Unit-III

ADVANCED ASP.NET

Active Server Pages Framework (.aspx) is a server side programming language capable of building powerful web applications. Asp.Net has built in capabilities which make easier for programmers to build applications. With Asp.net you can create dynamic web pages, access databases, send emails, and manage file system and much more.

Language Selection

Asp.net requires a support language. The .Net Framework has support for several programming languages with the following as the most popular ones:

- VB (Visual Basic)

- C# (C Sharp)

- JScript

To specify the selected language a **Page Directive** is included on an Asp.Net document:

<%@ Page Language="VB" %>

Active Server Pages or ASP, as it is more commonly known, is a technology that enables you to make dynamic and interactive web pages.

ASP uses server-side scripting to dynamically produce web pages that are not affected by the type of browser the web site visitor is using.

The default scripting language used for writing ASP is VBScript, although you can use other scripting languages like JScript (Microsoft's version of JavaScript).

ASP pages have the extension .asp instead of .htm, when a page with the extension .asp is requested by a browser the web server knows to interpret any ASP contained within the web page before sending the HTML produced to the browser. This way all the ASP is run on the web server and no ASP will ever be passed to the web browser.

Any web pages containing ASP cannot be run by just simply opening the page in a web browser. The page must be requested through a web server that supports ASP, this is why ASP stands for Active Server Pages, no server, no active pages.

Asp.net Webserver Controls

Server controls are objects placed on a page with the purpose of read and write data. These objects have a consistent programming model and their properties (attributes) can be accessed

programmatically. A Server Control starts with **<asp:** followed by the object name **<asp:textbox .** They must contain two other properties, **ID** which must be unique and **runat="server"** which tells the web server to give the control server side processing. They can be self closed by having a forward slash (/) before the ending tag or have a closing tag:

<asp:textbox id="txt" runat="server" />
<asp:textbox id="txt" runat="server" ></asp:textbox>

After a control is processed by the web server, a standard HTML tag is sent to the browser for displaying. The above example will be rendered as:

<input type="text" name="txt" id="txt" />

Button <asp:button/>

Post back control for form submission.

Properties:

Id="btn"	Creates an unique identifier
Runat="server"	Sets the button for server side processing
Text="Search"	Displays text on the button
Enabled="true/false"	Enables or Disables the button

Example: <asp:button id="btn" runat="server" text="Search"/>

Checkbox <asp:checkbox/>

Selection control (multiple selections).

Properties:

Id="ck"	Creates an unique identifier
Runat="server"	Sets the checkbox for server side processing
Text="Diploma"	Displays text on checkbox
Checked="true/false"	Checks or Uncheck the checkbox

Example: <asp:checkbox id="ck" runat="server" text="Diploma"/>

DropdownList <asp:dropdownlist>…</asp:dropdownlist>

Single selection control that displays a collection of list items.

Properties:

Id="lb"	Creates an unique identifier
Runat="server"	Sets the button for server side processing
	Set auto submission on item selected

AutoPostBack=”true,false”

Example: <asp:dropdownlist id="countries" runat="server">
<asp:listitem Text="Select" Value=""/>
<asp:listitem Text="UK" Value="UK"/>
<asp:listitem Text="Ireland" Value="Ireland"/>

</asp:dropdownlist>

Hyperlink <asp:hyperlink/>

Navigation control to move from page to page.

Properties:

Id=”hl”	Creates an unique identifier
Runat=”server”	Sets the image for server side processing
NavigateURL=”controls.aspx”	Maps the file to link to
Text=”Asp.Net Controls”	Creates the text link

Example: <asp:image id=”img” runat=”server” ImageURL=”ImageName.gif”/>

Image <asp:image/>

Displays image defined as ImageUrl

Properties:

Id=”img”	Creates an unique identifier
Runat=”server”	Sets the image for server side processing
ImageURL=”hello.gif”	Maps the image file to display
Width=”200”	Sets Image width
Height=”200”	Sets image height
ToolTip=”Hello Message”	Display text when mouse is over

Example: <asp:image id=”img” runat=”server” ImageURL=”ImageName.gif”/>

Label <asp:label/>

Output control used to display text at a specific location on a page.

Properties:

Id=”msg”	Creates an unique identifier
Runat=”server”	Sets the textbox for server side processing
Text=”some text”	Displays text inside the box

Example: <asp:label id=”msg” runat=”server” text=”Welcome to ASP.Net”/>

Link Button <asp:linkbutton>

Post back control for form submission. It renders as a link.

Properties:

Id=”lb”	Creates an unique identifier
Runat=”server”	Sets the button for server side processing
Text=”Search”	Displays text on the button
Enabled=”true/false”	Enables or Disables the button

Example: <asp:linkbutton id=”btn” runat=”server” text=”Search”/>

Listbox <asp:listbox>…</asp:listbox>

Multiple selection control that displays a collection of list items.

Properties:

Id=”lb”	Creates an unique identifier
Runat=”server”	Sets the button for server side processing
CssClass=”box”	Sets a Stylesheet class
Rows=”Number”	Sets the visible number of rows
SelectionMode=”single/multiple”	Sets single or multiple selection

```
Example: <asp:listbox id="countries" runat="server">
            <asp:listitem Text="UK" Value="UK"/>
            <asp:listitem Text="Ireland" Value="Ireland"/>

         </asp:lisbox>
```

Panel <asp:panel>…</asp:panel>

Container for other controls.

Properties:

Id=”pn”	Creates an unique identifier
Runat=”server”	Sets the panel for server side processing
Visible=”true/false”	Sets its visibility

```
Example: <asp:panel id="pn" runat="server" visible=”true”>
            <asp:label id=”msg2” runat=”server” text=”Control inside a panel”/>
         </asp:panel>
```

Radio Button <asp:radiobutton/>

Single selection control belonging to a group.

Properties:

Id=”lb”	Creates an unique identifier
Runat=”server”	Sets the radiobutton for server side processing
CssClass=”box”	Sets a Stylesheet class
Text=”Female”	Displays text on radiobutton
GroupName=”gender”	Sets a group it belongs to
Checked=”true/false”	Checks or Uncheck the radiobutton

Example:

<asp:radiobutton id=”rb” runat=”server” GroupName=”gender” text=”Female” checked=”true”/>

<asp:radiobutton id=”rb2” runat=”server” text=”Male” />

Table <asp:table>…</asp:table>

Builds up a HTML table.

Properties:

Id=”lb”	Creates an unique identifier
Runat=”server”	Sets the table for server side processing
BackColor=”#f6f6da”	Sets a background colour
Cellspacing=”2”	Sets distance between cells
Cellpadding=”2”	Sets distance around the cells’ contents
Width=”500”	Set table width
Height=”500”	Sets table height
HorizontalAlign=”center/right”	Sets table alignment
GridLines=”both/none/vertical/horizontal”	Sets what borders to display
BorderWidth=”1”	Sets border height

Table Row <asp:tablerow>…</asp:tablerow>

Creates table rows.

Properties:

BackColor=”#f6f6da”	Sets a background colour
Width=”500”	Set table row width
Height=”500”	Sets table row height
HorizontalAlign=”center/right”	Sets table row alignment

Table Header Cell <asp:tableheadercell>…</asp:tableheadercell>

Creates table heading.

Properties:

BackColor=”#f6f6da”	Sets a background colour
Width=”500”	Set table header width
Height=”500”	Sets table header height
HorizontalAlign=”center/right”	Sets table header alignment
VerticalAlign=”top/bottom”	Sets table header vertical alignment
ColumnSpan=”2”	Merge Columns

Table Cell <asp:tablecell>…</asp:tablecell>

Creates table data.

Properties:

BackColor=”#f6f6da”	Sets a background colour
Width=”500”	Set table data width
Height=”500”	Sets table data height
HorizontalAlign=”center/right”	Sets table data alignment
VerticalAlign=”top/bottom”	Sets table data vertical alignment
ColumnSpan=”2”	Merge Columns
RowSpan=”2”	Merge Rows

Example:

```
<asp:table id="tbl" runat="server" GridLines="Both" BorderWidth="1"
CellPadding="1" CellSpacing="1" BackColor="#FFFFCC" Width="100" Height="100">
        <asp:tablerow>
                <asp:tableheadercell>Prices</asp:tableheadercell>
        </asp:tablerow>
        <asp:tablerow>
<asp:tablecell HorizontalAlign="Center" VerticalAlign="Middle" ColumnSpan="1"
RowSpan="1">500</asp:tablecell>
        </asp:tablerow>
</asp:table>
```

Text Box <asp:textbox >

Single or multiline textbox for data input.

Properties:

Id=”txt”	Creates an unique identifier
Runat=”server”	Sets the textbox for server side processing
	Sets the maximum number of characters allowed

MaxLength=”10”	Sets a Stylesheet class
CssClass=”box”	Sets only read capability (contents can’t be changed)
Readonly=”true/false”	Displays text inside the box
Text=”some text”	Sets the width of textbox
Columns=”20”	Sets the number of lines (textarea)
Rows=”3”	Changes mode to password or textarea
TextMode=”Password/MultiLine”	

Example:

```
<asp:textbox id=”txt” runat=”server” columns=”30”/>
<asp:textbox id=”txtpwd” runat=”server” columns=”30” TextMode=”Password”/>
<asp:textbox id=”txtarea” runat=”server” columns=”30” Rows=”3” TextMode=”Multiline”/>
```

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 AdvertisementFile and 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 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 you ability to describe a document by defining meaningful tags for your application.

XML is not a language in itself, like HTML but, a set of rules for creating new markup languages. It is a meta-markup language. It allows developers to create custom tag sets for special uses. It structures, stores and transport information.

Following is an example of XML file:

```
<BOOK>
<NAME> Learn XML </NAME>
<AUTHOR> Samuel Peterson </AUTHOR>
<PUBLISHER> NSS Publications </PUBLISHER>
<PRICE> $30.00</PRICE>
</BOOK>
```

Like all XML files, the advertisement file needs to be a structured text file with well-defined tags delineating the data. 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 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.ediblеblooms.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;

	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; 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; 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 the 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>
```

The ads.xml file and the image files should be located in the root directory of the web site.

Try to run the above application an dobserve that each time the page is reloaded, the ad is changed.

MULTIVIEW CONTROL

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 for a MultiView control is:

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

The syntax for a 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 the View and MultiView controls

Both the View and MultiView controls are derived from Control class and inherits 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 controls 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 for the above properties:

Properties	Description
NextViewCommandName	NextView
PreviousViewCommandName	PrevView
SwitchViewByIDCommandName	SwitchViewByID

SwitchViewByIndexCommandName	SwitchViewByIndex

The following are the important methods of the MultiView control:

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, multi view 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 is as follows:

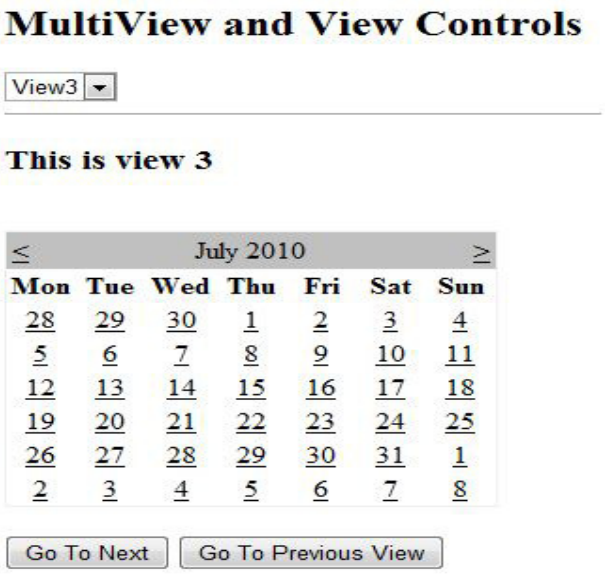
```
<%@ Page Language="C#"
        AutoEventWireup="true"
        CodeBehind="Default.aspx.cs"
        Inherits="multiviewdemo._Default" %>

<!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>
```

Observe the following:

The MultiView.ActiveViewIndex determines which view will be shown. This is the only view rendered on the page. The default value for the ActiveViewIndex is -1, when no view is shown. Since the ActiveViewIndex is defined as 2 in the example, it shows the third view when executed.



WIZARD CONTROL

The wizard - a standard user interface element in desktop applications - takes the user through a series of discrete steps in order to accomplish some task. A wizard step typically includes instructions, input controls, and an interface for moving between the wizard's steps (typically Next and Previous buttons, with a Finish button at the last step). Furthermore, wizards often include different steps depending on the inputs chosen in previous steps.

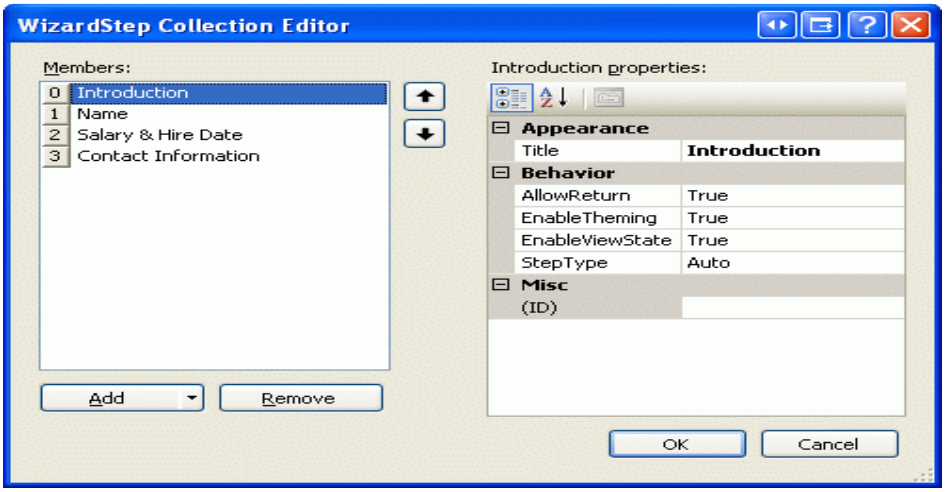
ASP.NET 2.0 makes creating wizard interfaces a lot less work thanks to its new Wizard control. With the Wizard control, we can define a series of Wizard steps and specify the content - static HTML and Web controls - that belongs in each step along with the function of the step, whether it's the first step, one step in the series of steps, the final step, or a summary step to appear after the wizard has completed. The Wizard control automatically includes the appropriate navigation elements for each step, remembers the values entered into the Web controls in each step, and includes a rich event model from which programmatic logic can be added to perform the desired task upon finishing the wizard (among other tasks).

Specifying the Wizard Control's Steps

The Wizard control is useful for breaking down a complex task into a sequence of simpler steps. Each step in the Wizard control can include the following information:

- **A Title** - along with each step, the Wizard control can optionally include a side bar that lists the available steps in the wizard and allows the user to quickly jump from one step to another. The title, if provided, is what appears in the side bar for the step.
- **A StepType** - the navigation controls displayed in each step depend upon the value of the step's StepType property. This property can be assigned one of the following values:
    - Start - the Start step includes just a Next button
    - Step - Step steps include both a Next and Previous button
    - Finish - the Finish step includes Next, Previous, and Finish buttons
    - Complete - the Complete step, if provided, displays a summary of the task performed *after* the Finish button has been clicked from the Finish step.
    - Auto - (the default) the navigation controls rendered are automatically determined based on the index of the step itself. That is, the first step in the wizard is assumed to be the Start step, the last step the Finish step, and all other steps are considered Step steps; no step is automatically considered a Complete step (if you want a Complete step, it must be explicitly specified).
- **Content** - each step can be composed of static HTML and Web controls. Often, steps will include Web controls for collecting user input, which is then used to perform the desired task after the Finish button has been clicked.

When you add a Wizard to an ASP.NET page you can specify the steps through the Designer or directly through the declarative syntax. From the Designer, click on the Wizard control, then go to the Properties window and scroll down to the WizardSteps property. Clicking on that property will bring up the WizardStep Collection Editor, from which you can add or remove steps, reorder them, and alter their properties.



To add content to a step, select the step from the Wizard's smart tag and then simply type in the text or add the Web controls from the Toolbox that you want to appear in the step.

The steps can also be examined and modified through the declarative syntax. The Wizard control specifies its steps through the <WizardSteps> element, with each step implemented as an <asp:WizardStep> or <asp:TemplatedWizardStep>. The Title and StepType properties can be specified through attributes in the <asp:WizardStep> or <asp:TemplatedWizardStep> tags; the actual content for the step is placed within the tags.

```
<asp:Wizard ID="AddEmployeeWizard" runat="server" CellPadding="5"
Width="95%">
  <WizardSteps>
    <asp:WizardStep runat="server" Title="Step 1" StepType="Start">
      ... Content for Step ...
    </asp:WizardStep>
    <asp:WizardStep runat="server" Title="Step 2" StepType="Step">
      ... Content for Step ...
    </asp:WizardStep>
    ...
    <asp:WizardStep runat="server" Title="Step 3" StepType="Finish">
      ... Content for Step ...
    </asp:WizardStep>
  </WizardSteps>
</asp:Wizard>
```

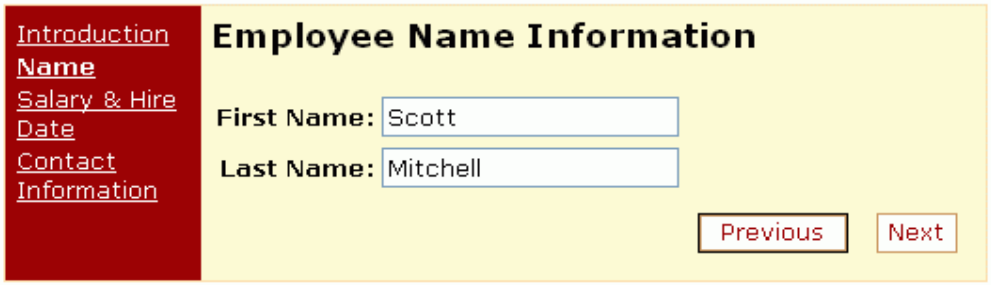
Creating a Wizard for Inserting a New Record

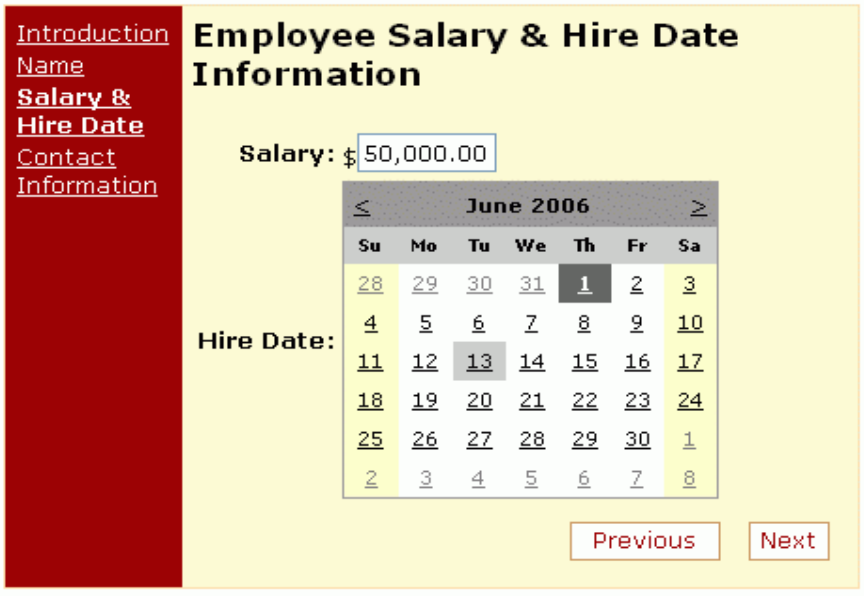
To illustrate how to use the Wizard control, I've whipped up a demo (which can be downloaded from the end of this article) that provides a wizard for collecting information that is, at wizard's completion, inserted into a database. In particular, this demo inserts data into an Access database that has an Employees table; this table has the following schema:

Employees		
Column	**Data Type**	**Comments**
EmployeeID	AutoNumber	Primary Key / Auto-Increment ID value
FirstName	Text	Required
LastName	Text	Required
Salary	Currency	Required
HireDate	Date/Time	Required

Phone	Text	Optional
Street	Text	Optional
City	Text	Optional
Email	Text	Optional

Rather than having one screen that prompts the user for all of these inputs at once, let's instead have a Wizard control where each step asks for a subset of the input values. The following screen shots illustrate this goal:





To create such an interface we need a wizard with at least three steps (one for the name information, one for salary and hire date, and one for the contact information). In addition to these three steps, let's also add an "Introduction" step that includes instructions. In the "Introduction" step, I have just text (and HTML) explaining the task about to be performed. In the other three steps, I include Web controls for collecting user input - a Calendar control for the HireDate field and TextBoxes for the other seven fields.

Let's look at the declarative syntax for this Wizard control. Since it's rather lengthy, let's first just focus on the overall structure and then dive into each step's content:

```
<asp:Wizard ID="AddEmployeeWizard" runat="server" Width="95%">
  <WizardSteps>
    <asp:WizardStep runat="server" Title="Introduction">
      ...
    </asp:WizardStep>
    <asp:WizardStep runat="server" Title="Name">
      ...
    </asp:WizardStep>
    <asp:WizardStep runat="server" Title="Salary & Hire Date">
      ...
    </asp:WizardStep>
    <asp:WizardStep runat="server" Title="Contact Information">
      ...
    </asp:WizardStep>
  </WizardSteps>
</asp:Wizard>
```

I've omitted virtually all style-related markup for brevity. However, I did leave in the Width="95%" setting in the <asp:Wizard> tag. If you don't set the Width for the entire Wizard control each step's width will be adjusted based on the content in the step. Therefore, you may want to set the Width of the Wizard control to some explicit value, either in pixels or as a percentage of the screen width; similarly, you may want to also set the Height property, if you also want a fixed height.

Note that for each <asp:WizardStep> I've omitted the StepType property. If you omit this property, the StepType is determined by the ordinal index of the step. Therefore, the

"Introduction" step is made a Start step, the "Contact Information" step a Finished step, and the "Name" and "Salary & Hire Date" steps are made Step steps.

Let's look at the "Name" step's content. Here, I've added two TextBoxes (and an HTML <table> for positioning purposes) to collect the new employee's FirstName and LastName values.

```
<asp:WizardStep runat="server" Title="Name">
   <h3>Employee Name Information</h3>
   <table border="0">
      <tr>
         <td class="InputLabel">First Name:</td>
         <td class="InputControl">
            <asp:TextBox ID="FirstName" runat="server" />
         </td>
      </tr>
      <tr>
         <td class="InputLabel">Last Name:</td>
         <td class="InputControl">
            <asp:TextBox ID="LastName" runat="server"  />
         </td>
      </tr>
   </table>
</asp:WizardStep>
```

The other steps include the appropriate markup and Web control syntax to collect the needed input. Note that for each step we do **not** need to specify the navigation controls. The appropriate Start, Next, Previous, and Finish buttons are automatically added based on the step's StepType setting.

Accessing the Content Defined in a Wizard Step

Eventually, after the Finish button has been clicked, we'll need to process the user's input and perform the desired task (in this case, inserting a record into the Employees table). At this point - or perhaps earlier in the Wizard life cycle - we may want to programmatically access the values entered by the user into the user controls. This can be accomplished by referencing the control's directly via their ID values in the code behind class, just like you would had the controls been defined outside of the Wizard control.

In order to insert a record when the finished button is clicked, we'll need to do one of two things:

- Write ADO.NET code that connects to the database and issues the appropriate INSERT statement, passing in the values entered by the user in the Web controls
- Add an AccessDataSource or SqlDataSource control to the page and specify its InsertCommand property. We can then use a series of <asp:ControlParameter> values in the InsertParameters collection to read the values entered by the user and use those

values in the actual INSERT statement. With this approach the only code we have to write is one line - we must invoke the data source control's Insert() method once the Finish button is clicked.

Let's use the second option (see the Accessing and Updating Data in ASP.NET 2.0 article series for more information on working with data source controls). The one subtlety here is that the <asp:ControlParameter> cannot see the controls in the <asp:WizardStep> *unless* it's placed within one of the <asp:WizardStep>s. I've put this control in the final step, along with the markup and Web controls for collecting the user's contact information:

```
<asp:WizardStep runat="server" Title="Contact Information">
  <h3>Employee Contact Information</h3>
  <table border="0">
    <tr>
      <td class="InputLabel">Street:</td>
      <td class="InputControl">
        <asp:TextBox ID="Street" runat="server" MaxLength="100"
Columns="25"></asp:TextBox>
      </td>
    </tr>
    <tr>
      <td class="InputLabel">City:</td>
      <td class="InputControl">
        <asp:TextBox ID="City" runat="server" MaxLength="25"
Columns="15"></asp:TextBox>
      </td>
    </tr>
    <tr>
      <td class="InputLabel">Phone:</td>
      <td class="InputControl">
        <asp:TextBox ID="Phone" runat="server" MaxLength="25"
Columns="12"></asp:TextBox>
      </td>
    </tr>
    <tr>
      <td class="InputLabel">Email:</td>
      <td class="InputControl">
        <asp:TextBox ID="Email" runat="server" MaxLength="100"
Columns="25"></asp:TextBox>
      </td>
    </tr>
  </table>
```

```
  <asp:AccessDataSource ID="InsertEmployeeDataSource" runat="server"
DataFile="~/App_Data/Employees.mdb"
      InsertCommand="INSERT INTO [Employees] ([FirstName], [LastName], [Salary],
[HireDate], [Phone], [Street], [City], [Email]) VALUES (?, ?, ?, ?, ?, ?, ?, ?)">
      <InsertParameters>
        <asp:ControlParameter Name="FirstName" Type="String"
ControlID="FirstName" PropertyName="Text" />
        <asp:ControlParameter Name="LastName" Type="String"
ControlID="LastName" PropertyName="Text" />
        <asp:ControlParameter Name="Salary" Type="Decimal" ControlID="Salary"
PropertyName="Text" />
        <asp:ControlParameter Name="HireDate" Type="DateTime"
ControlID="HireDate" PropertyName="SelectedDate" />
        <asp:ControlParameter Name="Phone" Type="String" ControlID="Phone"
PropertyName="Text" />
        <asp:ControlParameter Name="Street" Type="String" ControlID="Street"
PropertyName="Text" />
        <asp:ControlParameter Name="City" Type="String" ControlID="City"
PropertyName="Text" />
        <asp:ControlParameter Name="Email" Type="String" ControlID="Email"
PropertyName="Text" />
      </InsertParameters>
  </asp:AccessDataSource>
</asp:WizardStep>
```

The Wizard Control Event Model

When the user moves from one wizard step to another, the Wizard control raises one or more events. The ActiveStepChanged event fires anytime one step is about to transition to another. The NextButtonClick and PreviousButtonClick events fire when the user clicks the Next or Previous button, respectively. The FinishButtonClick event is raised when the Finish button is clicked. In this demo we'll only examine one of these events, FinishButtonClick. We'll create an event handler for this event to insert the employee record. The other events are useful for performing server-side validation of user input or implementing a non-linear workflow (for example, if, after clicking Next, the Next step depends on the user's input in this step or a previous one).

The event handler for the FinishButtonClick event for our demo is painfully simply - we just need to invoke the AccessDataSource control's Insert() method. At that point, the AccessDataSource control will grab the values from the specified controls and issue the INSERT statement. After inserting the data, the event handler redirects the user back to the site's homepage.

```
'This event handler fires when the user clicks Finish. We need to insert the new
employee record into the database
Protected Sub AddEmployeeWizard_FinishButtonClick(ByVal sender As Object, ByVal
e As System.Web.UI.WebControls.WizardNavigationEventArgs) Handles
AddEmployeeWizard.FinishButtonClick
    'Insert the employee into the database
    InsertEmployeeDataSource.Insert()

    'Send the user back to the homepage
    Response.Redirect("~/Default.aspx")
End Sub
```

IMAGEMAP CONTROL IN ASP.NET

ImageMap control is used to create an image that contains clickable hotspot region. When user click on the region, the user is either sent to a URL or a sub program is called Imagemap. When it is rendered on the page, it is implemented through <img /> HTML tag.

Hot spots features:

- There is no limit on number of hotspots each image may contain.
- Each hotspot is characterized by various attributes like shape, location and size.
- Overlapping hotspots are perfectly valid.
- Hot spots are defined using x and y coordinates.
- Hot spots can be assigned Image URL's and are capable of postback.

Different types of hot spots:

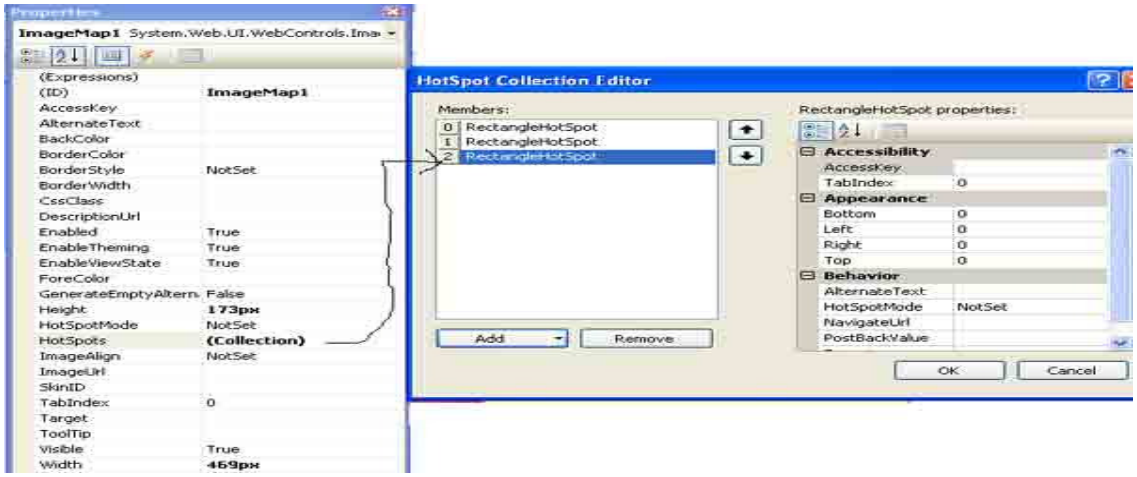
There are three different types of hot spots offered by ImageMap control. They are:

- CircleHotspot
- RectangleHotspot
- PolygonHotspot

CircleHotspot: CircleHotspot defines circle shaped hot spot region in an ImageMap control. To define the region for a circle hot spot, we should define X and Y coordinates for circle as well as radius property which usually is the distance from the center of circle to the edge.

RectangleHotspot: RectangleHotspot defines rectangle shaped hot spot region in an ImageMap control. To define the region for a Rectangle hot spot, we define Top, Bottom, Left and Right coordinates. Similar is the case for the Polygon hot spot.

Rightclick On Imagemap Control :



Example:

if you click the left side of the image you see below image

if you click the Right side of the image you see below image

MASTER PAGES

Master pages provide templates for other pages on your web site.

Master pages allow you to create a consistent look and behavior for all the pages (or group of pages) in your web application.

A master page provides a template for other pages, with shared layout and functionality. The master page defines placeholders for the content, which can be overridden by content pages. The output result is a combination of the master page and the content page.

The content pages contain the content you want to display.

When users request the content page, ASP.NET merges the pages to produce output that combines the layout of the master page with the content of the content page.

Master Page Example

```
<%@ Master %>

<html>
<body>
<h1>Standard Header From Masterpage</h1>
<asp:ContentPlaceHolder id="CPH1" runat="server">
</asp:ContentPlaceHolder>
</body>
</html>
```

The master page above is a normal HTML page designed as a template for other pages.

The @ **Master** directive defines it as a master page.

The master page contains a placeholder tag **<asp:ContentPlaceHolder>** for individual content.

The **id="CPH1"** attribute identifies the placeholder, allowing many placeholders in the same master page.

This master page was saved with the name **"master1.master".**

Note: The master page can also contain code, allowing dynamic content.

Content Page Example

```
<%@ Page MasterPageFile="master1.master" %>

<asp:Content ContentPlaceHolderId="CPH1" runat="server">
  <h2>Individual Content</h2>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</asp:Content>
```

The content page above is one of the individual content pages of the web.

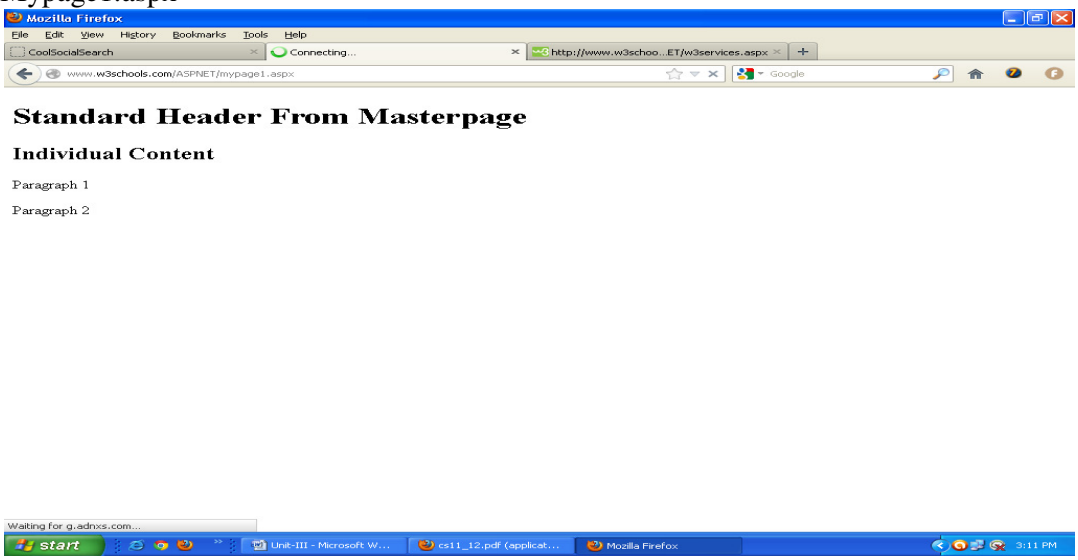
The @ **Page** directive defines it as a standard content page.

The content page contains a content tag **<asp:Content>** with a reference to the master page (ContentPlaceHolderId="CPH1").

This content page was saved with the name **"mypage1.aspx"**.

When the user requests this page, ASP.NET merges the content page with the master page.

Mypage1.aspx



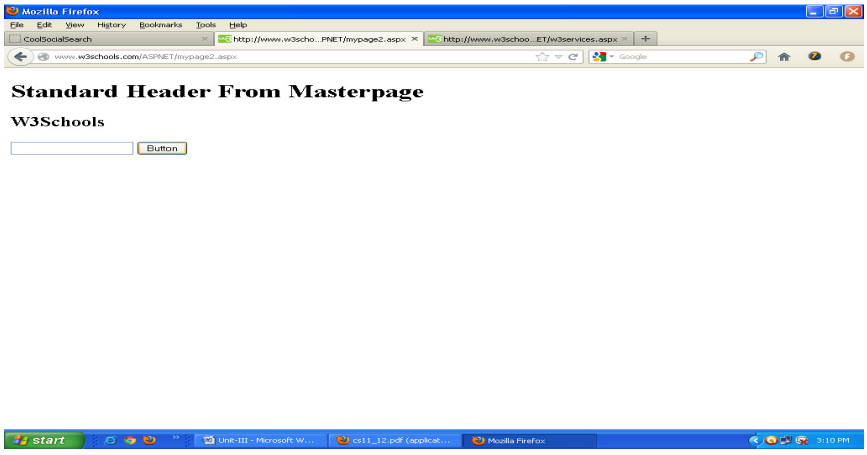
Note: The content text must be inside the <asp:Content> tag. No content is allowed outside the tag.

Content Page With Controls

```
<%@ Page MasterPageFile="master1.master" %>

<asp:Content ContentPlaceHolderId="CPH1" runat="server">
  <h2>W3Schools</h2>
  <form runat="server">
    <asp:TextBox id="textbox1" runat="server" />
    <asp:Button id="button1" runat="server" text="Button" />
  </form>
</asp:Content>
```

The content page above demonstrates how .NET controls can be inserted into the content page just like an into an ordinary page.

Mypage2.aspx



SITE NAVIGATION

You can use ASP.NET site-navigation features to provide a consistent way for users to navigate your site. As your site grows, and as you move pages around in the site, it can become difficult to manage all of the links. ASP.NET site navigation enables you to store links to all of your pages in a central location, and render those links in lists or navigation menus on each page by including a specific Web server control.

To create a consistent, easily managed navigation solution for your site, you can use ASP.NET site navigation. ASP.NET site navigation offers the following features:

- Site maps You can use a site map to describe the logical structure of your site. You can then manage page navigation by modifying the site map as pages are added or removed, instead of modifying hyperlinks in all of your Web pages.
- ASP.NET controls You can use ASP.NET controls to display navigation menus on your Web pages. The navigation menus are based on the site map.

- Programmatic control You can work with ASP.NET site navigation in code to create custom navigation controls or to modify the location of information that is displayed in a navigation menu.
- Access rules You can configure access rules that display or hide a link in your navigation menu.
- Custom site-map providers You can create custom site-map providers that allow you to work with your own site-map back end (for example, a database where you store link information) and plug your provider into the ASP.NET site-navigation system.
- With ASP.NET site navigation, you describe the layout of your site as a hierarchy. For example, a fictional online computer store might have a site comprised of eight pages, which are laid out in the following manner.

```
Home
  Products
    Hardware
    Software
  Services
    Training
    Consulting
    Support
```

- To use site navigation, start by creating a site map, or a representation of your site. You can describe the hierarchy of your site in an XML file, but other options are also available. For more information and an example, see ASP.NET Site Maps.
- After you create a site map, you can display the navigation structure on an ASP.NET page by using a site-navigation control.

Site-Map Load Process

The default ASP.NET site-map provider loads site-map data as an XML document and caches it as static data when the application starts. An excessively large site-map file can use a lot of memory and CPU power at load time. The ASP.NET site-navigation features depend on file notifications to keep navigation data up-to-date. When a site-map file is changed, ASP.NET reloads the site-map data. Make sure you configure any virus-scanning software so that it does not modify site-map files.

Site Navigation Controls

Creating a site map that reflects the structure of your site is one part of the ASP.NET site-navigation system. The other part is to display your navigation structure in your ASP.NET Web pages so that users can move easily around your site. You can easily build navigation into your pages by using the following ASP.NET site-navigation controls:

The SiteMapPath control displays a navigation path, which is also known as a breadcrumb, or eyebrow, that shows the current page location and displays links as a path back to the home page.

Note:

If an .aspx page contains a SiteMapPath control, the .aspx page must be listed in the Web.sitemap file for the control to render.

On a Web page, the SiteMapPath control displays something like the following if the user is browsing the Training page:

Home > Services > Training

The TreeView control displays a tree structure that users can traverse for links to different pages in your site. A node that contains child nodes can be expanded or collapsed by clicking it. When it is first rendered, the TreeView control is fully expanded. On a Web page, the TreeView control displays something like the following:

- Home

  - Services

    Training

    Consulting

The Menu control displays an expandable menu that users can traverse for links to different pages in your site. A node that contains child nodes is expanded when the cursor hovers over the menu item. For a code example that displays a site map in a Menu control

To use these site-navigation controls, you must describe the structure of your Web site in a Web.sitemap file.

To create a Web.sitemap file

1. Create a file in the root directory of your Web site called Web.sitemap.
2. Open the Web.sitemap file and add the following code.
3. <?xml version="1.0" encoding="utf-8" ?>
4. <siteMap>
5. <siteMapNode title="Home" >
6. <siteMapNode title="Services" >
7. <siteMapNode title="Training" url="~/Training.aspx"/>
8. <siteMapNode title="Consulting" url=""/>
9. </siteMapNode>
10. </siteMapNode>
11. </siteMap>
12. Later in this topic, you will create the Training.aspx page.
13. Save your file and then close it.

To add site navigation to a Web page

1. Create a file in the root directory of your Web site called Training.aspx.
2. Open Training.aspx and add the following code.

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

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

<script runat="server">
</script>

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

  <h2>Using SiteMapPath</h2>
  <asp:SiteMapPath ID="SiteMapPath1" Runat="server">
  </asp:SiteMapPath>

  <asp:SiteMapDataSource ID="SiteMapDataSource1" Runat="server" />

  <h2>Using TreeView</h2>
  <asp:TreeView ID="TreeView1" Runat="Server"
DataSourceID="SiteMapDataSource1">
  </asp:TreeView>

  <h2>Using Menu</h2>
  <asp:Menu ID="Menu2" Runat="server" DataSourceID="SiteMapDataSource1">
  </asp:Menu>

  <h2>Using a Horizontal Menu</h2>
  <asp:Menu ID="Menu1" Runat="server" DataSourceID="SiteMapDataSource1"
    Orientation="Horizontal"
    StaticDisplayLevels="2" >
  </asp:Menu>
  </div>  </form></body></html>
```

3. Save and close the file, and then you can view your file in a browser to see how the controls display the navigation structure of your Web site.

WEB PARTS

Introduction

Web Parts is a framework built into ASP.NET 2.0 for building highly customizable portal-style pages. End users can customize Web Parts pages by changing the page layout, adding and removing Web Parts, editing Web Parts properties, establishing connections between Web Parts, and more. Changes made to a Web Parts page are persisted by the Web Parts framework.

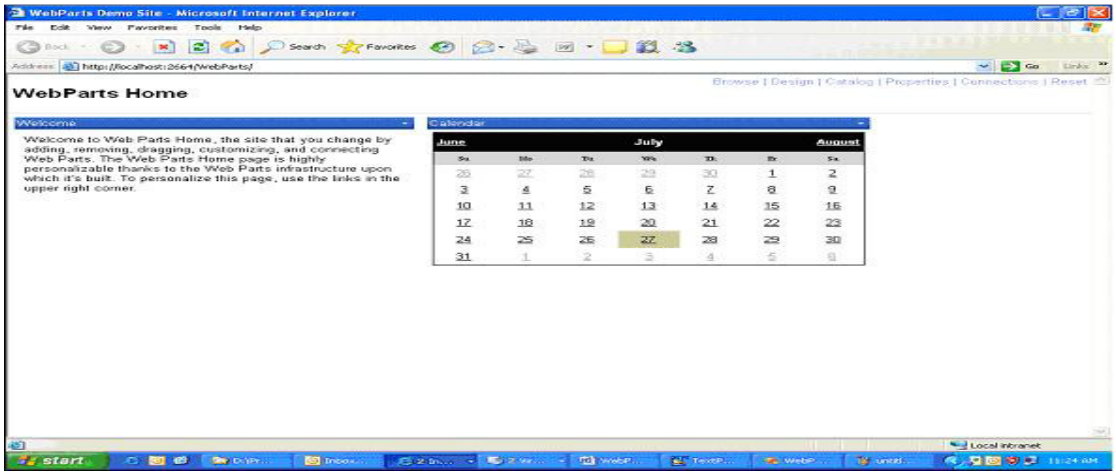
The Web Parts framework is often compared to Microsoft SharePoint Portal Server because both can be used to produce rich portal-style applications. The chief difference between them is that Web Parts is an application framework, whereas SharePoint Portal Server is an application featuring tools and components for building portal sites.

Building Web Parts

There are two basic ways to create a Web Part. You can treat any standard Microsoft ASP.NET control as a Web Part or you can build a custom control that derives from the base WebPart class.

You are not required to modify a control in any way to use it as a Web Part. Standard ASP.NET controls (such as the Calendar and GridView controls), Web User Controls, and even custom controls can all be used as Web Parts. When you treat a standard control as a Web part, the control is represented in the Web Part framework with the GenericWebPart class.

Alternatively, you can build a custom Web Part control by building a new control that derives from the base WebPart class. The advantage of this second method of creating a Web Part is that it provides you with additional options for customizing the appearance and behavior of the Web Part. Custom Web parts allow adding verbs to their verbs menus. Sample WebPart application is shown below:



Note: I have provided the source code for Web Parts application. Kindly download the source code and run the application to for web parts demonstration.

WebPart Controls

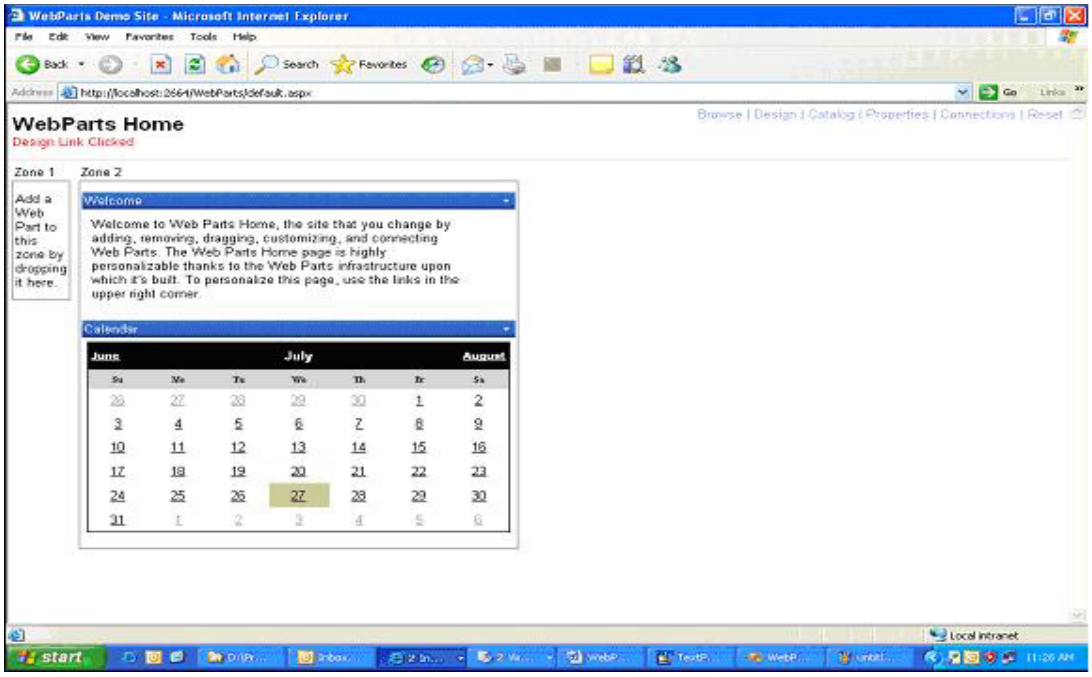
WebPartManager

WebPartManager is the most important of all the Web Part controls, responsible for managing and coordinating all controls inside WebPartZones. The Web Parts framework doesn't work without it, so every page that uses Web Parts must have an instance of WebPartManager declared, and it must be declared before other Web Parts controls. WebPartManager has no UI, so it's not visible on the page. It also exposes a very rich API for adding Web Parts to the page, closing Web Parts, connecting Web Parts, and more.

WebPartZone

WebPartZone is arguably the second most important Web Part control. It's used to define zones, which serve as containers for Web Parts. There is no practical limit to the number of WebPartZones a page can contain, and no limit to the number of Web Parts that a zone can contain. A WebPartZone control can host controls that do not derive from the WebPart class, by wrapping them with a GenericWebPart control at run time.

WebParts 'Welcome' and 'Calendar' are in WebPartZone 1 and WebPartZone 2 respectively in figure below:

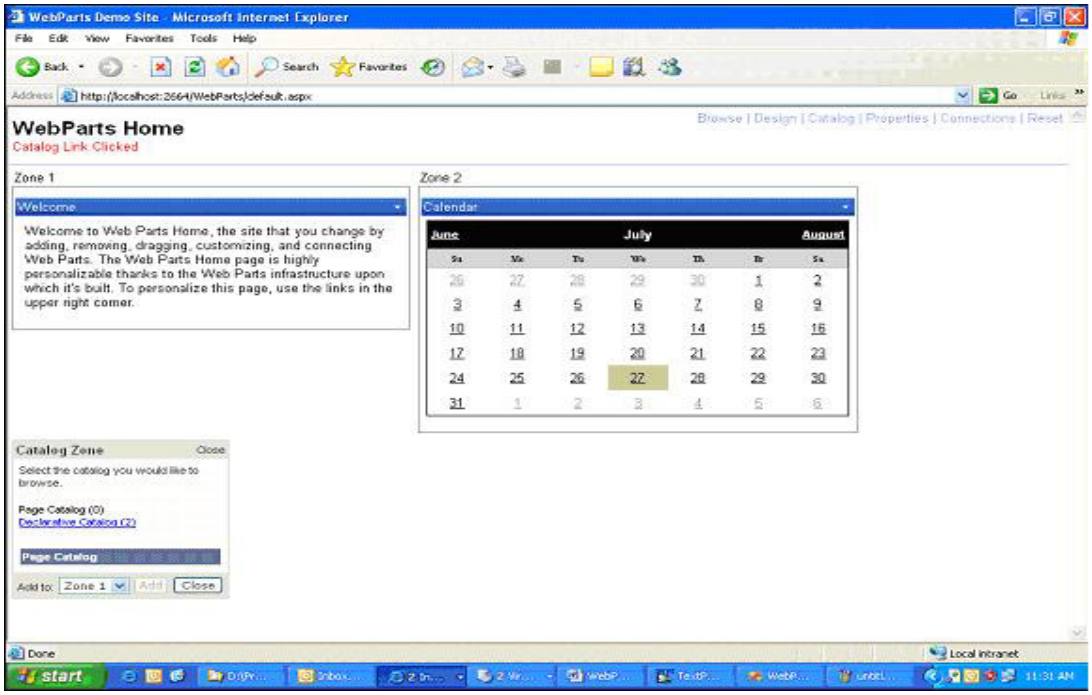


User can drag and drop web parts from one WebPartZone to another WebPartZone in Design mode.

Catalog Zone
One of the chief benefits to building pages from Web Parts is that the content of these pages can be interactively configured by the user. It's a simple matter, for example, to enable users to restore closed Web Parts to the page by including a CatalogZone in the page.

The purpose of the CatalogZone control is to allow end users to customize Web Parts pages by adding Web Parts to them. Web Parts can come from three sources: Web Parts that were previously present in the page but were closed, Web Parts that don't appear on the page by default but that can be added, and Web Parts imported from .WebPart files.

A CatalogZone control becomes visible only when a user switches a Web page to catalog display mode (CatalogDisplayMode) as shown below:



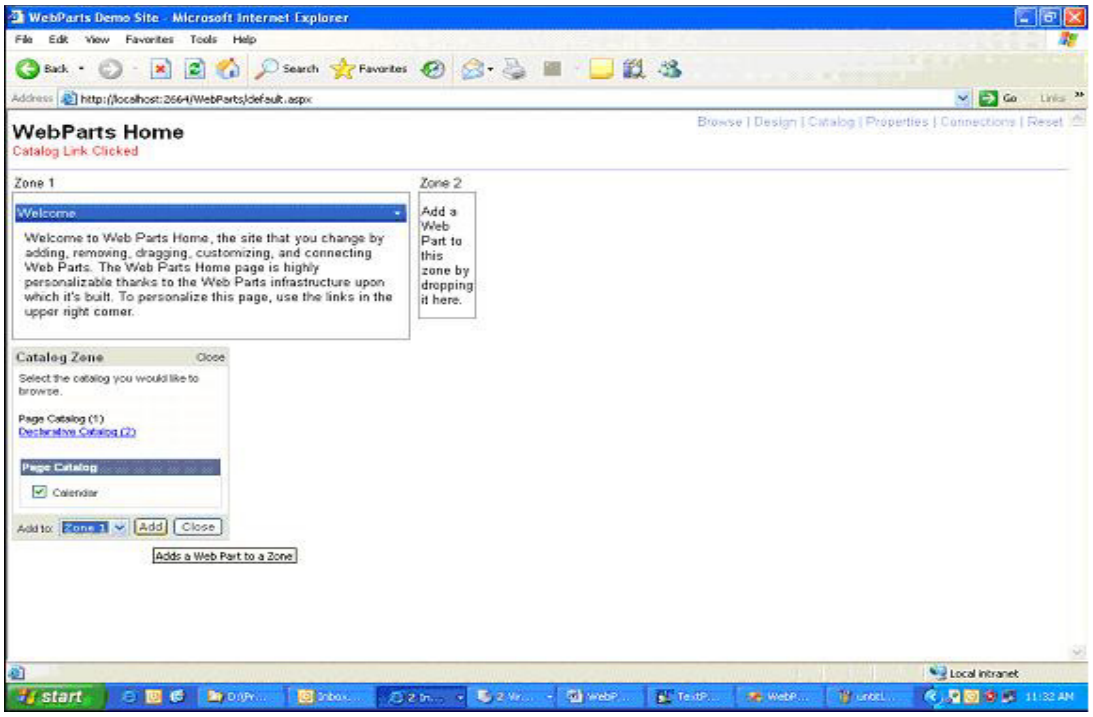
CatalogPart:
CatalogPart controls provide the UIs for adding Web Parts to the page. A CatalogZone can contain any combination of CatalogParts.

A CatalogZone can contain several types of CatalogPart controls. The following list summarizes the CatalogPart controls provided with the Web Parts control set:

- PageCatalogPart
- DeclarativeCatalogPart
- ImportCatalogPart

PageCatalogPart
The PageCatalogPart class serves one very specific purpose on a Web Parts page: it acts as a page catalog to contain any controls previously added to the page that a user has closed, and that the user can add back to the page. Add a PageCatalogPart control to your page if you want to provide users with the flexibility of closing and reopening controls. If your page does

not allow users to close controls at all, there is no need to add a PageCatalogPart control to your page.



DeclarativeCatalogPart

The DeclarativeCatalogPart control provides a way for developers to add a set of server controls declaratively to a catalog on a Web page. A catalog, in the Web Parts control set, is simply a list of WebPart or other server controls that is visible when a page is in catalog display mode. A user can select controls from the list and add them to the Web page, which effectively gives users the ability to change the set of controls and the functionality on a page, as shown below:

ImportCatalogPart

The ImportCatalogPart control enables users to import a description file that describes settings on a WebPart control or server control that a user wants to add to a Web page. After the user has imported the description file, the WebPart control referenced in the file appears within the ImportCatalogPart control when the page is in catalog mode, and a user can then

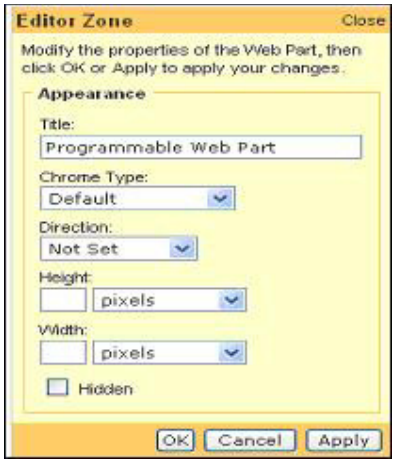
add the control to the page. User can view the ImportCatalogPart in Catalog Display mode, as shown below. User can browse the web part file and then upload by clicking Upload button.

Editor Zone

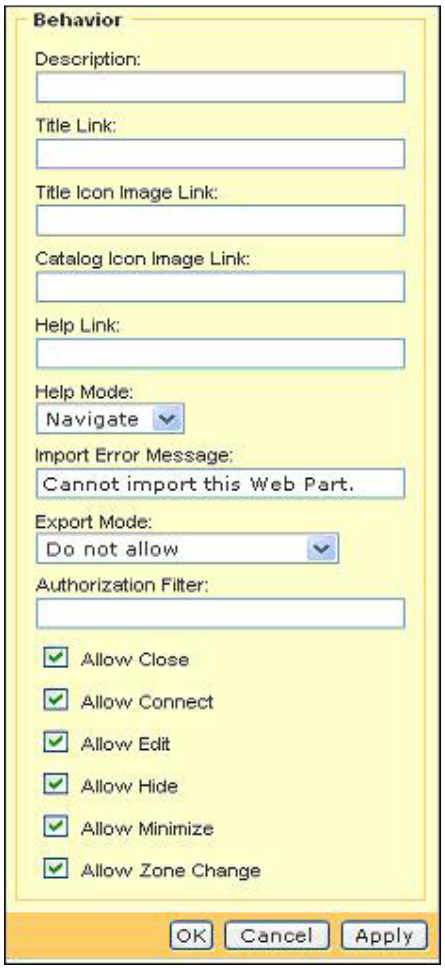
The purpose of the EditorZone control is to allow end users to customize Web Parts pages by editing the properties of the page's Web Parts. Editing UIs are provided by EditorPart controls, which divide Web Part properties into four categories:

Properties that affect appearance, Properties that affect behavior, Properties that affect layout and Custom properties added by Web Part developers.

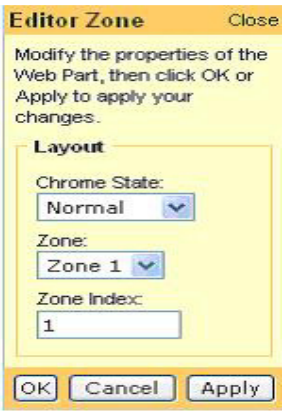
An EditorZone can contain any combination of EditorParts. EditorZones are only visible when the display mode is EditDisplayMode. User can click the Edit verb from webpart to open the Editor Zone.



Editor Zone with AppearanceEditorPart



Editor Zone with BehaviorEditorPart



Editor Zone with LayoutEditorPart

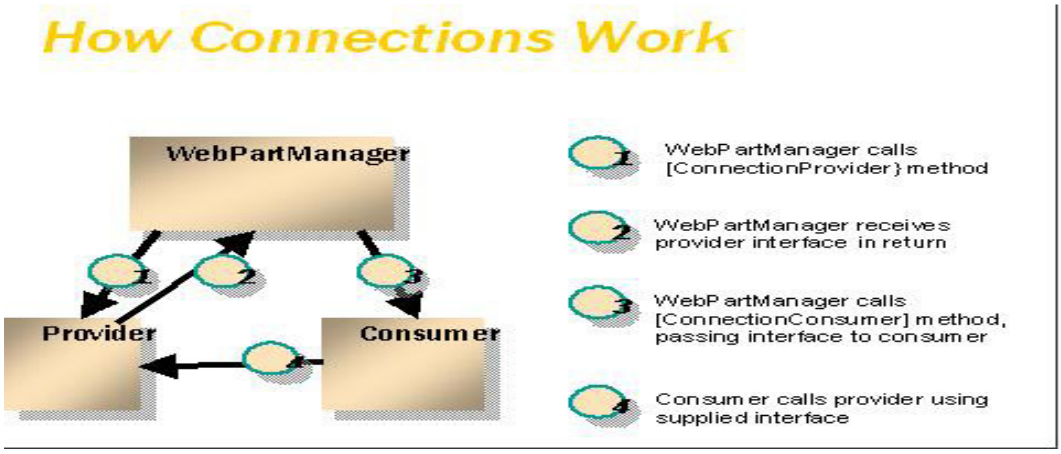
Web Parts can and often do implement custom properties to complement the built-in properties provided by the Web Parts framework. A Web Part that shows stock prices, for example, might implement a public property named "Stocks" to enable end users to specify which stock prices are shown. PropertyGridEditorParts provide UIs for editing custom properties. Attributing a property [WebBrowsable] enables that property to appear in a

PropertyGridEditorPart. Of course, the PropertyGridEditorPart must be declared in an EditorZone if it's to appear on the page.

WebPart Connection
Connections enable Web Parts to share data. A classic example is a Web Part control that shows the current weather and allows the user to enter a zip code so the weather can be localized. Another Web Part on that page--perhaps one that shows news headlines--might want that zip code so it, too, can localize content. Rather than require the user to enter the zip code twice, you can connect the two Web Parts so that one can get the zip code from the other. Connections can be defined statically by page developers, or they can be created dynamically by end users. The ConnectionsZone control provides a UI for creating connections dynamically.

Connection Provider
Writing a connection provider is no more difficult than writing a method that returns an interface reference and attributing that method [ConnectionProvider]. The first parameter to [ConnectionProvider] assigns a friendly name to the provider connection point and is displayed by the ConnectionsZone UI. The second parameter assigns a unique ID to the provider connection point. A provider can implement multiple provider connection points if desired. Each provider connection point must be assigned a unique ID.

ConnectionConsumer
Writing a connection consumer is a matter of writing a method that receives an interface reference and attributing that method [ConnectionConsumer]. The first parameter to [ConnectionConsumer] assigns a friendly name to the consumer connection point and is displayed by the ConnectionsZone UI. The second parameter assigns a unique ID to the consumer connection point.



The mechanics underlying Web Part connections is simple. In each page request, the WebPartManager enumerates the connections that are defined. Then, for each connection, it calls the connection provider's [ConnectionProvider] method to retrieve an interface and passes the interface to the corresponding connection consumer by calling the consumer's [ConnectionConsumer] method. The connection consumer then makes calls through the supplied interface to fetch data from the connection provider, as shown in figure above.

Each <asp:Connection> element defines one connection. A connection is characterized by the

Web Parts at either end of the connection (the provider and the consumer) and by the connection point IDs on each Web Part.

ConnectionZone
You can allow users to dynamically create connections between Web Parts. The ConnectionZone control provides a UI for making connections. Note there are no "ConnectionsPart" controls as there are EditorParts and CatalogParts; simply declaring a ConnectionZone and setting the display mode to ConnectDisplayMode is sufficient to display a connection UI.

Customizations applied to Web Parts pages by users--for example, changes to the page layout and changes to Web Part properties--are persisted using the Web Parts personalization service. The service is provider-based, and it can be accessed programmatically using the methods and properties of the System.Web.UI.WebControls.WebParts.PersonalizationAdministration class. The personalization service also persists custom properties that are attributed [Personalizable].

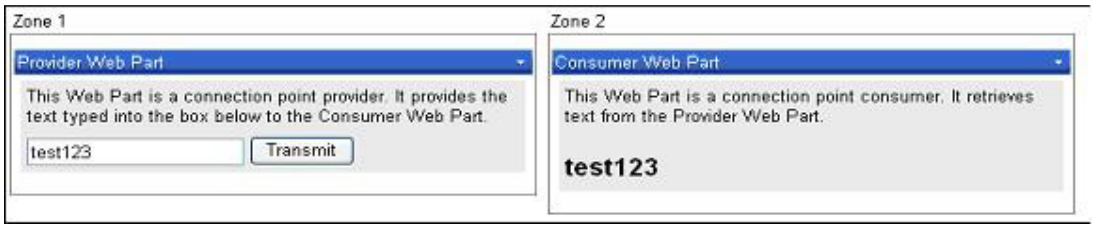
The fact that Web Parts personalization is provider-based adds a lot of flexibility to where Web Parts properties are persisted.

First add the 'Provider Web Part' to WebPartZone 1 and 'Consumer Web Part' to WebPartZone 2. Click the Connections link in browser window. Then click the Connect verb from Provider Web Part as shown in figure below.

Clicking on Connect verb displays the Connections Zone, as shown in figure below.

Click on 'Create a connection to a Consumer' in Connections Zone, select the web part to connect to (Consumer Web Part) and click Connect to create connection between Provider Web Part and Consumer Web Part.

Once connection is established, user can type text in Provider Web Part and click Transmit button to transmit the text to Consumer Web Part.



Export WebPart

You can export a Web Part to a .WebPart file by selecting the Export command from the Web Part's verbs menu. That command only appears in the verbs menu if the Web Part's ExportMode property is set to WebPartExportMode.All or WebPartExportMode.NonSensitiveData. A .WebPart file contains an XML representation of the Web Part's state at the time it was exported. Properties not attributed [Personalizable] aren't persisted to .WebPart files. In addition, if ExportMode=WebPartExportMode.NonSensitiveData, only "nonsensitive" properties attributed [Personalizable] are exported. You can use specify whether a property is sensitive in the [Personalizable] attribute. Examples of sensitive properties that perhaps should not be exported are passwords and social security numbers.

A Web Part exported to a .WebPart file can be imported using an ImportCatalogPart.