{
  public partial class _Default : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    {
      if (!IsPostBack)
      {
        lblmessage.Text = "First Loading Time: " +  DateTime.Now.ToLongTimeString
      }
      else
      {
        lblmessage.Text = "PostBack at: " + DateTime.Now.ToLongTimeString();
      }
    }

    protected void btnservice_Click(object sender, EventArgs e)
    {
      StockService proxy = new StockService();
      lblmessage.Text = String.Format("Current SATYAM Price:{0}",
      proxy.GetPrice("SATYAM").ToString());
    }
  }
}
```

Creating the Proxy

A proxy is a stand-in for the web service codes. Before using the web service, a proxy must be created. The proxy is registered with the client application. Then the client application makes the calls to the web service as it were using a local method.

The proxy takes the calls, wraps it in proper format and sends it as a SOAP request to the server. SOAP stands for Simple Object Access Protocol. This protocol is used for exchanging web service data.

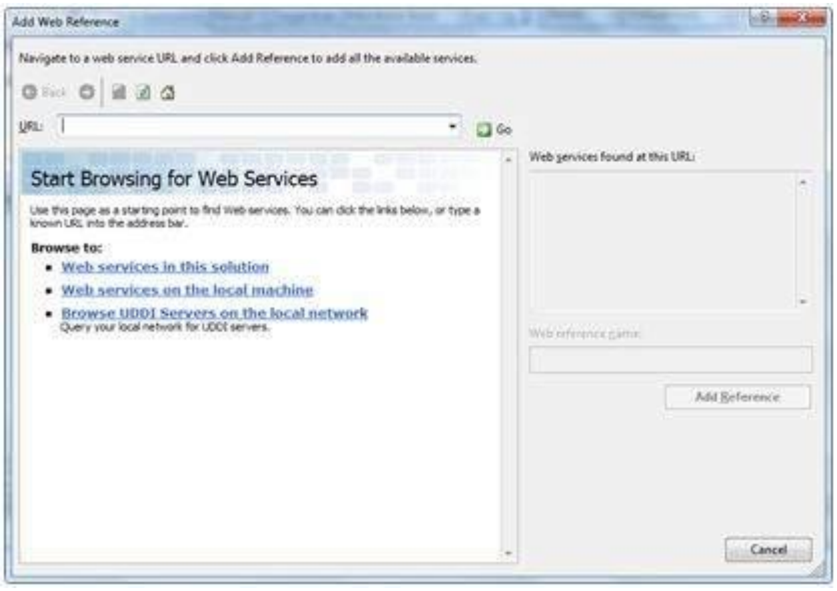
When the server returns the SOAP package to the client, the proxy decodes everything and presents it to the client application.

Before calling the web service using the btnservice_Click, a web reference should be added to the application. This creates a proxy class transparently, which is used by the btnservice_Click event.

```
protected void btnservice_Click(object sender, EventArgs e)
{
  StockService proxy = new StockService();
  lblmessage.Text = String.Format("Current SATYAM Price: {0}",
  proxy.GetPrice("SATYAM").ToString());
}
```

Take the following steps for creating the proxy:

Step (1) : Right click on the web application entry in the Solution Explorer and click on 'Add Web Reference'.



Navigate to a web service URL and click Add Reference to add all the available services.

Start Browsing for Web Services

Use this page as a starting point to find Web services. You can click the links below, or type a known URL into the address bar.

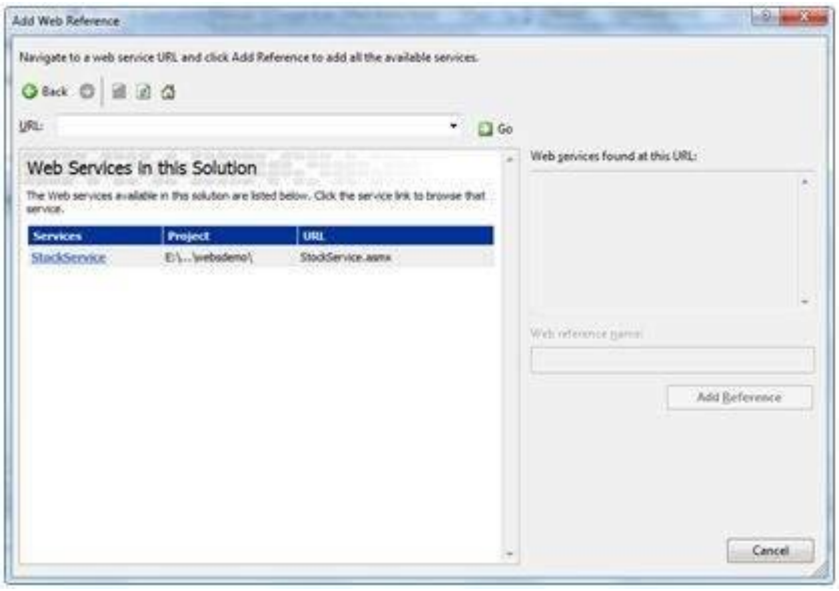
Browse to:

- Web services in this solution
- Web services on the local machine
- Browse UDDI Servers on the local network
  Query your local network for UDDI servers.

Web services found at this URL:

Web reference name:

Step (2) : Select 'Web Services in this solution'. It returns the StockService reference.



Navigate to a web service URL and click Add Reference to add all the available services.

Web Services in this Solution

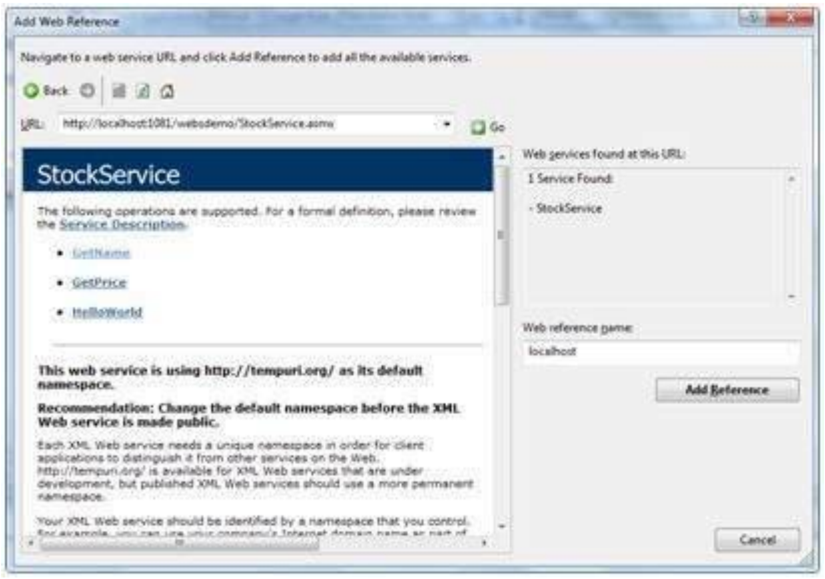
The Web services available in this solution are listed below. Click the service link to browse that service.

Services	Project	URL
StockService	E:\...\webdemo\	StockService.asmx

Web services found at this URL:

Web reference name:

Step (3) : Clicking on the service opens the test web page. By default the proxy created is called 'localhost', you can rename it. Click on 'Add Reference' to add the proxy to the client application.



Add Web Reference

Navigate to a web service URL and click Add Reference to add all the available services.

URL: http://localhost:1081/webdemo/StockService.asmx

StockService

The following operations are supported. For a formal definition, please review the Service Description.

- GetName
- GetPrice
- HelloWorld

This web service is using http://tempuri.org/ as its default namespace.

Recommendation: Change the default namespace before the XML Web service is made public.

Each XML Web service needs a unique namespace in order for client applications to distinguish it from other services on the Web. http://tempuri.org/ is available for XML Web services that are under development, but published XML Web services should use a more permanent namespace.

Your XML Web service should be identified by a namespace that you control.

Web services found at this URL:

1 Service Found:

- StockService

Web reference name: localhost

Include the proxy in the code behind file by adding:

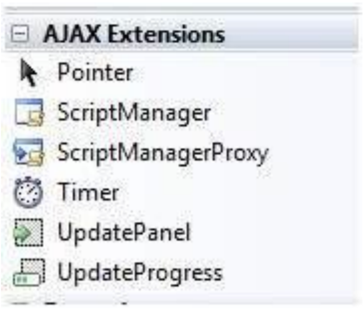
using localhost;

ASP.NET - Ajax Control

AJAX stands for Asynchronous JavaScript and XML. This is a cross platform technology which speeds up response time. The AJAX server controls add script to the page which is executed and processed by the browser.

However like other ASP.NET server controls, these AJAX server controls also can have methods and event handlers associated with them, which are processed on the server side.

The control toolbox in the Visual Studio IDE contains a group of controls called the 'AJAX Extensions'



The ScriptManager Control

The ScriptManager control is the most important control and must be present on the page for other controls to work.

It has the basic syntax:

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

</asp:ScriptManager>
```

If you create an 'Ajax Enabled site' or add an 'AJAX Web Form' from the 'Add Item' dialog box, the web form automatically contains the script manager control. The ScriptManager control takes care of the client-side script for all the server side controls.

The UpdatePanel Control

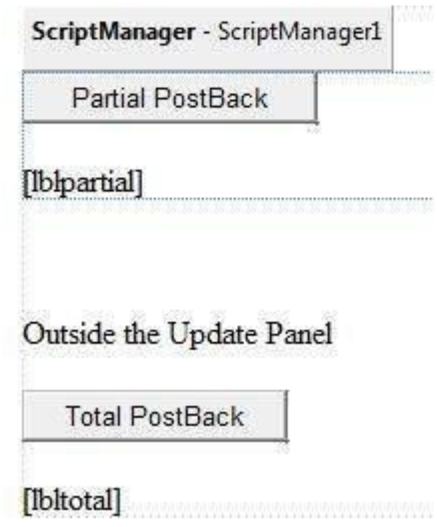
The UpdatePanel control is a container control and derives from the Control class. It acts as a container for the child controls within it and does not have its own interface. When a control inside it triggers a post back, the UpdatePanel intervenes to initiate the post asynchronously and update just that portion of the page.

For example, if a button control is inside the update panel and it is clicked, only the controls within the update panel will be affected, the controls on the other parts of the page will not be affected. This is called the partial post back or the asynchronous post back.

Example

Add an AJAX web form in your application. It contains the script manager control by default. Insert an update panel. Place a button control along with a label control within the update panel control. Place another set of button and label outside the panel.

The design view looks as follows:



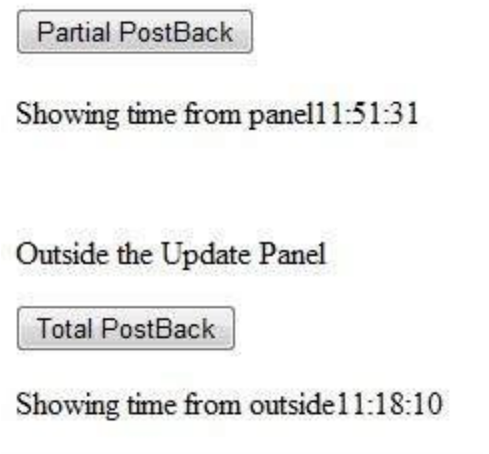
The source file is as follows:

```
<form id="form1" runat="server">
   <div>
      <asp:ScriptManager ID="ScriptManager1" runat="server" />
   </div>

   <asp:UpdatePanel ID="UpdatePanel1" runat="server">
      <ContentTemplate>
         <asp:Button ID="btnpartial" runat="server" onclick="btnpartial_Click" Text="Partial
PostBack"/>
         <br />
      <br />
      <asp:Label ID="lblpartial" runat="server"></asp:Label>
    </ContentTemplate>
  </asp:UpdatePanel>

  <p> </p>
  <p>Outside the Update Panel</p>
  <p>
    <asp:Button ID="btntotal" runat="server" onclick="btntotal_Click" Text="Total PostBack" />
  </p>

  <asp:Label ID="lbltotal" runat="server"></asp:Label>
</form>
```

Both the button controls have same code for the event handler:

```
string time = DateTime.Now.ToLongTimeString();

lblpartial.Text = "Showing time from panel" + time;

lbltotal.Text = "Showing time from outside" + time;
```

Observe that when the page is executed, if the total post back button is clicked, it updates time in both the labels but if the partial post back button is clicked, it only updates the label within the update panel.



```
Showing time from panel11:51:31

Outside the Update Panel

Showing time from outside11:18:10
```

A page can contain multiple update panels with each panel containing other controls like a grid and displaying different part of data.

When a total post back occurs, the update panel content is updated by default. This default mode could be changed by changing the UpdateMode property of the control. Let us look at other properties of the update panel.

Properties of the UpdatePanel Control

The following table shows the properties of the update panel control:

Properties	Description
ChildrenAsTriggers	This property indicates whether the post backs are coming from the child controls, which cause the update panel to refresh.
ContentTemplate	It is the content template and defines what appears in the update panel when it is rendered.
ContentTemplateContainer	Retrieves the dynamically created template container object and used for adding child controls programmatically.
IsInPartialRendering	Indicates whether the panel is being updated as part of the partial post back.
RenderMode	Shows the render modes. The available modes are Block and Inline.
UpdateMode	Gets or sets the rendering mode by determining some conditions.
Triggers	Defines the collection trigger objects each corresponding to an event causing the panel to refresh automatically.

Methods of the UpdatePanel Control

The following table shows the methods of the update panel control:

Methods	Description
CreateContentTemplateContainer	Creates a Control object that acts as a container for child controls that define the UpdatePanel control's content.
CreateControlCollection	Returns the collection of all controls that are contained in the UpdatePanel control.
Initialize	Initializes the UpdatePanel control trigger collection if partial-page rendering is enabled.
Update	Causes an update of the content of an UpdatePanel control.

The behavior of the update panel depends upon the values of the UpdateMode property and ChildrenAsTriggers property.

UpdateMode	ChildrenAsTriggers	Effect
Always	False	Illegal parameters.
Always	True	UpdatePanel refreshes if whole page refreshes or a child control on it posts back.
Conditional	False	UpdatePanel refreshes if whole page refreshes or a triggering control outside it initiates a refresh.
Conditional	True	UpdatePanel refreshes if whole page refreshes or a child control on it posts back or a triggering control outside it initiates a refresh.

The UpdateProgress Control

The UpdateProgress control provides a sort of feedback on the browser while one or more update panel controls are being updated. For example, while a user logs in or waits for server response while performing some database oriented job.

It provides a visual acknowledgement like "Loading page...", indicating the work is in progress.

The syntax for the UpdateProgress control is:

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server" DynamicLayout="true"
AssociatedUpdatePanelID="UpdatePanel1" >
  <ProgressTemplate>
    Loading...
  </ProgressTemplate>
</asp:UpdateProgress>
```

The above snippet shows a simple message within the ProgressTemplate tag. However, it could be an image or other relevant controls. The UpdateProgress control displays for every asynchronous postback unless it is assigned to a single update panel using the AssociatedUpdatePanelID property.

Properties of the UpdateProgress Control

The following table shows the properties of the update progress control:

Properties	Description
AssociatedUpdatePanelID	Gets and sets the ID of the update panel with which this control is associated.

Attributes	Gets or sets the cascading style sheet (CSS) attributes of the UpdateProgress control.
DisplayAfter	Gets and sets the time in milliseconds after which the progress template is displayed. The default is 500.
DynamicLayout	Indicates whether the progress template is dynamically rendered.
ProgressTemplate	Indicates the template displayed during an asynchronous post back which takes more time than the DisplayAfter time.

Methods of the UpdateProgress Control

The following table shows the methods of the update progress control:

Methods	**Description**
GetScriptDescriptors	Returns a list of components, behaviors, and client controls that are required for the UpdateProgress control's client functionality.
GetScriptReferences	Returns a list of client script library dependencies for the UpdateProgress control.

The Timer Control

The timer control is used to initiate the post back automatically. This could be done in two ways:

(1) Setting the Triggers property of the UpdatePanel control:

```
<Triggers>
  <asp:AsyncPostBackTrigger ControlID="btnpanel2" EventName="Click" />
</Triggers>
```

(2) Placing a timer control directly inside the UpdatePanel to act as a child control trigger. A single timer can be the trigger for multiple UpdatePanels.

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Always">
  <ContentTemplate>
    <asp:Timer ID="Timer1" runat="server" Interval="1000">
      </asp:Timer>
    <asp:Label ID="Label1" runat="server" Height="101px" style="width:304px" >
      </asp:Label>
  </ContentTemplate>

</asp:UpdatePanel>
```

ASP.NET - Deployment

There are two categories of ASP.NET deployment:

- **Local deployment** : In this case, the entire application is contained within a virtual directory and all the contents and assemblies are contained within it and available to the application.
- **Global deployment** : In this case, assemblies are available to every application running on the server.

There are different techniques used for deployment, however, we will discuss the following most common and easiest ways of deployment:

- XCOPY deployment
- Copying a Website
- Creating a set up project

XCOPY Deployment

XCOPY deployment means making recursive copies of all the files to the target folder on the target machine. You can use any of the commonly used techniques:

- FTP transfer
- Using Server management tools that provide replication on a remote site
- MSI installer application

XCOPY deployment simply copies the application file to the production server and sets a virtual directory there. You need to set a virtual directory using the Internet Information Manager Microsoft Management Console (MMC snap-in).

Copying a Website

The Copy Web Site option is available in Visual Studio. It is available from the Website -> Copy Web Site menu option. This menu item allows copying the current web site to another local or remote location. It is a sort of integrated FTP tool.

Using this option, you connect to the target destination, select the desired copy mode:

- Overwrite
- Source to Target Files
- Sync UP Source And Target Projects

Then proceed with copying the files physically. Unlike the XCOPY deployment, this process of deployment is done from Visual Studio environment. However, there are following problems with both the above deployment methods:

- You pass on your source code.
- There is no pre-compilation and related error checking for the files.
- The initial page load will be slow.

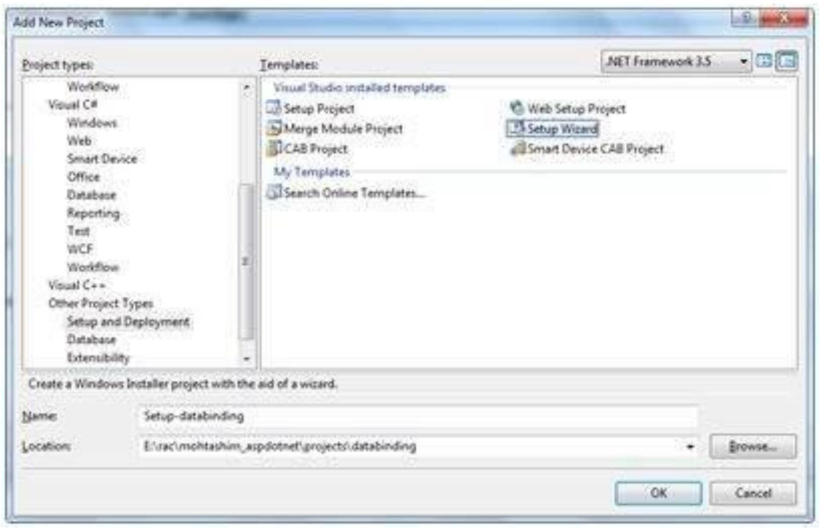
Creating a Setup Project

In this method, you use Windows Installer and package your web applications so it is ready to deploy on the production server. Visual Studio allows you to build deployment packages. Let us test this on one of our existing project, say the data binding project.

Open the project and take the following steps:

Step (1) : Select File -> Add -> New Project with the website root directory highlighted in the Solution Explorer.

Step (2) : Select Setup and Deployment, under Other Project Types. Select Setup Wizard.

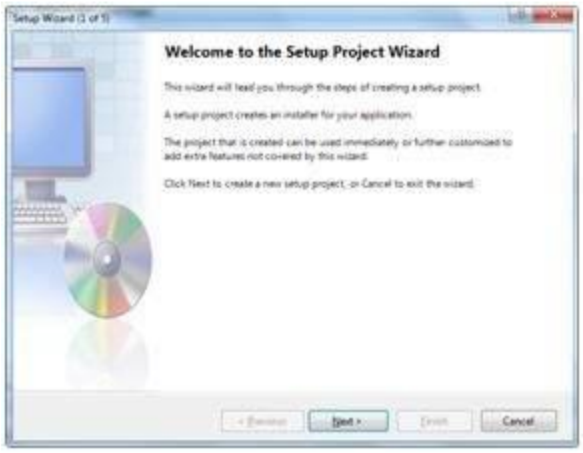


Create a Windows Installer project with the aid of a wizard.

Name: Setup-databinding

Location: E:\rac\mohtashim_aspdotnet\projects\databinding

Step (3) : Choosing the default location ensures that the set up project will be located in its own folder under the root directory of the site. Click on okay to get the first splash screen of the wizard.



Setup Wizard (1 of 5)

Welcome to the Setup Project Wizard

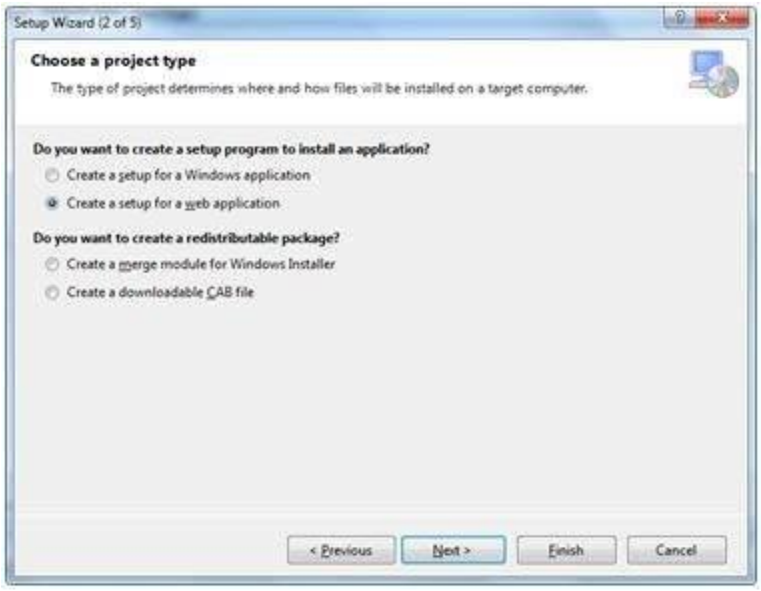
This wizard will lead you through the steps of creating a setup project.

A setup project creates an installer for your application.

The project that is created can be used immediately or further customized to add extra features not covered by this wizard.

Click Next to create a new setup project, or Cancel to exit the wizard.

Step (4) : Choose a project type. Select 'Create a setup for a web application'.



Setup Wizard (2 of 5)

Choose a project type

The type of project determines where and how files will be installed on a target computer.

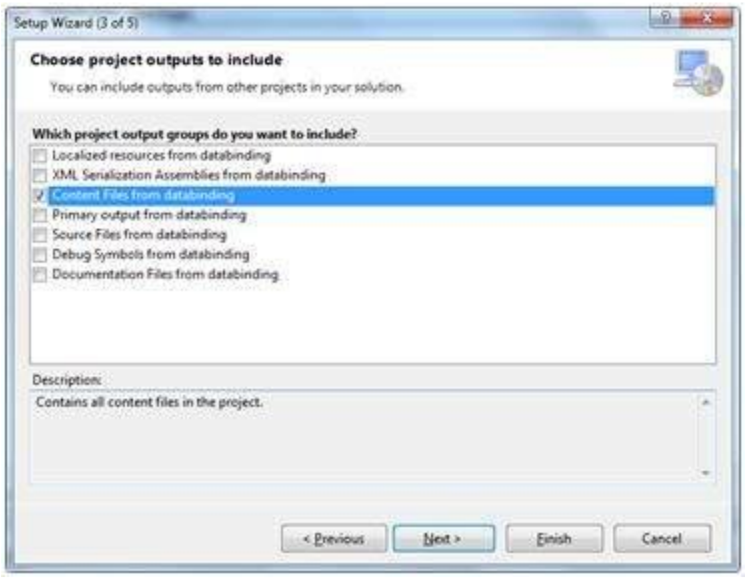
Do you want to create a setup program to install an application?

- ○ Create a setup for a Windows application
- ● Create a setup for a web application

Do you want to create a redistributable package?

- ○ Create a merge module for Windows Installer
- ○ Create a downloadable CAB file

Step (5) : Next, the third screen asks to choose project outputs from all the projects in the solution. Check the check box next to 'Content Files from...'



Setup Wizard (3 of 5)

Choose project outputs to include

You can include outputs from other projects in your solution.

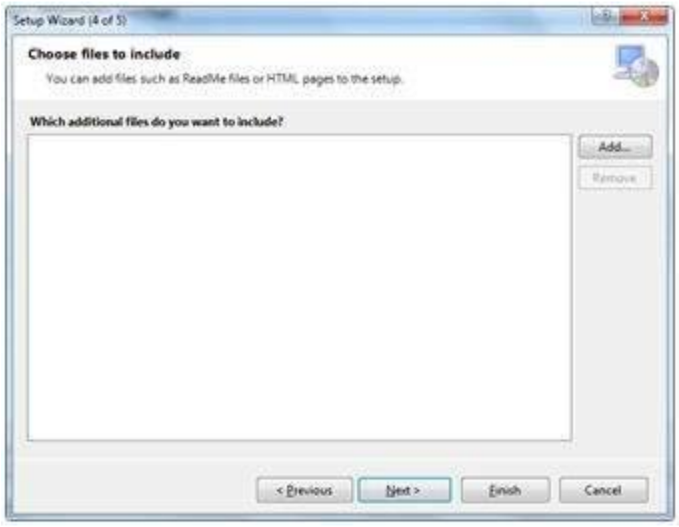
Which project output groups do you want to include?

- ☐ Localized resources from databinding
- ☐ XML Serialization Assemblies from databinding
- ☑ Content Files from databinding
- ☐ Primary output from databinding
- ☐ Source Files from databinding
- ☐ Debug Symbols from databinding
- ☐ Documentation Files from databinding

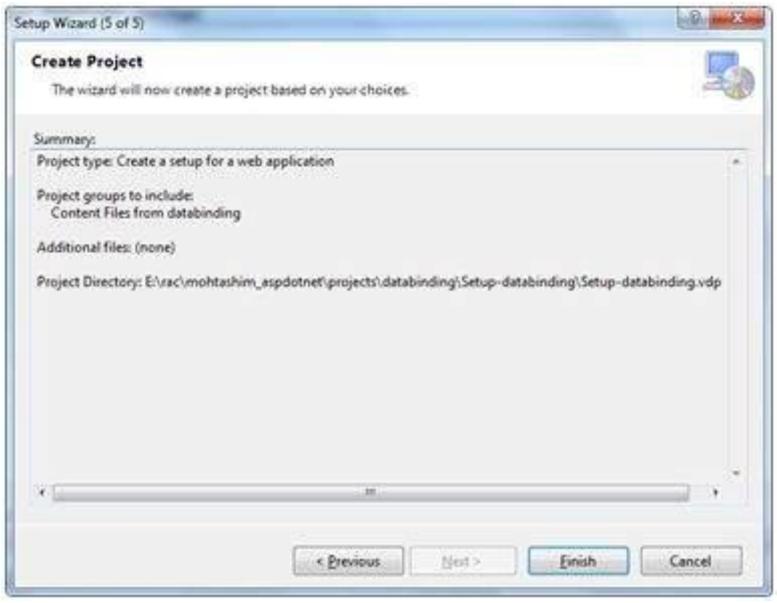
Description:

Contains all content files in the project.

Step (6) : The fourth screen allows including other files like ReadMe. However, in our case there is no such file. Click on finish.


Setup Wizard (4 of 5)
Choose files to include
You can add files such as ReadMe files or HTML pages to the setup.
Which additional files do you want to include?
Add...
Remove
< Previous
Next >
Finish
Cancel

Step (7) : The final screen displays a summary of settings for the set up project.



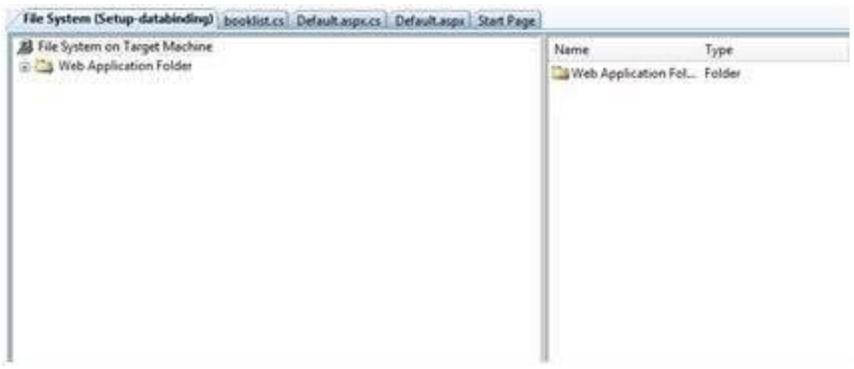
Summary:
Project type: Create a setup for a web application

Project groups to include:
  Content Files from databinding

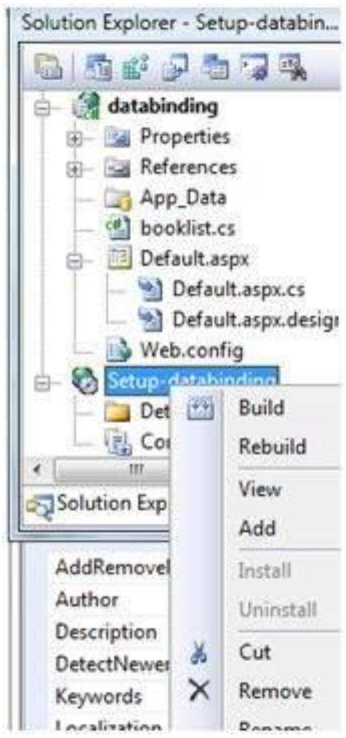
Additional files: (none)

Project Directory: E:\rac\mohtashim_aspdotnet\projects\databinding\Setup-databinding\Setup-databinding.vdp

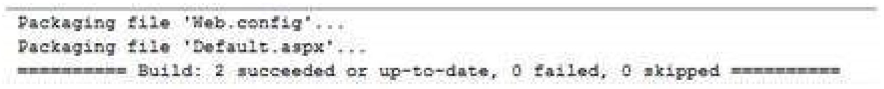
Step (8) : The Set up project is added to the Solution Explorer and the main design window shows a file system editor.



Step (9) : Next step is to build the setup project. Right click on the project name in the Solution Explorer and select Build.



Step (10) : When build is completed, you get the following message in the Output window:



```
Packaging file 'Web.config'...
Packaging file 'Default.aspx'...
========== Build: 2 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

Two files are created by the build process:

- Setup.exe
- Setup-databinding.msi

You need to copy these files to the server. Double-click the setup file to install the content of the .msi file on the local machine.