Information Passing, Standard Controls and Master Page

Passing Information from one page to another

Web Server Controls : Button, Image Button, Link Button, Textbox, Hyperlink, ImageMap control (Creating Hotspots),CheckBox and RadioButton, checkBoxList, RadioButtonList, ListBox, DropdownList

Rich Controls: Calendar, Adrotator control (showing advertisement from XML file and Database), FileUpload control

Grouping Controls : Panel, PlaceHolder

Using Navigation Controls : TreeView, SiteMapPath, Menu, Creating sitemap file for navigation

Designing Master page

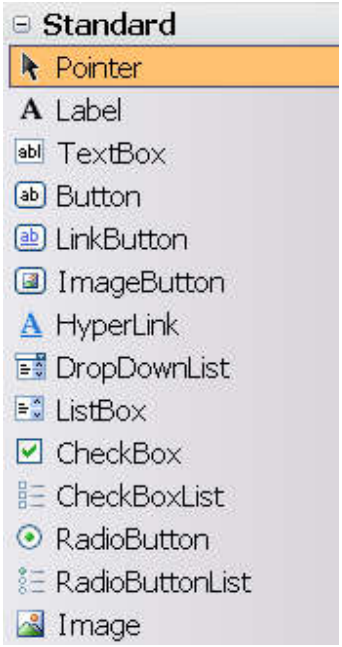
Using standard controls: [Control Properties]

Standard Controls— Enable you to render standard form elements such as buttons, input fields, and labels.

They are server side objects. They are programmable objects that act as user interfaces (UI) elements on a web page. The class of these controls is System.Web.UI.WebControls.

Syntax:

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



There are different types of standard controls available in ASP.NET 3.5, from that we refer some of that as below list:

1. Label
2. TextBox
3. Button
4. CheckBox
5. RadioButton
6. Link Button
7. Image Button
8. Hyperlink Control
9. DropDownList
10. List Box

Common/Public Properties of Visual Basic Controls

- Every object, such as a web page or control, has a set of properties that describe it. Although this set isn't identical for all objects, some properties (such as those listed in the table below) are common to most controls. You can see every design-time property for a given control by looking at the Properties window in the IDE (some controls have properties that are available only at run-time).

  The following table describes the properties inherited from the WebControl class:

Property	Description
AccessKey	Obtains the access key that allows you to quickly navigate to the web server control
Attributes	Obtains the collection of attributes applied to a control
BackColor	The background color of a control
BorderColor	The border color of a control
BorderStyle	The border style of a control
BorderWidth	The border width of a control
CssClass	The CSS class applied to a control
Enabled	A value indicating whether or not control is enabled
Font	The font attributes for the control
EnableTheming	Whether or not themes apply for a control
ForeColor	The foreground color of the control
Height	The height of the control
IsEnabled	A value indicating whether or not control is enabled (Get value only)
SkinID	The skin of the control which applied to the control
Style	The inline CSS style of the control
TabIndex	The tab order of the control
ToolTip	The text that appears when the user rests the mouse pointer over a control
Width	The width of the control

Common Methods of Visual Basic Controls

- Methods are blocks of code designed into a control that tells the control how to do things, such as set input focus to the control. Just as with properties, not all controls have the same methods, although some common methods do exist, as shown in the table below:

Method Name	Description
ApplyStyleSheetSkin	Applies the style properties defined in the page style sheet to the control.
CopyBaseAttributes	Copies the properties not encapsulated by the Style object from the specified Web server control to the Web server control that this method is called from. This method is used primarily by control developers.
DataBind()	Binds a data source to the invoked server control and all its child controls.
Dispose	Enables a server control to perform final clean up before it is released from memory.
FindControl(String)	Searches the current naming container for a server control with the specified *id* parameter.
Focus	Sets input focus to a control.
HasControls	Determines if the server control contains any child controls.
MergeStyle	Copies any nonblank elements of the specified style to the Web control, but will not overwrite any existing style elements of the control. This method is used primarily by control developers.
RenderBeginTag	Renders the HTML opening tag of the control to the specified writer. This method is used primarily by control developers.
RenderEndTag	Renders the HTML closing tag of the control into the specified writer. This method is used primarily by control developers.
ResolveClientUrl	Gets a URL that can be used by the browser.
ResolveUrl	Converts a URL into one that is usable on

	the requesting client.
SetRenderMethodDelegate	Infrastructure. Assigns an event handler delegate to render the server control and its content into its parent control.
ApplyStyleSheetSkin	Applies the style properties defined in the page style sheet to the control.

Common Events of Visual Basic Controls

- Events are what happen in and around your program. For example, when a user clicks a button, click_event ocuurs. Same as all controls have different events and common events.

Event Name	Description
DataBinding	Occurs when 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.
TextChanged	Occurs when the content of the text box changes between posts to the server.
Unload	Occurs when the server control is unloaded from memory.

1. Label:
Label controls are used to display text and cannot be edited by the user. They are used to identify objects on a form — provide a description of what a certain control will do if clicked, for example — or at run time, they can display information in response to an event or process in your application.

Figure- The label control

You can also write code that changes the text displayed by a label control in response to events at run time. For example, if your application takes a few minutes to process a change, you can display a processing-status message in a label.

→Properties:

Property	Description
AssociatedControlID	Gets or sets the identifier for a server control that the Label control is associated with.
Text	Gets or sets the text content of the Label control.

2. Textbox: The text box control is used to display information entered by the user at run time, or assigned to the Text property of the control at design or run time.

Figure of text box control

→Creating Multiline, Word-wrap Text Boxes:

By default, the TextMode property of the control is set to TextBoxMode.SingleLine, which displays a single-line text box. However, you can also use the TextBox control to display a multiline text box or a text box that masks user input by changing the value of theTextMode property to TextBoxMode.MultiLine or TextBoxMode.Password, respectively. The text displayed in the TextBox control is specified or determined by using theText property.

The TextBox control contains several properties that allow you to control the appearance of the control. The display width of the text box, in characters, is determined by its Columns property.

If the TextBox control is a multiline text box, the number of rows it displays is determined by the Rows property. To display text that wraps within theTextBox control, set the Wrap property to true.

→Creating a Password Text Box:

However, you can also use the TextBox control to display a password box or that masks user input by changing the value of theTextMode property to **TextBoxMode.Password**.

→Controlling Input in a Text Box:

You can also specify how data is entered in the TextBox control by setting a few properties. To prevent the text displayed in the control from being modified, set theReadOnly property to **true**. If you want to limit the user input to a specified number of characters, set the MaxLength property.

→Properties:

Property	Description
AutoCompleteType	Gets or sets a value that indicates the AutoComplete behavior of the TextBox control
AutoPostBack	A Boolean value that specifies whether the control is automatically posted back to the server when the contents change or not. Default is false
CausesValidation	Gets or sets a value indicating whether validation is performed when the TextBox control is set to validate when a postback occurs.
Columns	Gets or sets the display width of the text box in characters.
MaxLength	Gets or sets the maximum number of characters allowed in the text box.
ReadOnly	Gets or sets a value indicating whether the contents of the TextBox control can be changed.
Rows	Gets or sets the number of rows displayed in a multiline text box, i.e the height of the textbox (only used if TextMode="Multiline")
Text	The contents of the textbox
TextMode	Gets or sets the behavior mode (single-line, multiline, or password) of the TextBox control.
ValidationGroup	Gets or sets the group of controls for which the TextBox control causes validation when it posts back to the server.
Wrap	Gets or sets a value indicating whether the text content wraps within a multiline text box.

→Events:

Event	Means/Description
TextChanged	Occurs when the user changes the text of a Textbox.

3. Button:

- The Button control is used to create a button that sends a request to a web page. The Button control post data to the server when they are cliked.

- The Button control is used to display a push button. The push button may be a submit button or a command button. By default, this control is a submit button.
- A submit button does not have a command name and it posts the Web page back to the server when it is clicked. It is possible to write an event handler to control the actions performed when the submit button is clicked.
- A command button has a command name and allows you to create multiple Button controls on a page. It is possible to write an event handler to control the actions performed when the command button is clicked.

Figure of The button control

→ Properties:

Property	Description
CausesValidation	Specifies if a page is validated when a button is clicked
CommandArgument	Specifies additional information about the command to perform
CommandName	Specifies the command associated with the Command event
PostBackUrl	Specifies the URL of the page to post to from the current page when a button is clicked
Text	Specifies the text on a button
UseSubmitBehavior	Specifies whether or not a button uses the browser's submit mechanism or the ASP.NET postback mechanism
ValidationGroup	Specifies the group of controls a button causes validation, when it posts back to the server

→Events:

Event	Description
Click	Occurs when the Button control is clicked.
Command	Occurs when the Button control is clicked.

4. Checkbox:

The Checkbox control creates a check box that can be selected by clicking it. The Checkbox control is displays checkmarks that allow the user to toggle between a TRUE or FALSE condition.

Figure of The check box control

→ Properties:

Property	Description
AutoPostBack	Specifies whether the form should be posted immediately after the Checked property has changed or not. Default is false
CausesValidation	Specifies if a page is validated when a Button control is clicked
Checked	Specifies whether the check box is checked or not
InputAttributes	Attribute names and values used for the Input element for the CheckBox control
LabelAttributes	Attribute names and values used for the Label element for the CheckBox control
Text	The text next to the check box
TextAlign	On which side of the check box the text should appear (right or left)
ValidationGroup	Specifies the group of controls a check box causes validation, when it posts back to the server

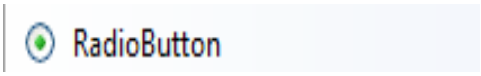
→Events:

Event	Description
CheckedChanged	Occurs when the value of the Checked property changes between various posts to the server.

5. Radio Button:

Similar to the Checkbox control, a Radio Button control creates a Single radio button. The RadioButton can be grouped, where only one RadioButton control can be selected at a time. In web, forms you have to set the Radiobutton's *GroupName* property to the same value to associate them into group.

Figure of the Radio Button control



→ Properties:

Property	Description
AutoPostBack	A Boolean value that specifies whether the form should be posted immediately after the Checked property has changed or not. Default is false
Checked	A Boolean value that specifies whether the radio button is checked or not
Id	A unique id for the control
GroupName	**The name of the group to which this radio button belongs**
Text	The text next to the radio button
TextAlign	On which side of the radio button the text should appear (right or left)

→Events:

Event	Description
CheckedChanged	Occurs when the value of the Checked property changes between various posts to the server.

6. Link Button:

The LinkButton control is used to link the current web page to some other web page, similar to Hyper Link control. The only difference between the wto is that the Hyperlink control just allows the browser to navigate to a new web page, whereas in the LinkButton control, we can also perform some other action by handling the Click and command events of this control

Figure of The link Button control

→ Properties:

Property	Description
CausesValidation	Specifies if a page is validated when a LinkButton

	control is clicked
CommandArgument	Additional information about the command to perform
CommandName	The command associated with the Command event
PostBackUrl	The URL of the page to post to from the current page when the LinkButton control is clicked
Text	The text on the LinkButton
ValidationGroup	Specifies the group of controls a link button causes validation, when it posts back to the server

→Events:

Event	Description
Click	Occurs when the Link Button control is clicked.
Command	Occurs when the Link Button control is clicked.

7. Image Button:

The ImageButton control is a button control that displays an image instead of text. The ImageButton control is specifically useful when you want to create image maps. Image maps are checkable regions on images and can initiative various actions depending on part of the image that you click.

Figure of the Image Button control

→ Properties:

Property	Description
CausesValidation	Specifies if a page is validated when an ImageButton control is clicked
CommandArgument	Additional information about the command to perform
CommandName	The command associated with the Command event
GenerateEmptyAlternateText	Specifies whether or not the control creates an empty string as an alternate text
PostBackUrl	The URL of the page to post to from the

	current page when the ImageButton control is clicked
ValidationGroup	Specifies the group of controls a link button causes validation, when it posts back to the server

→Events:

Event	Description
Click	Occurs when the Image Button control is clicked.
Command	Occurs when the Image Button control is clicked.

8. Hyper Link:

The HyperLink Control is used to create a link to another web page that can be a page in your web application or anywhere else on the World Wide Web (WWW).

You can specify the location of the linked page by specifying its URL on the current page. You can use text as well as an image in the Hyperlink control. Text is specified with the *Text property* and image specified by the *Imageurl* property. By default, when you click a Hyperlink control, the hyperlinked appears in a new browser window.

You can set the *Target* property to the name of a window or frame as listed below table:

Property	Description
_blank	Displays the hyperlinked content in a new window without frames.
_parent	Displays the hyperlinked content in the immediate frameset parent.
_self	Displays the hyperlinked content in the frame with class.
_top	Displays the hyperlinked content in the full window without frames.

Figure of the HyperLink control

→ Properties:

Property	Description
ImageUrl	The URL of the image to display for the link
NavigateUrl	The target URL of the link
Target	The target frame of the URL
Text	The text to display for the link

→Events: As Common event of control

9. DropDownList:

The DropDownList control displays the list of data as a drop-down list from which you can make a single selection. You cannot select multiple items in this control because when you make a selection from the list, the list closes automatically.

Figure of the DropDownList control

→ Properties:

Property	Description
SelectedIndex	The index of a selected item
ValidationGroup	Specifies the group of controls a DropDownlist causes validation, when it posts back to the server

→Events:

Event	Description
CallingDataMethods	Occurs when data methods are being called. (Inherited from DataBoundControl.)
CreatingModelDataSource	Occurs when the ModelDataSource object is being created. (Inherited from DataBoundControl.)
SelectedIndexChanged	**Occurs when the selection from the list control changes between posts to the server. (Inherited from ListControl.)**

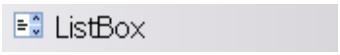
10. List Box:

The ListBox control is used to select one or more items from a list of items on a web page at runtime.

You can use Rows property to specify the height of the control. To enable multiple item selection, you can set the SelectionMode property to ListSelectionMode.Multiple.

You can find the selected item in a single-selection Listbox control with the help of SelectedItem and SelectedIndex properties. In multiple-selection Listbox controls, the loop runs over the selected items in the ListBox controls and returns each selected item as a ListItem object by using the SelectedIndex property.

Figure of the List box control



→ Properties:

Property	Description
Rows	The number of rows displayed in the list
SelectionMode	Allows single or multiple selections

→Events:

Event	Description
CallingDataMethods	Occurs when data methods are being called.
CreatingModelDataSource	Occurs when the ModelDataSource object is being created.
SelectedIndexChanged	**Occurs when the selection from the list control changes between posts to the server.**

→Adding Items to a List:

You can add items to a list box at either design time or at run time; the first item will have index 0, the next index 1, and so on in the list box. At design

time, you can use the **Items** property, which is a very handy array of the items in the list box, and at run time, you can use both the **Items** property and the **Add** (formerly **AddItem**) method.

Design time: you can add items directly to your list box by typing them into the **Items** property in the Properties window. Selecting the **Items** property displays the String Collection Editor,and you can type item after item into the list box that way.

Runtime: To add items to a list box, use the AddItem method, which has the following syntax:

Listboxname.**Item.add(** *item*)

Argument	**Description**
ListBoxname	Name of the list box.
Item	String expression to add to the list. If *item* is a literal constant, enclose it in quotation marks.

While list items are commonly added in the Form_Load event procedure, you can use the AddItem method at any time. This gives you the ability to add items to the list dynamically (in response to user actions).

The following code places "Germany," "India," "France," and "USA" into a list box named List1:

```
Form_Load ()
{
   List1.Item.Add( "Germany");
   List1.Item.Add( "India");
   List1.Item.Add("France");
   List1.Item.Add( "USA");
}
```

Whenever the form is loaded at run time, the list appears as shown in Figure.

Figure of "Countries" list box

→Adding an Item at a Specified Position:

To add an item to a list at a specific position, specify an index value for the new item. For example, the next line of code inserts "Japan" into the first position, adjusting the position of the other items downward:

List1.Item.Add("Japan", 0)

Notice that it is 0, not 1, that specifies the first item in a list see below figure.

→Referring to Items in a List Box by Index:

When you add items to a list box, each item is given an *index*, and you can refer to the item in the list box with this index by using the **Items** property, like this: **ListBox1.Items(5)**. The first item added to a list box gets the index 0, the next index 1, and so on. You also can get the index of an object in a list box with the **IndexOf** method, like this: **ListBox1.Items.IndexOf(Object5)**.

When the user selects an item in a list box, you can get the selected item's index with the list box's **SelectedIndex** (formerly **ListIndex**) property. You also can get the selected item's corresponding object with the **SelectedItem** property. Here's an example where I display the index of an item the user has selected in the **SelectedIndexChanged** event of a list box:

```
TextBox1.Text = "You selected item " & ListBox1.SelectedIndex + 1
```

→Removing Items from a List Box:

The Testing Department is calling again-how about letting the users customize your program? You ask: what do you mean? Well, they say, let's give the user some way of removing the 50 fine French cooking tips from the list box.

You can use the **RemoveAt** method to delete items from a list box. To remove the item at index 5, you'd use this code:

ListBox1.Items.RemoveAt(5)

Here's how you'd remove the currently selected item with **RemoveAt:**

ListBox1.Items.RemoveAt(ListBox1.SelectedIndex)

You also can use the **Remove** method to remove a specific object from a list box. Here's how I'd remove the currently selected item with **Remove:**

ListBox1.Items.Remove(ListBox1.SelectedItem)

You also can remove items by passing the corresponding object to **Remove**. For example, if I've filled a list box with **String** objects, I can remove the item "Item 1" this way:

ListBox1.Items.Remove ("Item 1")

You also can use the **Items.Clear** method to remove all items from the list box.

RadioButtonList

ASP.NET RadioButtonList control enable user to select an item from list. RadioButtonList support data bind programmatically from database. We can also populate it manually by input list item inside RadioButtonList tag. RadioButtonList is a single selection radio button group. RadioButtonList have an items collection. We can determine which item is selected by test it's SelectedItem property.

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

- RepeatColumns: It displays the number of columns of radio buttons. By default RepeatColumns value is zero.
- RepeatDirection: The direction that the radio buttons repeat. By default RepeatDirection value is vertical. Possible values are Horizontal and Vertical. When you click on the RepeatDirection button the layout will be changed as Horizontal.
- RepeatLayout: Determines whether the radio buttons display in an HTML table.
  Possible values are as follows:
  - Table
  - Flow
  - OrderedList
  - UnorderedList

```
<asp:RadioButtonList ID="RadioButtonList1" runat="server">
                        <asp:ListItem>ColdFusion</asp:ListItem>
                        <asp:ListItem>Asp.Net</asp:ListItem>
                        <asp:ListItem>PHP</asp:ListItem>
</asp:RadioButtonList>
```

To display selected item of radiobuttonlist in label control

```
protected void Button1_Click(object sender, EventArgs e)
   {
      Label1.Text = "You have selected </br> Item=" +
```

```
                    RadioButtonList1.SelectedItem.Text + "</br> Value =" +
                    RadioButtonList1.SelectedValue      + "</br> Index =" +
                    RadioButtonList1.SelectedIndex ;
  }
```

To change repeatdirection of radiobuttonlist:

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

ImageMap Control

The ImageMap control in ASP.NET 2.0 and onward versions can be used to create an image that contains defined hot spot regions. When a user clicks a hot spot region, the control can either generate a post back to the server or navigate to a specified URL.

There are three kinds of hot spot regions defined in ImageMap control.

- RectangleHotSpot
- CircleHotSpot
- PolygonHotSpot

The RectangleHotSpot defines rectangular hot spot regions. The CircleHotSpotdefines circle-shaped ones and the PolygonHotSpot is used for irregularly shaped hot spot area.

```
<asp:ImageMap ID="india" runat="server" ImageUrl="~/Map.jpg"
Height="500px" Width="887px" HotSpotMode="PostBack"
OnClick="india_Click" BackColor="#C0C000" BorderColor="DarkSlateGray"
ForeColor="Red">
                    <asp:CircleHotSpot Radius="7" X="215" Y="125"
HotSpotMode="PostBack" NavigateUrl="~/map.aspx" PostBackValue="bhuj"
AlternateText="Bhuj" />
                    <asp:CircleHotSpot Radius="7" X="525" Y="95"
HotSpotMode="PostBack" NavigateUrl="~/map.aspx" PostBackValue="patan"
/>
                    <asp:CircleHotSpot Radius="7" X="565" Y="45"
HotSpotMode="PostBack" NavigateUrl="~/map.aspx" PostBackValue="ambaji"
/>
                    <asp:CircleHotSpot Radius="7" X="525" Y="150"
HotSpotMode="PostBack" NavigateUrl="~/map.aspx" PostBackValue="me" />
                    <asp:CircleHotSpot Radius="7" X="625" Y="120"
HotSpotMode="PostBack" NavigateUrl="~/map.aspx" PostBackValue="him" />
                    <asp:CircleHotSpot Radius="7" X="495" Y="195"
HotSpotMode="PostBack" NavigateUrl="~/map.aspx" PostBackValue="sure" />
                    <asp:CircleHotSpot Radius="7" X="565" Y="195"
```

```
HotSpotMode="PostBack" NavigateUrl="~/map.aspx" PostBackValue="ahd" />
</asp:ImageMap>
                </td>
```

Events

Click	Occurs when a HotSpot object in an ImageMap control is clicked.
DataBinding	Occurs when 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

CheckBoxList

The CheckBoxList control provides a multi selection check box group that can be dynamically generated with data binding. It contains an Items collection with members corresponding to individual items in the list. To determine which items are checked, iterate through the collection and test the Selected property of each item in the list.

Properties

Item : Gets the collection of items in the list control.

RepeatColumns	Gets or sets the number of columns to display in the CheckBoxList control.
RepeatDirection	Gets or sets a value that indicates whether the control displays vertically or horizontally.
RepeatedItemCount	Gets the number of list items in the CheckBoxList control.
RepeatLayout	Gets or sets a value that specifies whether the list will be rendered by using a `table` element, a `ul` element, an `ol` element, or a `span` element.

RequiresDataBinding	Gets or sets a value indicating whether the DataBind() method should be called.
SelectArguments	Gets a DataSourceSelectArguments object that the data-bound control uses when retrieving data from a data source control.
SelectedIndex	Gets or sets the lowest ordinal index of the selected items in the list.
SelectedItem	Gets the selected item with the lowest index in the list control.
SelectedValue	Gets the value of the selected item in the list control, or selects the item in the list control that contains the specified value.
SelectMethod	The name of the method to call in order to read data.
Site	Gets information about the container that hosts the current control when rendered on a design surface.

Events

CallingDataMethods	Occurs when data methods are being called.
CreatingModelDataSource	Occurs when the ModelDataSource object is being created.
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.

SelectedIndexChanged	Occurs when the selection from the list control changes between posts to the server.
TextChanged	Occurs when the Text and SelectedValue properties change.
Unload	Occurs when the server control is unloaded from memory

Rich Controls

There are THREE type of grouping controls:
1) AdRotator
2) File Upload
3) Calendar

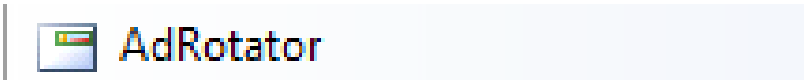
1) AdRotator:

ASP.NeT even supports banner ads. These ads, which all Internet users are familiar with, are image files in GIF, JPEG, or other formats, that the user can click to cause the browser to navigate to the advertiser's website.

Using an AdRotator, you can automatically cycle through a series of ad banners. The AdRotator automates the cycling process, changing the displayed ad when the page is refreshed. Note also that ads can be 'weighted' to control how often they appear compared with others, and if you prefer, you can also write custom logic that cycles through the data.

You use the AdRotator control to display an advertisement banner in a web page.

Figure of the AdRotator control



→ Properties:

Property	Description
AdvertisementFile	Specifies the path to the XML file that contains ad information
AlternateTextField	Specifies a data field to be used instead of the Alt text for an ad

ImageUrlField	Specifies a data field to be used instead of the ImageURL attribute for an ad
KeywordFilter	Specifies a filter to limit ads after categories
NavigateUrlField	Specifies a data field to be used instead of the NavigateUrl attribute for an ad
Target	Specifies where to open the URL

→Events:

Event	Description
AdCreated	It occurs before the web page is displayed, allowing you to customize ad displays.

This control uses an XML file to store the ad information. The XML file must begin and end with an <Advertisements> tag. Inside the <Advertisements> tag there may be several <Ad> tags which defines each ad.

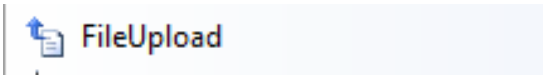
The predefined elements inside the <Ad> tag are listed below:

Element	Description
<ImageUrl>	Optional. The path to the image file
<NavigateUrl>	Optional. The URL to link to if the user clicks the ad
<AlternateText>	Optional. An alternate text for the image
<Keyword>	Optional. A category for the ad
<Impressions>	Optional. The display rates in percent of the hits

2) File Upload:

This control is intended to be used to upload the files from the client-side to the server-side. While working with the FileUplaod class one should always remember that it does not automatically save the files on the server-side. To save files on the server-side you need to use the SaveAs() method., you must ensure that ASP.NET has write permissions on the specified directory, otherwise the operation will be failed.

Figure of the Fileupload control



→ Properties:

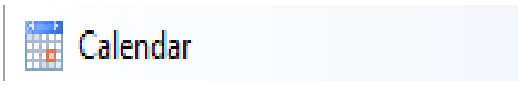
Property	Description
FileByte	It will return an array of bytes of the specified using a FileUpload control
FileContent	Obtains stream object that points to a file to upload using the FileUpload control.
FileName	Obtains the name of a file on a client computer to upload using the FileUpload control.
HasFile	Obtains a value indicating whether the FileUpload control contains a file or not.
PostedFile	Obtains the underlaying HttpPosteFile object for a file uploaded using the FileUpload control.

3) Calendar:

This control is used to display a single month of a calendar on a web page. This control allows you to select dates and move to the next or previous month. You can choose whether the Calender control allows users to select a single day, week or month by setting the SelectionMode property.

By default, the control displays the days of the month, day headings for the days of the week, a title with the month name and arrow characters for navigating to the next and previous month. You can customize the appearance of the calendar control by setting the properties that control the style for customize the appearance of the calendar control by setting the properties that controls the style for customize parts of the control.

Figure of the Calendar control



→ Properties:

Property	Description
Caption	The caption of the calendar
CaptionAlign	The alignment of the caption text

CellPadding	The space, in pixels, between the cell walls and contents
CellSpacing	The space, in pixels, between cells
DayHeaderStyle	The style for displaying the names of the days
DayNameFormat	The format for displaying the names of the days
DayStyle	The style for displaying days
FirstDayOfWeek	What should be the first day of week
NextMonthText	The text displayed for the next month link
NextPrevFormat	The format of the next and previous month links
NextPrevStyle	The style for displaying next and previous month links
OtherMonthDayStyle	The style for displaying days that are not in the current month
PrevMonthText	The text displayed for the previous month link
SelectedDate	The selected date
SelectedDates	The selected dates
SelectedDayStyle	The style for selected days
SelectionMode	How a user is allowed to select dates
SelectMonthText	The text displayed for the month selection link
SelectorStyle	The style for the month and weeks selection links
SelectWeekText	The text displayed for the week selection link
ShowDayHeader	A Boolean value that specifies whether the days of the week header should be shown
ShowGridLines	A Boolean value that specifies whether the grid lines between days should be shown
ShowNextPrevMonth	A Boolean value that specifies whether the next and previous month links should be shown
ShowTitle	A Boolean value that specifies whether the title of the calendar should be shown
TitleFormat	The format for the title of the calendar
TitleStyle	The style of the title of the calendar
TodayDayStyle	The style for today's date
TodaysDate	Today's date

UseAccessibleHeader	Specifying whether to use the <th> element for the day headers instead of the <td> element
VisibleDate	The date that specifies the month that is currently visible in the calendar
WeekendDayStyle	The style for weekends

→Events:

Event	Description
DayRender	The name of the function to be executed when when each day cell is created
SelectionChanged	The name of the function to be executed when the user selects a day, week, or month
VisibleMonthChanged	The name of the function to be executed when the user navigates to a different month

Grouping Controls

There are two type of grouping controls:
1) Panel
2) Place Holder

1) Panel:

The Panel control is one of the most frequently used controls. It creates a borderless division on the form; by using this division one can place the controls inside this control. These controls are useful when you want to show or hide a group of controls at once or when you want to add controls to a web page through code.

Figure of The panel control

→ Properties:

Property	Description
BackImageUrl	Specifies a URL to an image file to display as a background for this control

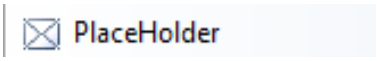
DefaultButton	Specifies the ID of the default button in the Panel
Direction	Specifies the content display direction of the Panel
GroupingText	Specifies the caption for the group of controls in the Panel
HorizontalAlign	Specifies the horizontal alignment of the content
ScrollBars	Specifies the position and visibility of scroll bars in the Panel
Wrap	Specifies whether the content should wrap or not

→Events: As Common Events of controls

2) Place Holder:

The Place Holder control is used as a container to store server controls that are added to the web page at runtime. It does not produce any visible output and is used only as a container for other controls on the web page.

Figure of the PlaceHolder control



→ Properties:

Property	Description
EnableTheming	Obtains or sets a value indicating whether themes apply to this control.

→Events: As Common Events of controls

Navigation Controls

SITE NAVIGATION AND SITE MAPS

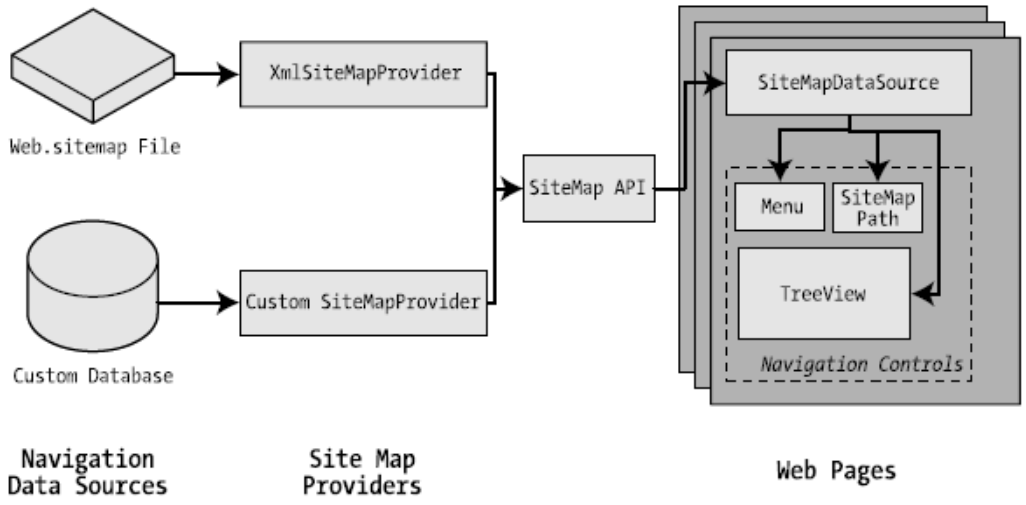
You've already learned simple ways to send a website visitor from one page to another. For example, you can add HTML links (or HyperLink controls) to your page to let users surf through your site. If you want to perform page navigation in response to another action, you can call the Response.Redirect() method or the Server.Transfer() method in your code. But in professional web applications, the navigation requirements are more intensive. These applications need a system that allows users to surf through a hierarchy of pages, without forcing you to write the same tedious navigation code in every page.

Site Maps

ASP.NET navigation is flexible, configurable, and pluggable. It consists of three components:

- A way to define the navigational structure of your website. This part is the XML site map, which is (by default) stored in a file.
- A convenient way to read the information in the site map file and convert it to an object model. The SiteMapDataSource control and the XmlSiteMapProvider perform this part.
- A way to use the site map information to display the user's current position and give the user the ability to easily move from one place to another. This part takes place through the navigation controls you bind to the SiteMapDataSource control, which can include breadcrumb links, lists, menus, and trees.

You can customize or extend each of these ingredients separately. For example, if you want to change the appearance of your navigation controls, you simply need to bind different controls to the SiteMapDataSource. On the other hand, if you want to read site map information from a different type of file or from a different location, you need to change your site map provider.
Figure shows how these pieces fit together.



Defining a Site Map

You can create it in Visual Studio by right on **Website ➢ Add New Item** and then choosing the **Site Map** option.

Rules for creating Site Maps:

Rule 1: Site Maps Begin with the <siteMap> Element
Every Web.sitemap file begins by declaring the <siteMap> element and ends by closing that element. You place the actual site map information between the start and end tags

<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
.....
…….
…….
</siteMap>

Rule 2: Each Page Is Represented by a <siteMapNode> Element

To insert a page into the site map,

Add the <siteMapNode> element with some basic information.

- Namely, you need to supply the title of the page (which appears in the navigation controls),
- a description (which you may or may not choose to use),
- the URL (the link for the page).

You add these three pieces of information using three attributes. The attributes are named title, description, and url, as shown here:

```
<siteMapNode description="CLASS" title="CLASS LIST"  url="~/CLASS.aspx">
```

Here's a complete, valid site map file that uses this page to define a website with exactly one page:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
 <siteMapNode description="CLASS" title="CLASS LIST" url="~/CLASS.aspx">
   <siteMapNode description="FYBCA" title="FYBCA" url="~/FYBCA.aspx"/>
 </siteMapNode>
</siteMap>
```

Rule 3: A <siteMapNode> Element Can Contain Other <siteMapNode> Elements

Site maps don't consist of simple lists of pages. Instead, they divide pages into groups. To represent this in a site map file, you place one <siteMapNode> inside another. Instead of using the empty element syntax shown previously, you'll need to split your <siteMapNode> element into a start tag and an end tag:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode description="CLASS" title="CLASS LIST"  url="~/CLASS.aspx">
    <siteMapNode description="SEM-I" title="SEM-I"  url="~/SEM1.aspx" />
    <siteMapNode description="SEM-II" title="SEM-II"  url="~/SEM2.aspx" />
    <siteMapNode description="SEM-III" title="SEM-III"  url="~/SEM3.aspx" />
```

```
  </siteMapNode>
</siteMap>
```

When you show this part of the site map in a web page, the Products node will appear as ordinary text, not a clickable link.

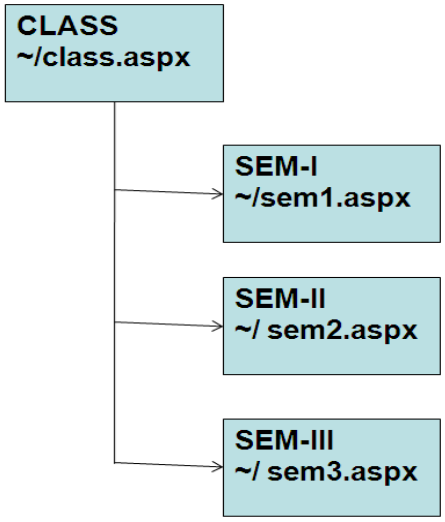


Figure : Three node in a site map

Rule 4: Every Site Map Begins with a Single <siteMapNode>
Rule 5: Duplicate URLs Are Not Allowed

Binding an Ordinary Page to a Site Map

To bind the site Map to the page , add the SiteMapDataSource control to your page. You can drag and drop it from the Data tab of the Toolbox. It creates a tag like this:

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

The SiteMapDataSource control appears as a gray box on your page in Visual Studio, but it's invisible when you run the page.
The last step is to add controls that are linked to the SiteMapDataSource
These are the three navigation controls:
TreeView: The TreeView displays a "tree" of grouped links that shows your whole site map at a look.
Menu: The Menu displays a multilevel menu. By default, you'll see only the first level, but other levels pop up when you move the mouse over the subheadings.
SiteMapPath: The SiteMapPath is the simplest navigation control—it displays the full path you need to take through the site map to get to the current page. For example, it might show

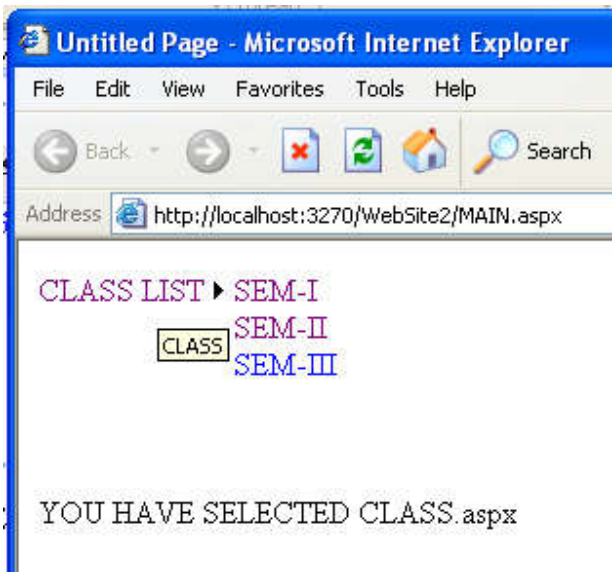
CLASS > SEM I > US01CBCA03

To connect a control to the SiteMapDataSource, you simply need to set its DataSourceIDproperty to match the name of the SiteMapDataSource.
For example, if you added a TreeView, you should tweak the tag so it looks like this:

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

For Menu

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



Binding a Master Page to a Site Map

Website navigation works best when combined with another ASP.NET feature—master pages. That's because you'll usually want to show the same navigation controls on every page. The easiest way to do this is to create a master page that includes the SiteMapDataSource and the navigation controls. You can then reuse this template for every other page on your site.
Here's how you might define a basic structure in your master page that puts navigation controls on the left:

```
<%@ Master Language="VB" CodeFile="MasterPage.master.vb" Inherits="MasterPage" %>
<html>
        <head runat="server">
                <title>Navigation Test</title>
        </head>
        <body>
        <form id="form1" runat="server">
        <table>
        <tr>
        <td style="width: 226px;vertical-align: top;">
        <asp:TreeView ID="TreeView1" runat="server" DataSourceID="SiteMapDataSource1" />
```

```
        </td>

        <td style="vertical-align: top;">
        <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server" />
        </td>
        </tr>
        </table>
        <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
        </form>
        </body>
</html>
```

Then, create a child with some simple static content:

Using Different Site Maps in the Same File

Imagine you want to have a dealer section and an employee section on your website. You might split this into two structures and define them both under different branches in the same file, like this:

```
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
<siteMapNode title="Root" description="Root" url="~/default.aspx">
<siteMapNode title="Dealer Home" description="Dealer Home"
url="~/default_dealer.aspx">
...
</siteMapNode>
<siteMapNode title="Employee Home" description="Employee Home"
url="~/default_employee.aspx">
...
</siteMapNode>
</siteMapNode>
</siteMap>
```

To bind the SiteMapDataSource to the dealer view (which starts at the Dealer Home page), you simply set the StartingNodeUrl property to "~/default_dealer.aspx".

You can even make your life easier by breaking a single site map into separate files using the siteMapFile attribute, like this:

```
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
<siteMapNode title="Root" description="Root" url="~/default.aspx">
<siteMapNode siteMapFile="Dealers.sitemap" />
<siteMapNode siteMapFile="Employees.sitemap" />
</siteMapNode>
</siteMap>
```

SiteMapNode Navigational Properties

Property	Description
ParentNode	Returns the node one level up in the navigation hierarchy, which contains the current node. On the root node, this returns a null reference.

ChildNodes HasChildNodes	Provides a collection of all the child nodes. You can check the property to determine whether child nodes exist.
PreviousSibling	Returns the previous node that's at the same level (or a null reference if no such node exists).
NextSibling	Returns the next node that's at the same level (or a null reference if no such node exists).

To see this in action, consider the following code, which configures two labels on a page to show the heading and description information retrieved from the current node:

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
lblHead.Text = SiteMap.CurrentNode.Title
lblDescription.Text = SiteMap.CurrentNode.Description
End Sub
```

If you're using master pages, you could place this code in the code-behind for your master page, so that every page is assigned its title from the site map.

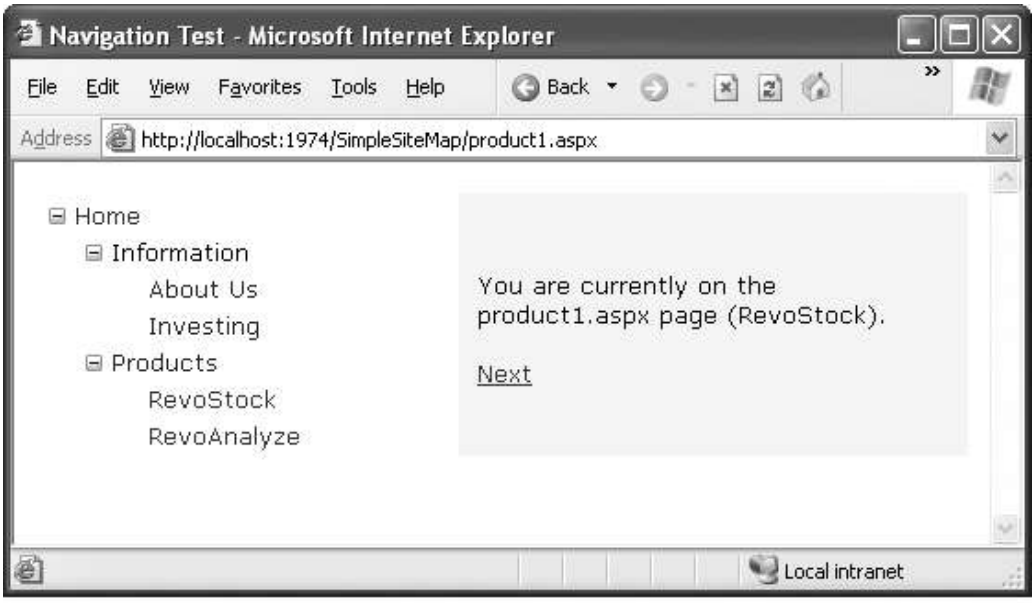
The next example is a little more ambitious. It implements a Previous/Next set of links, allowing the user to traverse an entire set of subnodes. The code checks for the existence of sibling nodes, and if there aren't any in the required position, it simply hides the links:

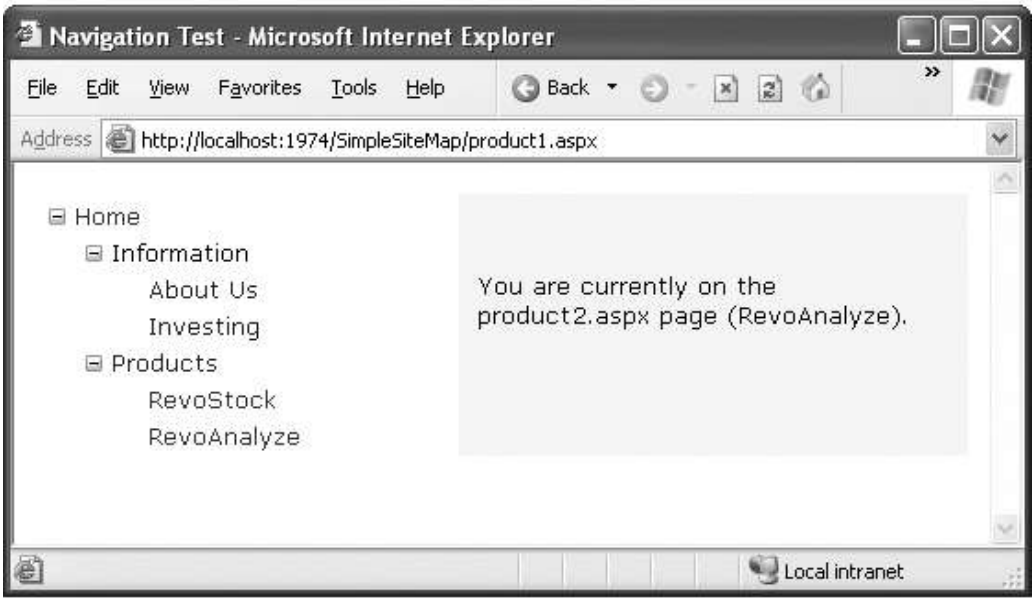
```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load

If SiteMap.CurrentNode.NextSibling IsNot Nothing Then
        lnkNext.NavigateUrl = SiteMap.CurrentNode.NextSibling.Url
        lnkNext.Visible = True
Else
        lnkNext.Visible = False
End If

End Sub
```

The first picture shows the Next link on the product1.aspx
page. The second picture shows how this link disappears when you navigate to product2.aspx (either by clicking the Next link or the RevoAnalyze link in the TreeView).





Mapping URLs

You define URL mapping in the <urlMappings> section of the web.config file. You supply two pieces of information—the request URL (as the attribute url) and the new destination URL (mappedUrl). Here's an example:

```
<configuration>
<system.web>
        <urlMappings enabled="true">
                <add url="~/category.aspx" mappedUrl="~/default.aspx?category=default" />
```

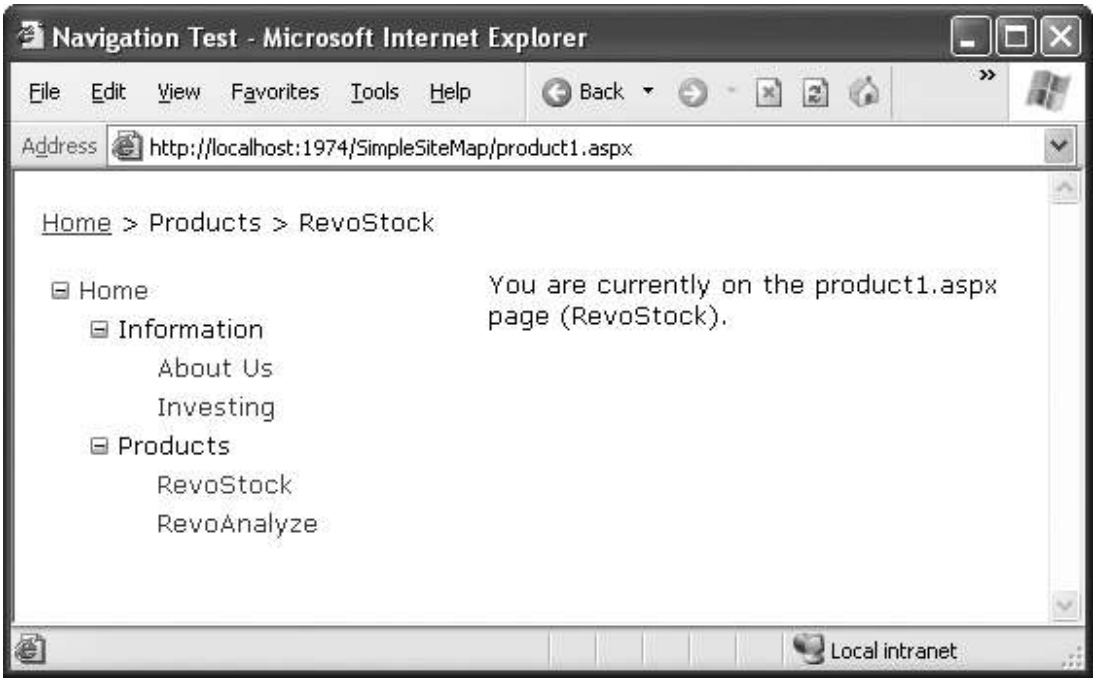
```
            <add url="~/software.aspx" mappedUrl="~/default.aspx?category=software" />
        </urlMappings>
...
…..
…..
</system.web>
</configuration>
```

The SiteMapPath Control

The TreeView shows the available pages, but it doesn't indicate where you're currently positioned.
To solve this problem, it's common to use the TreeView in conjunction with the SiteMapPath control. Because the SiteMapPath is always used for displaying navigational information (unlike the TreeView, which can also show other types of data), you don't even need to explicitly link it to the SiteMapDataSource:

```
<asp:SiteMapPath ID="SiteMapPath1" runat="server" />
```

The SiteMapPath provides ***breadcrumb*** *navigation*, which means it shows the user's current location and allows the user to navigate up the hierarchy to a higher level using links.
Following Figure shows an example with a SiteMapPath control when the user is on the product1.aspx page. Using the SiteMapPath control, the user can return to the default.aspx page.



Breadcrumb navigation with SiteMapPath

SiteMapPath Appearance-Related Properties

Property Description

ShowToolTips	Set this to False if you don't want the description text to appear when the user hovers over a part of the site map path.
ParentLevelsDisplayed	This sets the maximum number of levels above the current page that will be shown at once. By default, this setting is -1, which means all levels will be shown
RenderCurrentNodeAsLink	If True, the portion of the page that indicates the current page is turned into a clickable link. By default, this is False because the user is already at the current page.
PathDirection	You have two choices: RootToCurrent (the default) and CurrentToRoot (which reverses the order of levels in the path).
PathSeparator	This indicates the characters that will be placed between each level in the path. The default is the greater-than symbol (>). Another common path separator is the colon (:).

Using SiteMapPath Styles and Templates

Style	**Template**	**Applies To**
NodeStyle	NodeTemplate	All parts of the path except the root and

		current node
CurrentNodeStyle	CurrentNodeTemplate	The node representing the current page.
RootNodeStyle	RootNodeTemplate	The node representing the root. If the root node is the same as the current node, the current node template or styles are used.
PathSeparatorStyle	PathSeparatorTemplate	The separator in between each node

Imagine you want to change how the current node is displayed so that it's shown in italics.
To get the name of the current node, you need to write a data-binding expression that retrieves the title.
This data-binding expression is bracketed between <%# and %> characters and uses a method named Eval() to retrieve information from a SiteMapNode object that represents a page.

Here's what the template looks like:

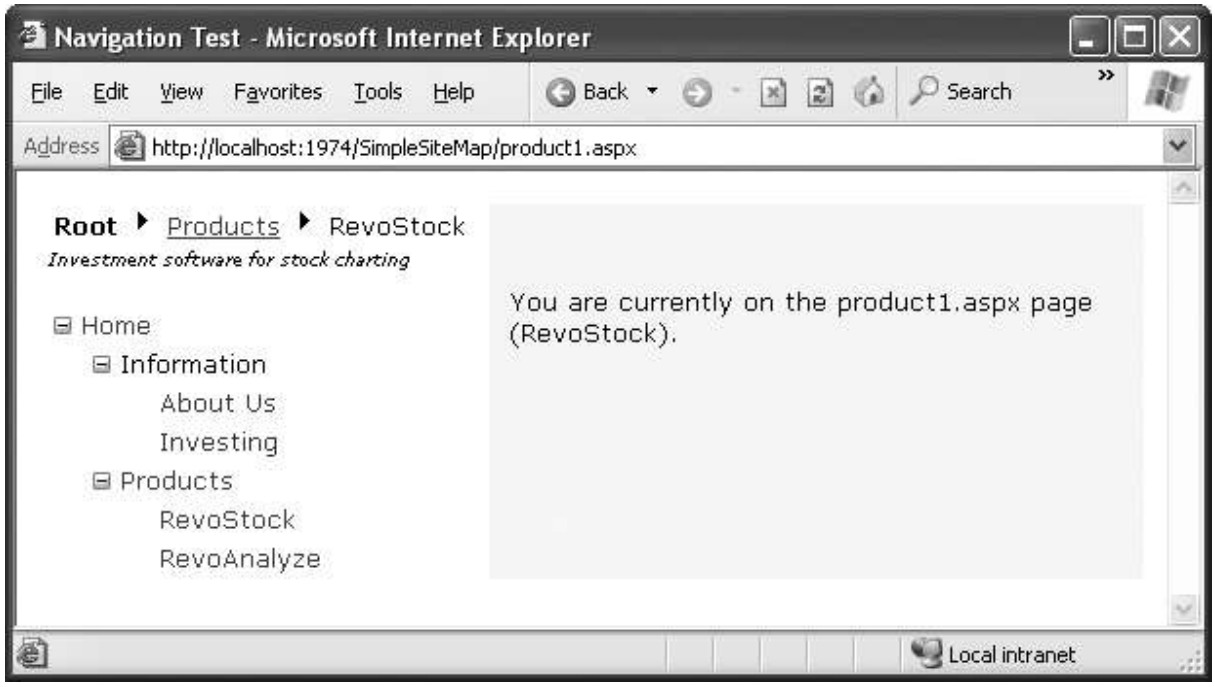
```
<asp:SiteMapPath ID="SiteMapPath1" runat="server">
<CurrentNodeTemplate>
        <i><%# Eval("Title") %></i>
</CurrentNodeTemplate>
</asp:SiteMapPath>
```

Data binding also gives you the ability to retrieve other information from the site map node, such as the description. Consider the following example:

```
<asp:SiteMapPath ID="SiteMapPath1" runat="server">
<PathSeparatorTemplate>
        <asp:Image ID="Image1" ImageUrl="~/arrowright.gif" runat="server" />
</PathSeparatorTemplate>

<RootNodeTemplate>
        <b>Root</b>
</RootNodeTemplate>


<CurrentNodeTemplate>
        <%# Eval("Title") %> <br />
        <small><i><%# Eval("Description") %></i></small>
</CurrentNodeTemplate>
</asp:SiteMapPath>
```



A SiteMapPath with templates

The TreeView Control

the TreeView can show a portion of the full site map or the entire site map. Each node becomes a link that, when clicked, takes the user to the new page. If you hover over a link, you'll see the corresponding description information appear in a tooltip.

TreeView Properties

Useful TreeView Properties

Property	Description
MaxDataBindDepth	Determines how many levels the TreeView will show. By default, MaxDataBindDepth is -1, and you'll see the entire tree. However, if you use a value such as 2, you'll see only two levels under the starting node. This can help you pare down the display of long, multileveled site maps.
ExpandDepth	Lets you specify how many levels of nodes will be visible at first. If you use 0, the TreeView begins completely closed. If you use 1, only the first level is expanded, and so on. By default, ExpandDepth is set to the constant FullyExpand (-1), which means the tree is fully expanded and all the nodes are visible on the page.
NodeIndent	Sets the number of pixels between each level of nodes in the TreeView. Set this to 0 to create a nonindented TreeView, which saves space. A nonindented TreeView allows you to emulate an in-place menu (see, for example, Figure 14-12).

ImageSet	Lets you use a predefined collection of node images for collapsed, expanded, and nonexpandable nodes. You specify one of the values in the TreeViewImageSet enumeration. You can override any node images you want to change by setting the CollapseImageUrl, ExpandImageUrl, and NoExpandImageUrl properties.
CollapseImageUrl,	Sets the pictures that are shown next to nodes for collapsed nodes
ExpandImageUrl, and	(CollapseImageUrl) and expanded nodes (ExpandImageUrl). The
NoExpandImageUrl	NoExpandImageUrl is used if the node doesn't have any children. If you don't want to create your own custom node images, you can use the ImageSet property instead to use one of several built-in image collections.
NodeWrap	Lets a node text wrap over more than one line when set to True.
ShowExpandCollapse	Hides the expand/collapse boxes when set to False. This isn't recommended, because the user won't have a way to expand or collapse a level without clicking it (which causes the browser to navigate to the page).
ShowLines	Adds lines that connect every node when set to True.
ShowCheckBoxes	Shows a check box next to every node when set to True. This isn't terribly useful for site maps, but it is useful with other types of trees.

TreeView Styles

Styles are represented by the TreeNodeStyle class, which derives from the more conventional Style class.

Table : TreeNodeStyle-Added Properties

Property	Description
ImageUrl	The URL for the image shown next to the node.
NodeSpacing	The space (in pixels) between the current node and the node above and below.
VerticalPadding	The space (in pixels) between the top and bottom of the node text and border around the text.
HorizontalPadding	The space (in pixels) between the left and right of the node text and border around the text.
ChildNodesPadding	The space (in pixels) between the last child node of an expanded parent node and the following node (for example, between the Investing and Products nodes in above (sitemap template) Figure).

Applying Styles to Node Types

The TreeView allows you to individually control the styles for types of nodes—for example, root nodes, nodes that contain other nodes, selected nodes, and so on. Table lists different TreeView styles and explains what nodes they affect.

Table: TreeView Style Properties

Property	Description
NodeStyle	Applies to all nodes. The other styles may override some or all of the details that are specified in the NodeStyle.
RootNodeStyle	Applies only to the first-level (root) node.
ParentNodeStyle	Applies to any node that contains other nodes, except root nodes.
LeafNodeStyle	Applies to any node that doesn't contain child nodes and isn't a root node.
SelectedNodeStyle	Applies to the currently selected node.
HoverNodeStyle	Applies to the node the user is hovering over with the mouse. These settings are applied only in up-level clients that support the necessary dynamic script.

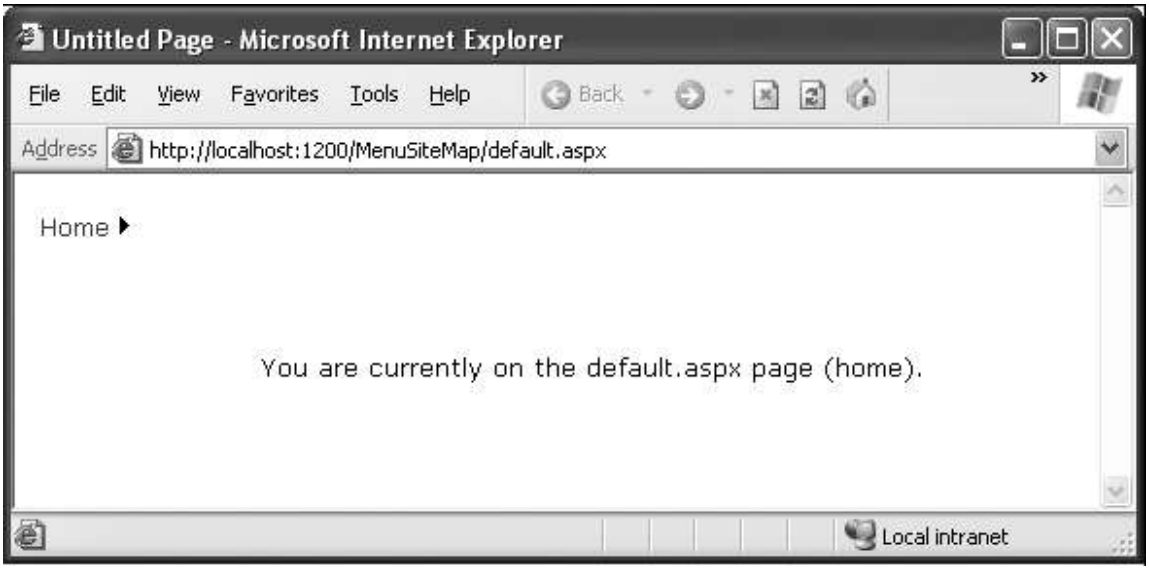
For example

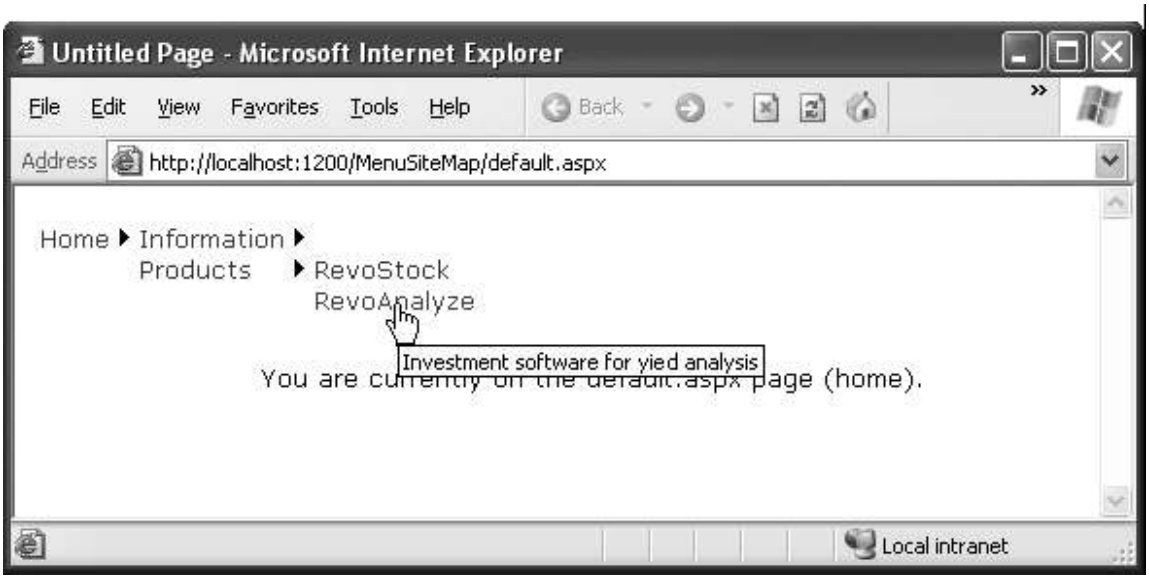
```
<asp:TreeView ID="TreeView1" runat="server" DataSourceID="SiteMapDataSource1">
        <NodeStyle Font-Names="Tahoma" Font-Size="10pt" ForeColor="Blue"
                HorizontalPadding="5px" NodeSpacing="0px" VerticalPadding="0px" />
        <ParentNodeStyle Font-Bold="False" />
        <HoverNodeStyle Font-Underline="True" ForeColor="#5555DD" />
        <SelectedNodeStyle Font-Underline="True" ForeColor="#5555DD" />
</asp:TreeView>
```

The Menu Control

The Menu control is another rich control that supports hierarchical data. Like the TreeView, you can bind the Menu control to a data source, or you can fill it by hand using MenuItem objects. To try the Menu control, remove the TreeView from your master page, and add the following Menu control tag:

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





Navigating through the menu

Difference between tree view and Menu as follow:

- The Menu displays a single submenu. The TreeView can expand an arbitrary number of node branches at a time.
- The Menu displays a root level of links in the page. All other items are displayed using fly-out menus that appear over any other content on the page. The TreeView shows all its items inline in the page.
- The Menu supports templates. The TreeView does not. (Menu templates are discussed later in this section.)
- The TreeView supports check boxes for any node. The Menu does not.
- The Menu supports horizontal and vertical layouts, depending on the Orientation property. The TreeView supports only vertical layout.

Menu Styles

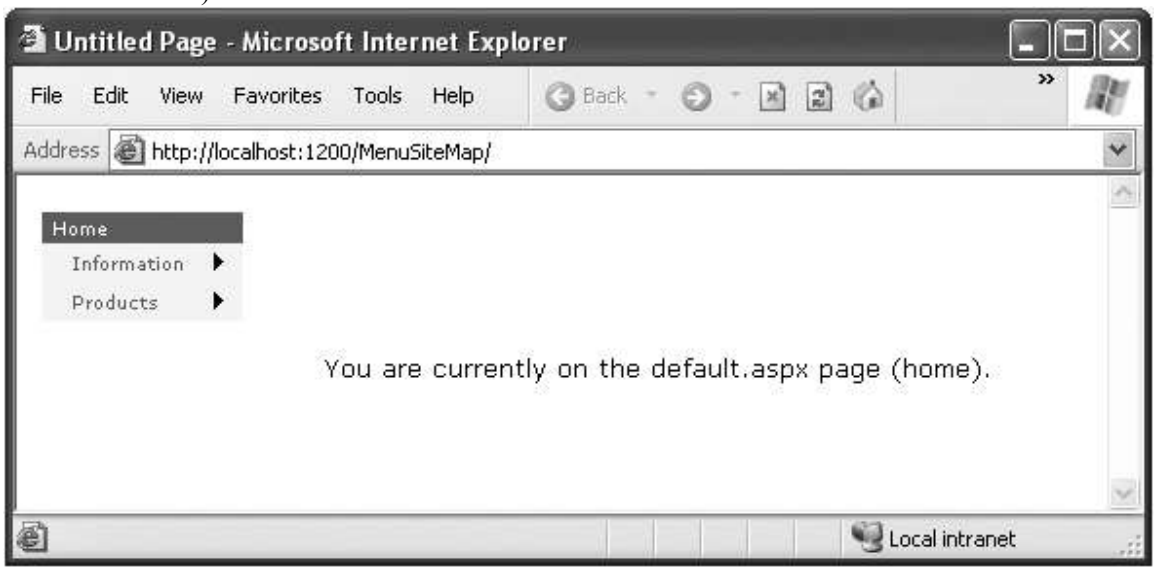
The Menu supports defining different menu styles for different menu levels.

Table *Menu Styles*

Static Style	Dynamic Style	Description
StaticMenuStyle	DynamicMenuStyle	Sets the appearance of the overall "box" in which all the menu items appear. In the case of StaticMenuStyle, this box appears on the page, and with DynamicMenuStyle it appears as a pop-up.
StaticMenuItemStyle	DynamicMenuItemStyle	Sets the appearance of individual menu items
StaticSelectedStyle	DynamicSelectedStyle	Sets the appearance of the selected item. Note that the selected item isn't the item that's currently being hovered over; it's the

		item that was previously clicked (and that triggered the last postback)
StaticHoverStyle	DynamicHoverStyle	Sets the appearance of the item that the user is hovering over with the mouse.

Following Figure shows the menu with StaticDisplayLevels set to 2 (and some styles applied through the Auto Format link).



A menu with two static levels

Master Page

Master page is a feature in ASP.NET 3.5 that helps define the overall layout of a web application and reuse the defined layout in all the pages derived from the master page. A master page contains markups and controls that you can share across different web pages of your web site. This makes your website more manageable and also avoids the duplication of code.

A master page contain markups, controls, banners, navigation menus and other elements that you want to include in all the pages of your website. The web pages that inherit the properties defined in master pages are called **Content Pages.** The content pages display their own properties as well as the properties inherited from master pages.

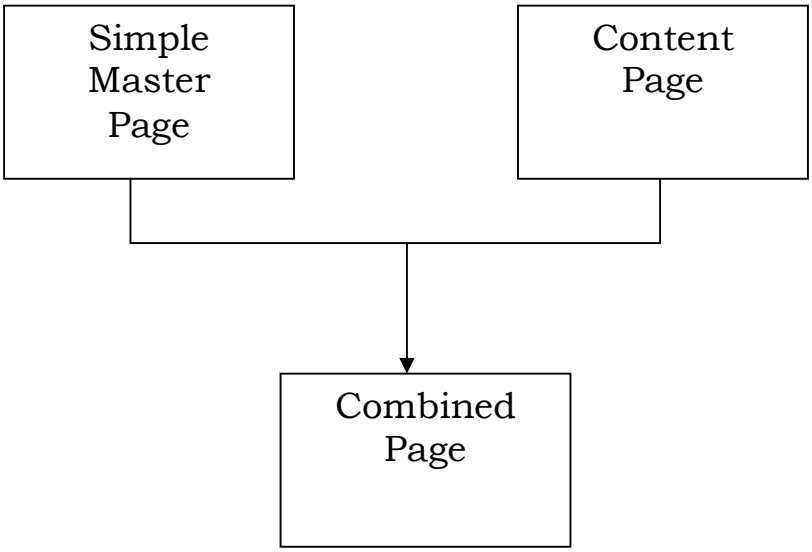
The following are the key features of the master pages:
→ Defining common properties of a website, such as banner, navigation menus or multiple master pages
→ Allowing a single or multiple content pages to access single or multiple master pages

→ Displaying the content of each content page in the content place holder of the master pages

To create a master page for a web site, identify the controls that you want to display on all the pages and then add these controls to the master page. You then create the ContentPlaceHolder control for the matser page to place the content of the web pages. Whne this website is executed, then the layout of the master page and the content in ContentPlaceHolder control are merged to display the output of the webpage. A master page contains multiple ContentPlaceHolder.

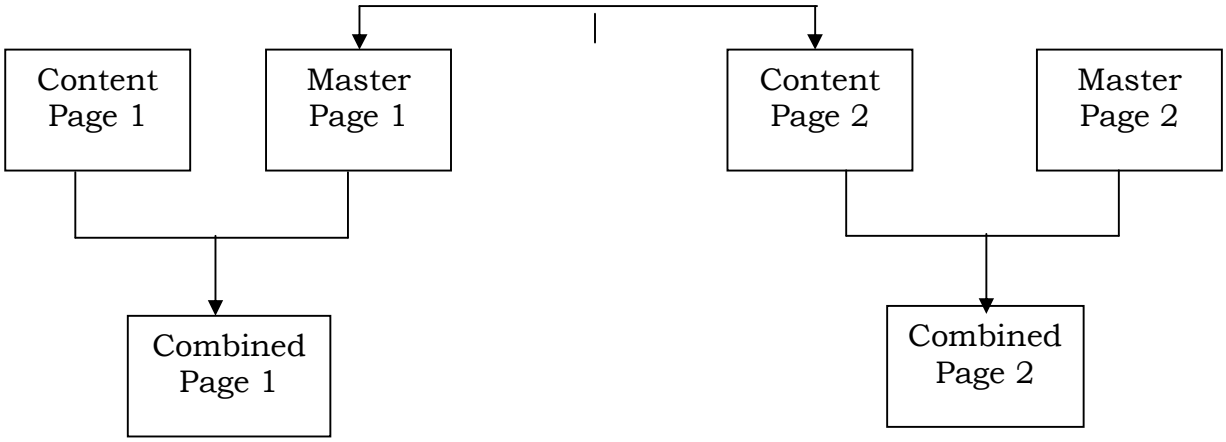
There TWO type of master pages: Simple mater page and Nested master page.

1) Simple master page: A simple master page simplify your website design by creating master pages and the content pages. You don't need to create the submaster pages. It is a page that is combined with the content page to display the combined page. The master and content pages are combined together to generate a single page that inherits its properties from both the pages.



[Figure of the block diagram of a combined page]

2) Nested master page: A nested master page is very much same as that of single master page. The only difference is that nested master pages are used in websites that have several hierarchical levels. For example, An application can have number of master pages depending on the level of hierarchy it incorporates.



[Figure of the block diagram of the Nested Master Page]