UNIT – 3

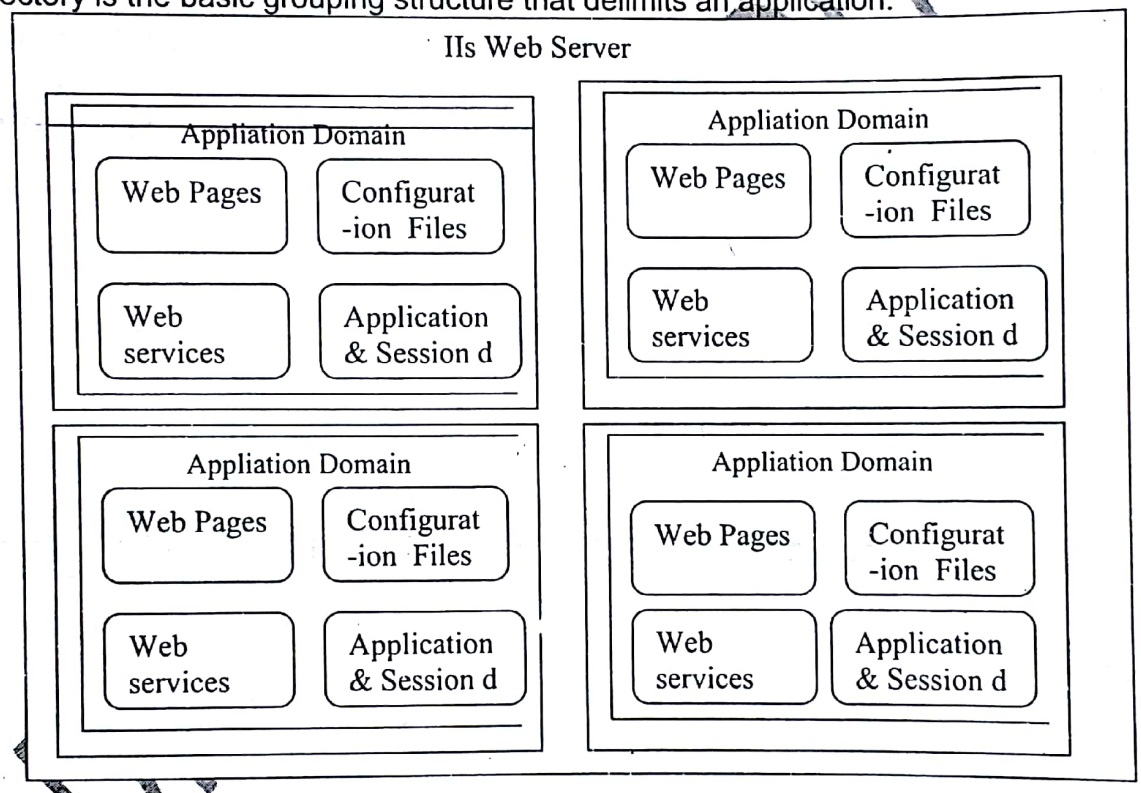
ASP.NET Applications

Web Application:

It's sometimes difficult to define exactly what a web application. It is unlike a traditional desktop program (which is usually contained in a single.exe file), ASP.NET applications are divided into multiple web pages.

Every ASP.NET application is executed inside a separate application domain, which is roughly similar to a windows "process" in unmanaged code. Application domains are isolated in memory, meaning that if one web application causes a fatal error it will not affect any other applications that are currently running. Similarly, it also means that the web pages in one application can't access the in memory information from another application. Each application has its own set of caching, application, and session state data.

ASP.NET application is a combination files, pages, handlers, modules, and executable code that can be invoked from a virtual directory on a web server. In other words, the virtual directory is the basic grouping structure that delimits an application.



ASP.NET File Types

FileType	Description
.aspx	These are Asp.NET web pages they contain the user interface and some or all of the underlying code.
.ascx	These are ASP.NET user controls. They are very similar to web pages, except that they must be hosted inside an .aspx file. User controls allow you to develop user interface.
.asmx	This is asp.net web services.
Web.config	This is the XML based configuration file for your ASP.NET Application. It includes settings for customizing security, state Management, memory management, etc.

Global.asax	This is the global application file. You use this file to define global variables and react to global events.
.disco or .vsdisco	These are special"discovery" files used to help clients find web services.
.vb or .cs	These are code behind files created in visual basic or c#. They allow you to separate code from the user interface logic in the web form page.
.resx	These file may exist if you are using Visual studio.NET .They are used to store information that you add at design time.
.sln, .suo, .vbproj, .csproj	This files are used by Visual Studio .NET to group together **projects**(a collection of files in a web application)and **solutions** (a collection of projects that you are developing or testing).

Execution Hello World.aspx

The Helloworld.aspx web page has two parts:-
1). A Script block that contains .CS or .Vb.net code.
2). A Form tag that holds all the ASP.NET server Controls used on the page.
For example a label control outputs plain text in a browser.

```
<form id="form" runat="server"
   <asp: Label id="lbltest" runat="server" / >
</form>
```

A Script block that contains .cs code:-

```
protected void Page_Load(object sender, EventArgs e)
     {
       lbltest.text="Hello, The Page load event occurred."

     }
```

C # code

.vb code

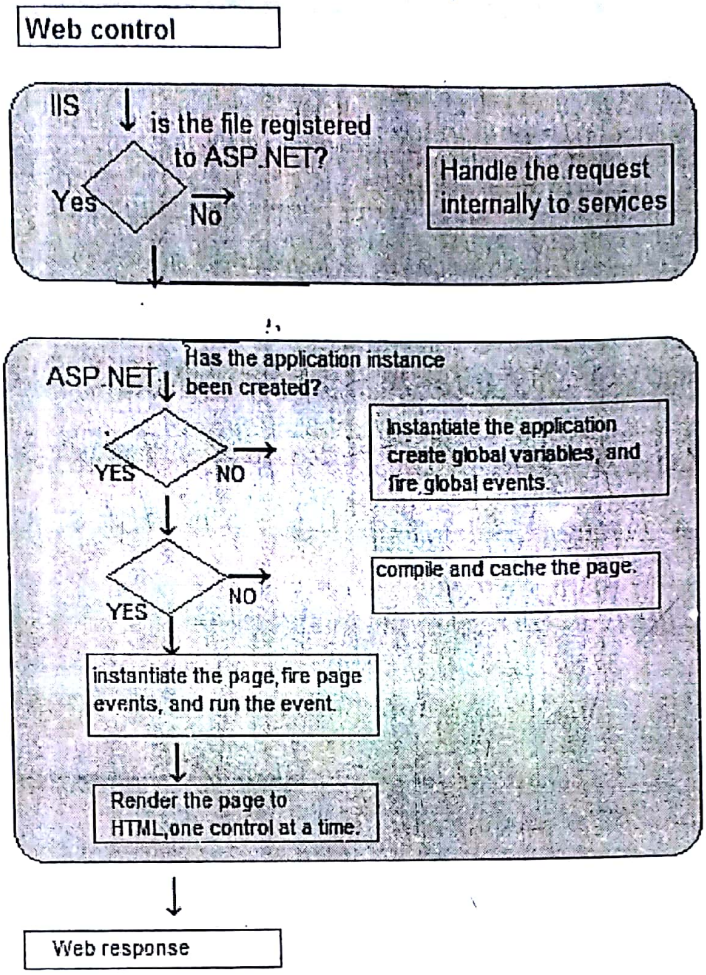
```
Public sub page_load()
       lbltest.text="Hello, The Page load event occurred."
End sub
```

V.b. code

Behind the scenes with HelloWorld.aspx(steps)

When asp.net receives the request for the helloworld.aspx file. The process has several steps.

1. First of all, IIS determines that the .aspx file extension belongs to ASP.NET and passes it along to the ASP.NET worker process. If the file extension belonged to another service, ASP.NET would never get involved.
2. If this is the first time a page in this application has been requested, ASP.NET automatically creates the application domain and a special application object. If there are any global.asax variables that need to be created or events that need to be run, ASP.NET performs those tasks now.



3. ASP.NET now considers the specific .aspx file. If it has never been executed, ASP.NET compiles and caches an executable version for optimum performance.and the file has not been changed, ASP.NET will use the precompiled version.
4. The compiled HelloWorld.aspx acts like a miniature program. It starts and runs all its code. At this stage, everything is working together as a set of in memory .NET objects.
5. When the code is finished, ASP.NET asks every control in the web page to render itself.
6. The final page is sent to the user, and the application ends.

Code Behind

With the code-behind, you create a separate code file (.vb for vb.net and .cs file for c#) to match every .aspx file. The .aspx file contains the user interface logic, which is the series of html code and ASP.NET tags that create controls on the page. The .vb or .cs file contains code only. At the top of the .aspx file, you use a special Page directive that identifies the matching code –behind file.

C#

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

Vb.net

<%@ Page Language="VB" Inherits="Mypageclass" src="Mypage.vb" %>

This directive defines the language(vb), identifies the page class that contains all the page

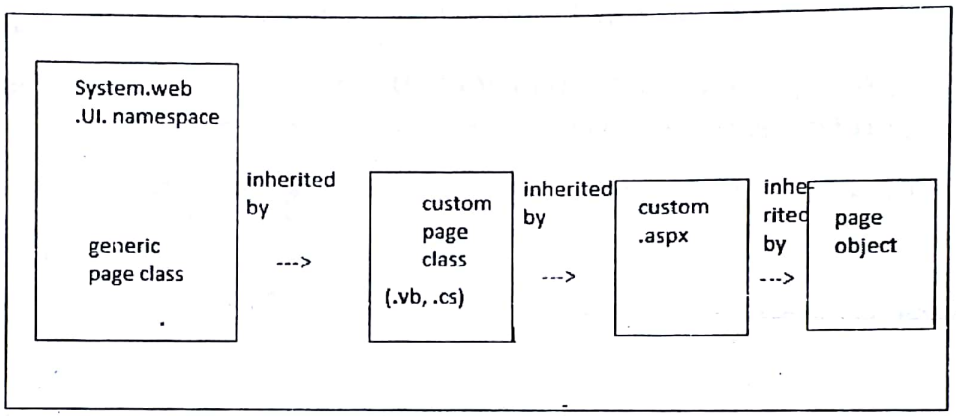
code(Mypageclass), and identifies the source file where the page class can be found(Mypage.vb).

Note :- The Page Directive Sets Pre-Processor Options, Directives are processed before any ASP.NET code is executed, and set various options that influence how the ASP.NET compiler process your file.set the default language

Web Form Inheritance Explained

The words "Inherits" is used two twice:

1) **Once in the .aspx template file**
2) **Once in the code file**



1. First, there is page class from the .NET class library. This class library. This class defines the basic functionality that allows a web page to host other controls, render itself to HTML, and provide access to the traditional ASP objects such as Request, Response and Session.
2. Second, there is your code behind class. This class inherits from the page class to acquire the basic set of ASP.NET web page functionality.
3. Finally, the aspx page inherits the code from the custom form class your created.

Notes:→

First you create a code behind file as normal when you compile it using the command line compiler for VB.NET you use the VBC.EXE utility included with the .NET framework & found in the c:\winivt\microsoft.net\framework\version\directory name.
So we cannot learn this long path.net facility the path variable to store the long path in path variable.

Three Ways to Code Web Forms:→

There are three different ways that you can program web forms.

1. **Traditional inline code**
2. **Code- behind**
3. **Compiled code behind**

Development Type	You create	You Deploy(to the Web Server)
Traditional inline code	.aspx files that contain code and user interface layout.	.aspx files
Code behind	.aspx files with user interface and .VB files with code.	.aspx and .vb files
Compiled code behind	.aspx files with user interface and .VB files with code.	.aspx files and the compiled .DLL files(which go in the \bin directory)

Traditional in line code:-

You can also import namespace into an .aspx file using a directive this is technique you would probably use if you work not using code behind development,

<%@ import Namespace ="System. Web. Ui " %>
<%@ import Namespace ="System. Web. Ui.Webcontrols "%>

Code Behind:-

Code behind represent a true class file is uses all Namespaces hierarchy, class than after all the code events.

Advantage of code behind file (.cs or .vb)

1) **Better organization:**
With code behind file, your code is encapsulated in a class it's easier to read & it's easier to isolate & reuse paces of useful functionality.

2) **Separation of the user interface:**
With code behind the aspx file can be manipulated, polished & perfected by any other user as long as the control names remain the same.

3) **The ability to use advanced code editors:**
VS.NET can automatically verified the syntax of your code behind files, provide intelligence statement completion & allow you to designer corresponding .aspx web form.

Web Form Fundamentals

In ASP.NET, server side web controls. These controls are created and configured as objects, and automatically provide their own HTML output.

Server Controls:

In Asp.net Provides two sets of server controls

1. HTML Server Controls
2. WEB Server Controls

➢ HTML SERVER CONTROL

Html server controls are server based equivalents for standard html elements. They are also useful when migrating existing asp pages to asp.net as they require the some changes.

➢ WEB SERVER CONTROL

Web controls are similar to html server controls but provide a richer object model with a variety of properties for style of formatting details. They also provide more events and more closely resemble the controls used for Windows development.

HTML SERVER CONTROLS:

Html server controls provide an object interface for standard html elements. They provide three key features

1> They generate their own interface:-

You set properties in code & they underlying html tag is updated automatically when the page is rendered & sent to the client.

2> They retain their state:-

You can write your program like traditional window program. there's no need to recreate a web page form scratch each time you send it to the user.

3> They fire events:-

Your codes can response to these events, just like normal controls in window application.

NOTES

Asp.net controls maintain state automatically, for example: - if you enter information in the text box and click on the submit button to post the page, the refresh page will still contain the value you enter in the text box.

HTML CONTROL CLASSES:

All the HTML server controls are defined in the System.Web.UI.HTMLControls namespace. There is a separate class for each kind of control. Following table describe the HTML server controls.

Class Name	HTML TAG Represented
HtmlAnchor	<a>
HtmlButton	<button>
HtmlForm	<form>
HtmlImage	<img>
HtmlInputButton	<input type="button">, <input type="submit">and <input type="retest">
HtmlInputCheckbox	<input type="chkbox">
HtmlInputControl	<input type="text">, <input type="submit"> and <input type="file">
HtmlInputFile	<input type="file">
HtmlInputHidden	<input type="hidden">
HtmlInputImage	<input type="image">
HtmlInputRadiaobutton	<input type="radio">

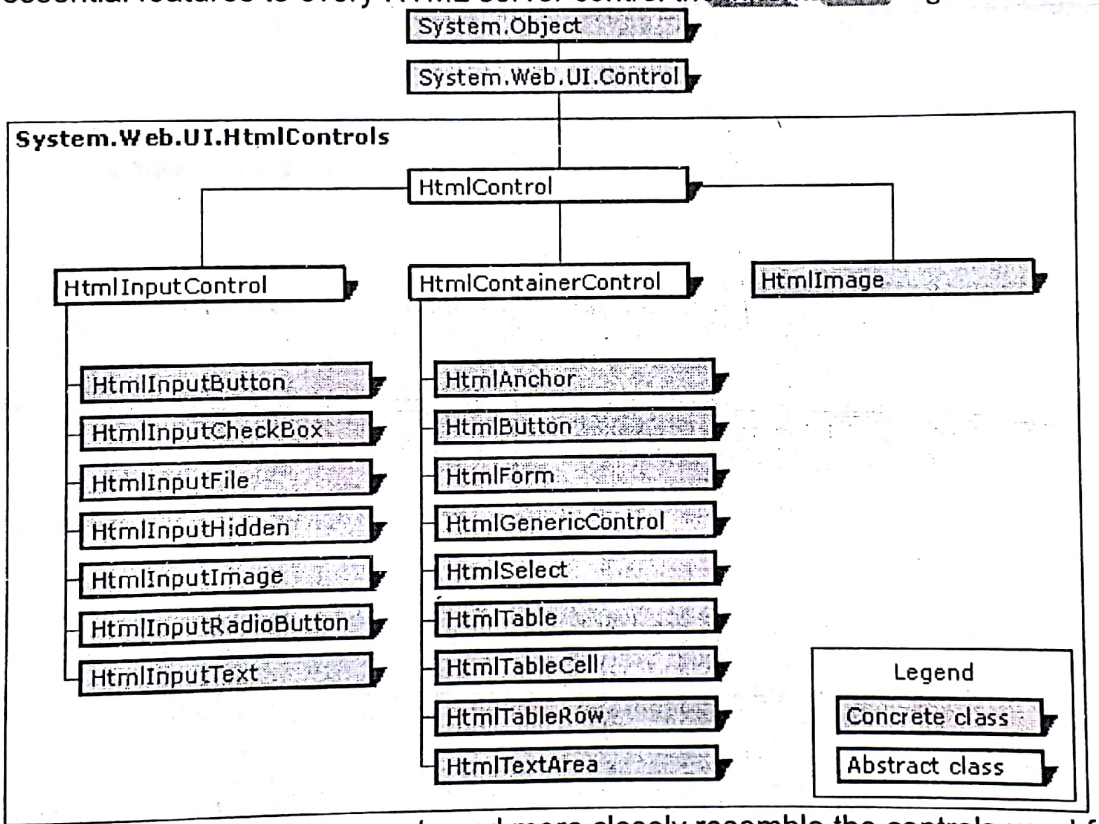
HtmlInputText	<input type="text"> and <input type="password">
HtmlAnchor	Href, Target, Title
HtmlImage and HtmlinputImage	Src, Alt, Width, and Height
HtmlInputCheckbox and HtmlInputRadiaobutton	Checked
HtmlInputText	Value
Htmlselect	Items
HtmlTextArea	Value
HtmlGenericControl	Inner Text

Most common used Properties

HtmlInputSelect	<select>
HtmlTable,Htmltablerow, and Htmltablecell	<table>,<tr>,<td>
Htmltextarea	<textarea>
HtmlGenericControl	Any other HTML element

HTML Control Classes

Every HTML controls inherits from the base class HTMLControl class, which provides essential features to every HTML server control the inheritance diagram : -



They also provide more events and more closely resemble the controls used for Windows development.

HtmlServer controls generally provide properties that closely match their tag attributes, for example:- HtmlImage class provides Aligns, Alt, Border, Src, Height and Width properties and one of two possible events:

a) ServerClick
b) ServerChange

a. ServerClick Event:

ServerClick is simply a click that is processed on the server side. It's provided by most button controls, and it allows your code to take immediate action. This option might override the expected behaviour.

b. ServerChange Event:

ServerChange event responds when a change has been made to a text or Selection control. This event is not as useful as it appears because it doesn't occur until the page is posted back. For example:- After the user clicks a submit button. The serverChange event occurs for all changed controls, followed by the appropriate ServerClick.

Event	Control
ServerClick	HtmlAnchor , HtmlForm, HtmlButton, HtmlInputButton ,HtmlInputImage.
ServerChange	HtmlInputText, HtmlInputCheckBox ,HtmlInputRadioButton, HtmlInputHidden, HtmlSelect, HtmlTextArea.

HTML CONTROL BASE CLASS:

Every HTML control inherits from the base class HtmlControl. This relationship means that every HTML control will support a basic set of properties and features.

The Html Control class also provides built-in-Support for data binding.

Properties	Description
Attributes	Provides a collection of all the tag attributes and their values. Rather than setting an attribute directly, it's better to use the corresponding property. However, this collection is useful if you need to add or configure a custom attribute or an attribute that doesn't have a corresponding property.
Controls	Provides a collection of all the controls contained inside the current control.
Disabled	Set this to true to disable the control, ensuring that the user cannot interact with it and its events will not be fired.
EnableViewState	Set this to false to disable the automatic state management for this control. In this case, the control will be reset to the properties and formatting specified in the control tag every time the page is posted back. If this is set to true the control uses the hidden input field to store information about its properties, ensuring that any changes you make in code are remembered.
Page	Provides a reference to the web page that contains this control as a System.Web.UI.Page object.
Parent	Provides a reference to the control that contains this control. if the control is placed directly on the page, it will return a reference to the page object.
Style	Provides a collection of css style properties that can be used to format the control.
TagName	Indicates the name of the underlaying HTML element(for example,"img" or "div")
Visible	When set to false, the control will be hidden and will not be rendered to the final HTML page that is sent to the client.

HtmlInputControl Class

This defines some properties that are common to all the HTML controls based on the <input type="text">, <input type="submit"> and <input type="file"> elements.

Properties

1). Type: - Provides the type of input control.
2). Value: - Returns the contents of the control as a string.

HtmlContainerControl Class

Any HTML control that requires a closing tag also inherita from the HtmlContainer control. For example, elements such as <a>, <form>, and always use a closing tag, while <img> and <input> can be used as single tags. Thus , the HtmlAnchor, Htmlform, and HtmlGenericControl classes inherit from HtmlContainerControl, while HtmlImage and HtmlInput do not.

The HtmlContainer Control adds two properties

1). InnerHtml:- The Html content between the opening and closing tags of the control. Special characters that are set through this property will not be converted to the equivalent HTML entities.

2).InnerText : - The text content between the opening and closing tags of the control. Special characters will be automatically converted to the HTML entities and displayed like text. For example, the less than character(<) will be converted to ⁢ and will be displayed as < in the web page.

The HttpRequest Class:

The HTTPRequest class encapsulates all the information related to a client request for a web page. Most of this information corresponds to low-level details such as posted back from values, server, variables, and the response encoding & so on.

The HttpResponse Class:

The HttpResponse class allows you to send information directly to the client. In traditional ASP development, the response object was used heavily to create dynamic pages.

Now, with the introduction of the new server based control model, these relatively crude methods are no longer needed. The HttpResponse does still provide some important functionality: namely caching support, cookie features, and the Redirect method that allows you to transfer the user to another page.

Member	Description
BufferOutput	When set to True, the page is not sent to the client until it is completely rendered and ready to send, rather than piecemeal.
Cache	References an Http cache policy object that allows you to configure how this page will be cached.
Cookies	The collection of cookies sent with the response you can use this property to add additional cookies.
Write( ), Binary( ), Write( ) and WriteFile( )	These methods allow you to write text or binary content directory to the response stream. You can even write the contents of a file. These methods are deemphasized in ASP.NET, and shouldn't be used in conjunction with server controls.
Redirect( )	This method transfers the user to another page in your application or a different website.

Example: (vb.net)

'You can redirect to a file in the current directory……..

Response. Redirect ("Newpage.aspx")

'You can redirect to another website……..

Response. Redirect ("Http:/www.prosetech.com")

The ServerUtility Class:

The Server utility class provides some miscellaneous helper methods. The most commonly used are UrlEncode / UrlDncode / and HtmlEncode / HtmlDecode. These functions change a string into a representation that can safely be used as part of a URL or displayed in a webpage.

Methods	Description
CreateObject	Creates an instance of the COM object that is identified by its programmatic ID. This is included for backward compatibility as it will generally be easier to interact with COM objects using the .NET framework services.
HtmlEncode & HtmlDecode	Changes an ordinary string into a string with legal. HTML characters & back again.
UrlEncode & UrlDecode	Changes an ordinary string into a string with legal URL characters & back again.
MapPath	Returns the physical file path that corresponds to a specified virtual file path on the web server.
Transfer	Transfer execution to another web page in the current application. This is similar to the Response. Redirect method, but slightly faster. It Cannot be transferring the user to a site on another web server.

Basic Web Controls

The Web Server Control:

Html Server controls are still limited than web server controls need to .. … For example, Every Html control corresponds directly to an Html tag, meaning that you are bound by the limitations & abilities of the Html language. Web controls on the other hand, emphasized the features of web design:-

Features of Web Control:

- They provide rich user interface.
- They provide a consistent object model.
- They tailor their output automatically.
- They provide high level features.

The WebServer Control Classes:

The basic control classes and the HTML elements they generate. Note: - that some controls such as the button and textbox, can be rendered as

different HTML elements. ASP.NET uses the element that matches the properties you have set. Also, some controls have no single HTML equivalent. For Example,

The checkboxlist and radiobuttonlist controls output as a<table> that contains the multiple HTML checkboxes or radio buttons. ASP.NET warps them into a single object on the server side for convenient programming. This is the most features of web controls.

Control Class	HTML Element
Label	<Span>
Button	<input type= "submit"> or <input type = "button">
Textbox	<input type= "text"> ,<input type = "Password"> or<text area>
Checkbox	<input type= " Checkbox">
Radio Button	<input type= " Radio">
Hyperlink	<a>
Link Button	<a> with a contained <img>tag.
Image button	<input type= " Image">
Image	<img>
ListBox	<select size="x" where x is the number of rows that are visible at once.
DropdownList	<select>
CheckboxList	A list or <table> with multiple <input type="Checkbox">tags.
RadioButtonList	A list or <table> with multiple<input type="Radio">tags.
Panel	
Table, Table Rows, & Table Cell	<Table>, <tr> and <td> or <th>

The Web Control Tags

The ASP.NET tags have a special format. They always begin with the prefix asp: followed by the class name (<asp: textbox>). If there is no closing tag, the tag must end with />. Any attributes in the tag correspond to a control property, aside from the runat="server" attribute, which declares that the control will be processed on the server.

The following example of text box in the asp.net

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

When a client requests this .aspx page (default.aspx), the following HTML is returned. The name is a special attribute that ASP.NET uses to track the control.

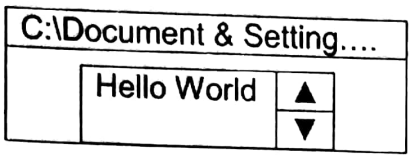
<Input type="text" name="ctrl0"/>

You could place some text in the textbox, set its size, make it read only and change the background color. All this actions have defined properties. For example:- Textbox.textmode property allows you to specify SingleLine(Default), Multiline, or Password.

<asp: Textbox Id="txt" Backcolor="Yellow" Text="Hello World" Readonly="True" Textmode="Multiline" Rows="5" Runat="server" />

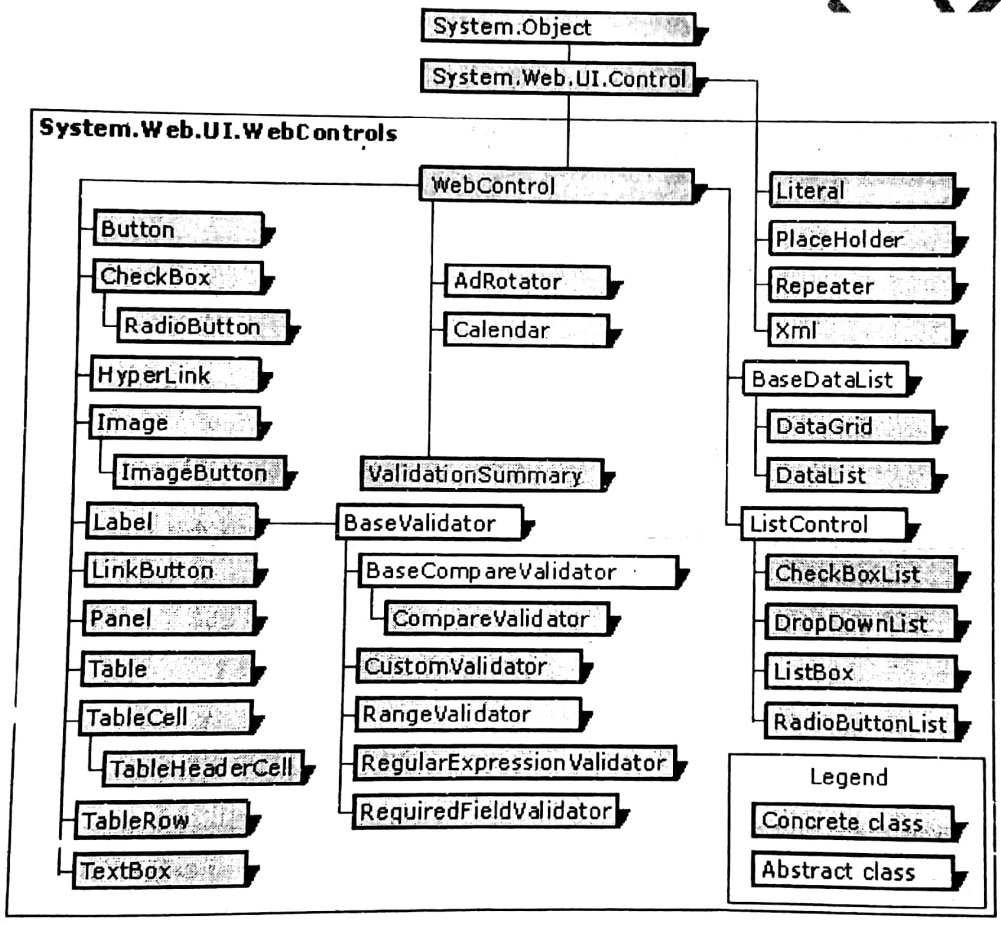
The resulting HTML uses the text area element and sets all the required style attributes.

<Textarea Name="txt" Rows="5" Readonly="Readonly" Id="txt" Style="background-Color: yellow ;> Hello World < / text area >



The Web Control Classes:

Web control classes are defining in the System.web.UI.webcontrols namespace. They follow a slightly more tangled object hierarchy than HTML server controls.



The Web Control Base Classes:

All the web controls begin by inheriting from the WebControl base class. This class defines all functionality for tasks as data binding, and includes some basic properties that you can use with any control.

Property	Description
Access Key	It specifies the keyboard shortcuts as one letter… **Ex:** If you set this to "y", the Alt-y keyboard combination will automatically change focus to this web control. This feature is only supported on internet explorer 4.0 & higher.
Backcolor, Bordercolor, & forecolor	Sets the colors used for the background, forground & border of the control. Usually, the foreground color is used for text.
Border Width	Specifies the size of the control border.
BorderStyle	One of the styles values from the border Style enumeration, including dashed, Dotted, double, groove, Ridge, Inset, Outset solid &None.
Controls	Provides a collection of all the controls contained inside the current control. Each object is provided as a generic System.web.UI.Control object, so you may need to cast the reference with the ctype function to access control specific properties .
Enabled	When set to false, the control will be visible, but will not be able to receive user input or focus.
EnableViewState	Set this to false to disable the automatic state management for this control. In this case, the control will be reset to the properties & formatting specified in the control tag every time the page is posted back. If this is set to true(the default), the control uses the hidden input field to store information about its properties, ensuring that any changes you make in code are remembered.
Font	Specifies the font used to render any text in the control as a special System drawing font object.
Height & Width	Specifies the Width & height of the control for some controls, these properties will be ignored when used with older browsers.
Page	Provides a reference to the web page that contains this control as a System.web.UI.Page object.
Parent	Provides a reference to the control that contains this control. If the control is placed directly on the page, it will return a reference to the page object.
TabIndex	A number that allows you to control the tab order. The control with a tabindex of 0 has the focus when the page first loads. →Pressing TAB moves the user to the control with the next lowest tabIndex,provided it is enabled. This property is only supported in the internet explorer 4.0 & higher.
ToolTip	Display a text message when the user hover the mouse above the control. Many older browsers do not support this property.
Visible	When set to false, the control will be hidden & will not be rendered to the final HTML page that is sent to the client.

Units

All the properties are measurements, including BorderWidth, Height, and Width, use the special Unit structure. When setting a unit in a control tag, make sure you add append px (pixel) or % (percentage) to the number to indicate the type of unit.

<asp:Panel height="300px" width="50%" id="pnl" runat="server" />

if you are assigning one of these values at runtime, you need to use one of the shared properties of the Unit type.

- **Border width, Height:**
  //Convert the number 300 to a Unit object representing pixels, and assign it.

EX:-

Pnl. Height=Unit. Pixel (300)

OR

//Convert the number 50 to a Unit object representing percent, and assign it.
Pnl. Width=Unit. percentage(50)

- **Border Style:**

```
Ctrl.BorderStyle=BorderStyle.Dashed
<asp: Textbox BorderStyle="Dashed" Text="Hello"Id="txt"Runat="server"/>
```

Colors:

The color property refers to color object from the System. Drawing namespace. Color objects can be created in three ways……

1. **Using an ARGB (Alpha, Red, Green, Blue):-** You specify each value as an integer.
2. **Using a predefined .Net color name:** You choose appropriate name read-only property from the color class.
3. **Using an HTML color name:** You specify this value as a string using ColorTranslator class.

EX:--
```
//Create a color from argb value.
Int Alfa, Red, Green, Blue;
Alfa=255;
Green=255;
Red=0;
Blue=0;
Ctrl.Forecolor=Color.FromARGB (Alfa, Red, Green, Blue);

//Create a color using a .NET name
Ctrl.Forecolor=Color.Crimson;

// Create a color from an HTML code
Ctrl.Forecolor=ColorTranslator.FromHtml ("Blue");
```

Font properties:

The font property is actually references the FontInfo object which is defining in the **System.Drawing namespace.**

Every font Info object has several properties that define its name, size and Bold, Italic, Strikeout, Underline and Overline.

Example:-

```
Ctrl.Font.Name = " Verdana ";

Ctrl.Font.Bold= "True";

// specifies a relative size using the FontUnit type.
Ctrl.Font.Size = FontUnit.Small;
// specifies an absolute size of 14 pixels.
Ctrl.Font.Size = FontUnit.Points(14)
```

List Controls:

The simple list controls (List Box, DropDwonList, CheckBoxList, and RadioButtonList). Every list controls provides a **selectedIndex property** that indicates the selected row as a zero based index.

For Example: If the first item in the list is selected, ctrl.SelectedIndex will be 0. The List controls also provide an additional SelectedItem property, which allows your code to retrieve the ListItem object that represents the selected item. The list item object provides three important properties.

1) Text(Display the contains)
2) Value(The hidden value in the HTML code)
3) Selected (True or False depending on whether the item is selected)

Multiple Selected List Controls:

If you have set the selectionmode property to the enumerated value ListSelectionmode.Multiple. with the CheckBoxList, multiple selections are always possible.

You need to iterate through the items collection of the list control, and check the ListItem.

Example:
The aspx file defines CheckListBox, Button, and Label controls

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="WebApplication1._Default" %>

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

            Choose Your Favorite Programming Languages : <br />
              <asp:CheckBoxList ID="chklst" runat="server"> </asp:CheckBoxList><br />
    <asp:Button ID="cmdok" runat="server" Text="OK" /> <br/>
              <asp:Label ID="lblResult" runat="server" Text="Label"></asp:Label>
        </div>
        </form>
    </body>
</html>
```

C#.net

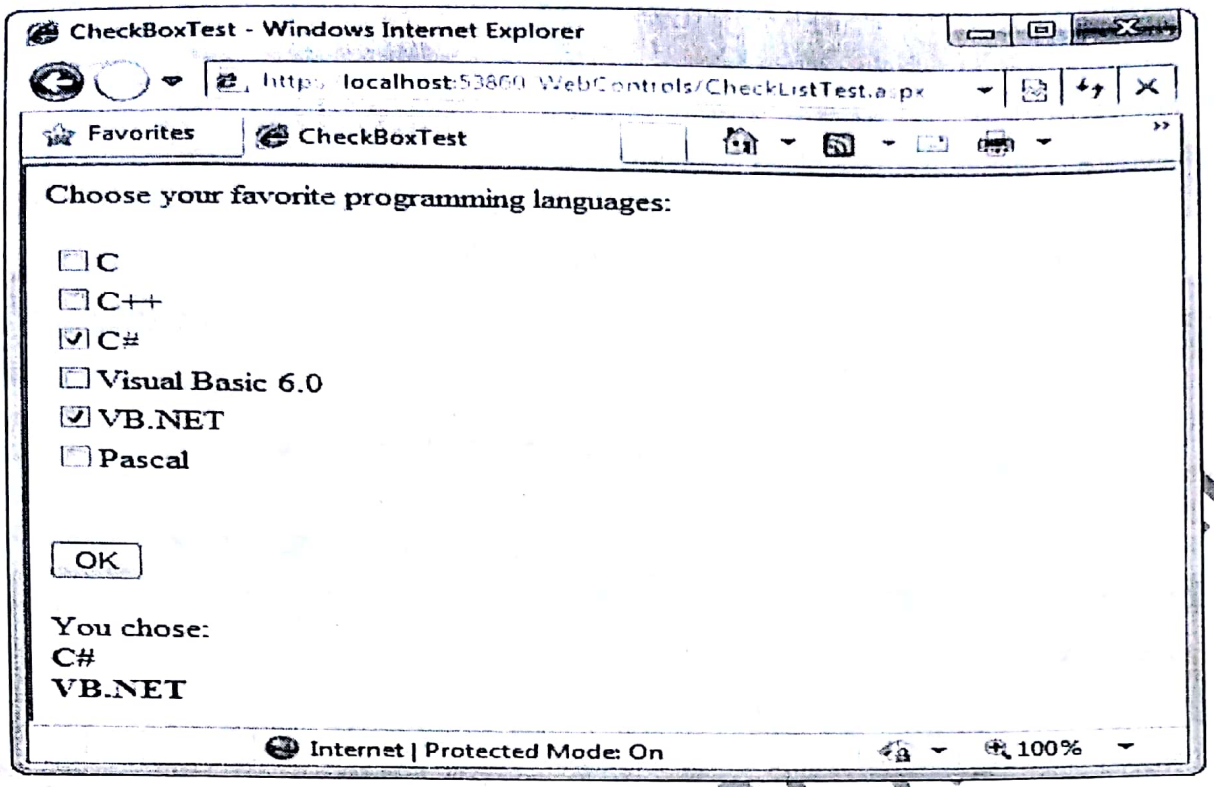
```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;

namespace WebApplication1
{
   public partial class _Default : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         if (IsPostBack)!= false
            chklst.Items.Add("C");
            chklst.Items.Add("C++");
            chklst.Items.Add("C#");
            chklst.Items.Add("Visual Basic 6.0");
            chklst.Items.Add("VB.Net");
            chklst.Items.Add("Pascal");

      }

      protected void cmdok_Click(object sender, EventArgs e)
      {
         lblResult.Text = "You Choose: <b>";
         ListItem lstitem;
         foreach lstitem in chklst.Items
         {
            if (lstitem.Selected=true)
              {
               // Add text to the Label.
               lblResult.Text += "<br>" + lstitem.Text;
              }

         }
      }
   }
}
```



The BulletedList Control

The BulletedList control is a server-side equivalent of the <ul> (unordered list) and <ol> (ordered list) elements. As with all list controls, you set the collection of items that should be displayed through the Items property. Additionally, you can use the properties in Table 6-4 to configure how the items are displayed.

Table 6-4. *Added BulletedList Properties*

Property	Description
BulletStyle	Determines the type of list. Choose from Numbered (1, 2, 3, . . .), LowerAlpha (a, b, c, . . .) and UpperAlpha (A, B, C, . . .), LowerRoman (i, ii, iii, ; . . .) and UpperRoman (I, II, III, . . .), and the bullet symbols Disc, Circle, Square, or CustomImage (in which case you must set the BulletImageUrl property).
BulletImageUrl	If the BulletStyle is set to CustomImage, this points to the image that is placed to the left of each item as a bullet. FirstBulletNumber
FirstBulletNumber	In an ordered list (using the Numbered, LowerAlpha, UpperAlpha, LowerRoman, and UpperRoman styles), this sets the first value. For example, if you set FirstBulletNumber to 3, the list might read 3, 4, 5 (for Numbered) or C, D, E (for UpperAlpha).
DisplayMode	Determines whether the text of each item is rendered as text (use Text, the default) or a hyperlink (use LinkButton or HyperLink). **The difference between LinkButton and HyperLink** is how they treat clicks. When you use LinkButton, the BulletedList fires a Click event that you can react to on the server to perform the navigation. When you use HyperLink, the BulletedList doesn't fire the Click event—instead, it treats the text of each list item as a relative or absolute URL, and renders them as ordinary HTML hyperlinks. When the user clicks an item, the browser attempts to navigate to that URL.

If you set the DisplayMode to LinkButton, you can react to the Button.Click event to determine which item was clicked. Here's an example:

```
protected void BulletedList1_Click(object sender, BulletedListEventArgs e)
{
// Find the clicked item in the list.
// (You can't use the SelectedIndex property here because static lists
// don't support selection.)
string itemText = BulletedList1.Items[e.Index].Text;
```

```
Label1.Text = "You choose item" + itemText;
}
```

Figure 6-6 shows all the BulletStyle values that the BulletList supports. When you click one of the items, the list changes to use that BulletStyle.

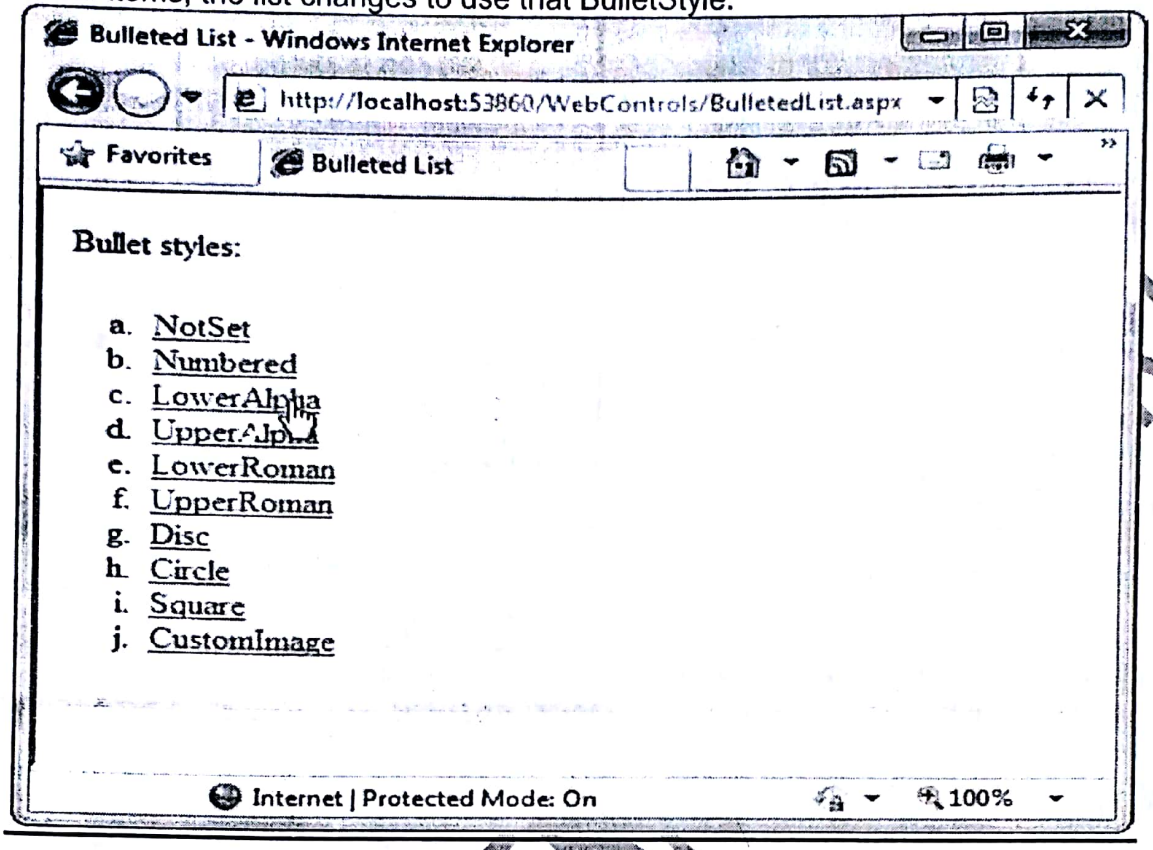


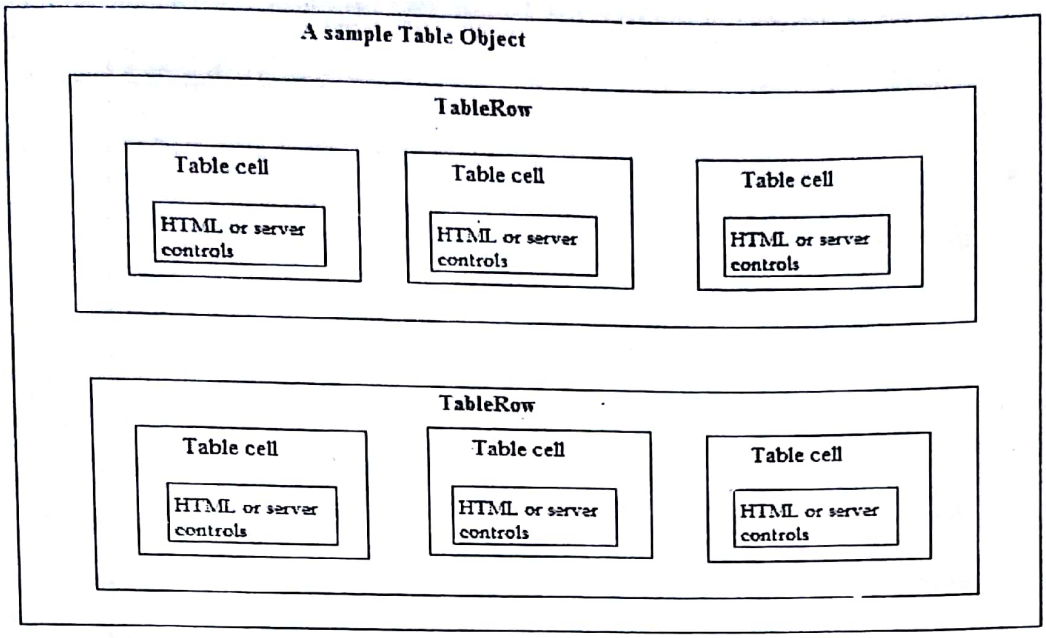
Figure 6-6. *Various BulletedList styles*

Table Controls:

The table object contains a hierarchy of objects. Each table object contains one or more TableRow objects. Each TableRow object contains one or more TableCell object and each TableCell object contains other ASP.NET controls that display information.

Figure:

Figure 6-7. Table control containment



If you are familiar with the HTML table tags you can show. When the user click the generate button the table is field dynamically with the sample data according to selected options.

The Web.config File:-

The web.config file uses a special XML format. Everything is nested in a root <configuration> element, which contains a <system.web> element. Inside the <system.web> element are separate elements for each aspect of configuration.

```
<?xml version ="1.0" encoding="utf-8"?>
<configuration>
   <system.web>
      <!—configuration section go here. -->
   </system.web>
</configuration>
```

You can include as few or as many configuration section as you want.

Example:

If you need to special error settings, you could add just the<customError>group. If you create an ASP.NET application in VS.NET, a web.config file is created for you with a basic skeleton showing you all the important section. Comments are added to each section, describing the purpose of various options.

```
<!—This is the format for an XML comment- ->
```

The following example of web.config file. The web.config file is case-sensitive, and uses a standard where the first word is always lowercase. That means you cannot write, <CustomErrors>instead of<customErrors>.

```
<?xml version ="1.0" encoding="utf-8"?>
<configuration>
    <system.web>
```

```
        <httpruntime/>
        <pages/>
        <compilation/>
        <CustomErrors/>
        <authentication/>
        <authorization/>
        <identity/>
        <trace/>
        <sessionstate/>
        <httphandlers/>
        <httpmodules/>
       <globalization/>
        <compilation/>

    </system.web>
</configuration>
```

Nested Configuration:

ASP.NET uses a multilayered configuration system that allows using different setting for different parts of your application. To use this technique, you need to create additional subdirectories inside your virtual directory. These subdirectories can contain their own web.config files with additional settings.

Subdirectories also inherit web.config settings from the parent directory.

Custom Settings in the Web.config files:

You can add some of your own settings to the web.config file, and use them with simple HelloWorld.aspx page.

You add custom settings to a special element called<appSettings>.Note that this is nested in the root <configuration>element, not the <system. Web> element, which contains the other groups of predefined settings.

```
<? XML version="1.0" encoding="utf-8"?>

<configuration>

  <appsettings>
   <!- -special application settings go here.- ->
  </appsettings>

 <system.web>
   <!- -Configuration section go here.- - >
 </system.web>

</configuration>
```

There are several reasons that you might want to use a special web.config setting:

1. **To make it easy to quickly switch between different modes of operation.**

You might create a special debugging variable. Your web pages could check for this variable, and if it is set to a specified value, output additional information to help you test the application.

2. **To centralize an important setting that needs to be used in many different pages.**

You could create a variable that contains a database connection string. Any page that needs to connect to the database could then retrieve this value and use it. As you will learn later in this book, it's good idea to always use the exact same database connections string in all your pages, as this ensures that SQL Server can reuse connections for different clients. The technical term for this time saving feature is connection pooling.

3. **To set some initial values. Depending on the operation, the user might be able to modify these values, but the web.config file could supply the default selection.**

Note
MyFirstWebApp application, there is currently no web.config file. That means that the default settings are used from the server machine.config file. Every web server has a single machine.config file that defines the default options.

The web.config files has several advantages over traditional asp configuration :-
1). It's never locked.
2). It's easily accessed and replicated.
3). It's easy to edit and understand.

Global.asax Application file:

The global.asax file allows you to write global application code. The global .asax file looks similar to a normal .aspx file, except for fact that it can't contain any HTML or ASP.NET tags. Instead, it contains event handling code that reacts to application or session events.

Example:

```
<script runat ="server">
  Public sub Application_On EndRequest( )
     Response.write("<hr>This page was served at "&_ DateTime.Now.Tostring())
  End Sub
</script>
```

This event handler writes footer at the bottom of the page with the date and time that the page was created.

Note:
Each ASP.NET application can have one global.asax file. Once you place it in the appropriate Virtual directory, ASP.NET recognizes it and uses it automatically.

Global .asax code behind:→

The global.asax file also supports code behind development. However, the code behind class does not inherit from the page class, because the global.asax file does not represent

a page. Instead, it inherits from the special System.Web.HttpApplication class. Similarly, the global.asax uses an application directive instead of a page directive.

Example:

```
<%@ Application Codebehind="Global.vb"  Inherits="GlobalApp" %>
```

Once again, ASP.NET always create a custom HttpApplication class to represent your Application when you first run it.

Application Events:

1) Application_Onstart
2) Application_OnEnd
3) Application_OnBeginRequest
4) Application_OnEndRequest
5) Session_OnStart
6) Session_Onend
7) Application_OnError

ASP.NET CLASSES

All web pages are instance of page class and all global.asax files are instances of the HttpApplication class. In ASP.NET, the response object is available because it's a part of the HttpApplication class.

When you inherit from Application, your class acquires all the features of the HttpApplication class. These include properties such as Request, Response, Session, Application, and Server. Each of these properties holds an instance of a special ASP.NET object.

For Example: HttpApplication.Response holds a reference to the HttpResponse object, which provides a method called write.

When you create an .aspx page, you are creating a class that inherits from ASP.NÉT page class. This class also provides properties such as Request, Response, Session, Application, along with a number of additional user interface related members.

Themes

Themes allow you to define a set of style details that you can apply to controls in multiple pages. However, unlike CSS, themes aren't implemented by the browser. Instead, ASP.NET processes your themes when it creates the page.

How Themes Work

All themes are application specific. To use a theme in a web application, you need to create a folder that defines it. This folder needs to be placed in a folder named App_Themes, which must be placed inside the top-level directory for your web application. An application can contain definitions for multiple themes, as long as each theme is in a separate folder. Only one theme can be active on a given page at a time.

To actually make your theme accomplish anything, you need to create at least one skin file in the

theme folder. A *skin file* is a text file with the .skin extension. ASP.NET never serves skin files directly— instead, they're used behind the scenes to define a theme.

A skin file is essentially a list of control tags—with a twist. The control tags in a skin file don't need to completely define the control. Instead, they need to set only the properties that you want to standardize. For example, if you're trying to apply a consistent color scheme, you might be interested in setting only properties such as ForeColor and BackColor. When you add a control tag for the ListBox in the skin file, it might look like this:

```
<asp:ListBox runat="server" ForeColor="White" BackColor="Orange"/>
```

The runat="server" portion is always required. Everything else is optional. You should avoid setting the ID attribute in your skin, because the page that contains the ListBox needs to define a unique name for the control in the actual web page.
It's up to you whether you create multiple skin files or place all your control tags in a single skin file.

Both approaches are equivalent, because ASP.NET treats all the skin files in a theme directory as part of the same theme definition. Often, it makes sense to put the control tags for complex controls (such as the data controls) in separate skin files. Figure 12-9 shows the relationship between themes and skins in more detail.

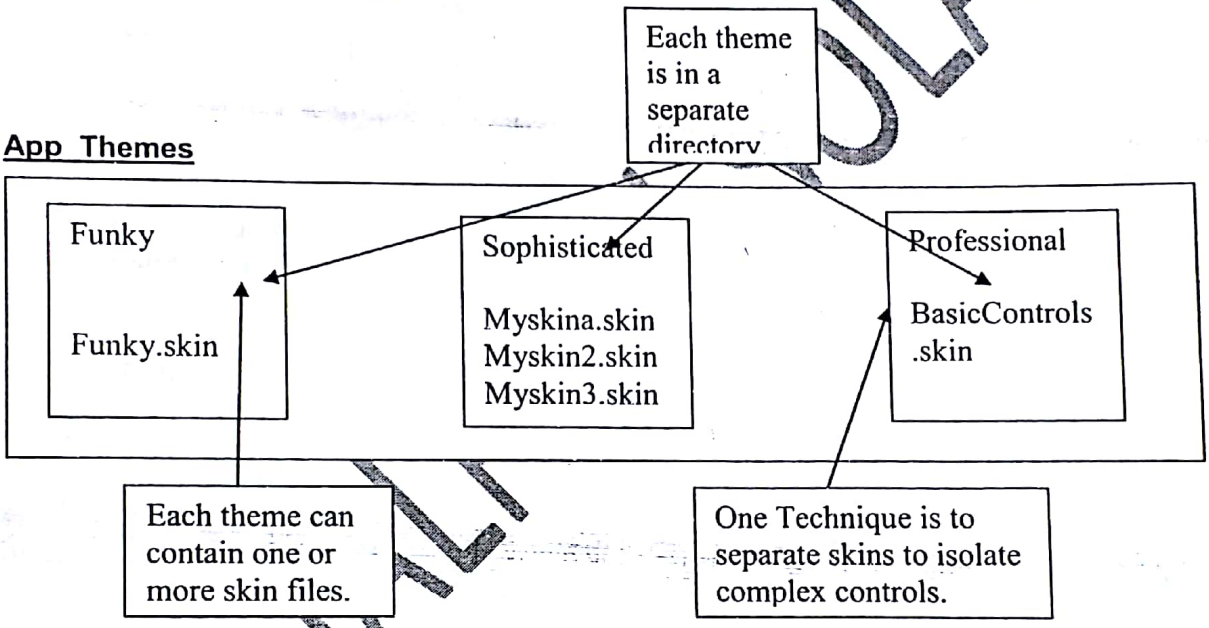


***Figure 12-9.** Themes and skins*

ASP.NET also supports global themes. These are themes you place in the c:\Inetpub\wwwroot\aspnet_client\system_web\v2.0.50727\Themes folder. However, it's recommended that you use local themes, even if you want to create more than one website that has the same theme. Using local themes makes it easier to deploy your web application, and it gives you the flexibility to introduce site-specific differences in the future. If you have a local theme with the same name as a global theme, the local theme takes precedence, and the global theme is ignored. The themes are *not* merged together.

Applying a Simple Theme

To add a theme to your project, select Website ➤ Add New Item, and choose Skin File. Visual Studio will warn you that skin files need to be placed in a subfolder of the App_Themes folder and ask you whether that's what you intended. If you choose Yes, Visual Studio will create a folder with the same name as your theme file.

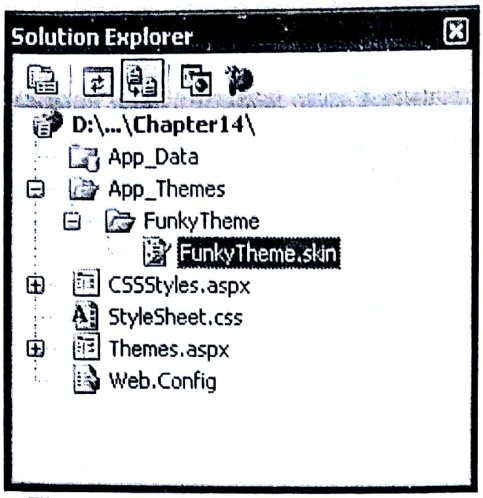


Figure 12-10. *A theme in the Solution Explorer*

Here's a sample skin that sets background and foreground colors for several common controls:

```
<asp:ListBox runat="server" ForeColor="White" BackColor="Orange"/>
<asp:TextBox runat="server" ForeColor="White" BackColor="Orange"/>
<asp:Button runat="server" ForeColor="White" BackColor="Orange"/>
```

(1). To apply the theme in a web page, you need to set the Theme attribute of the Page directive to the folder name for your theme. (ASP.NET will automatically scan all the skin files in that theme.)

```
<%@ Page Language="C#" AutoEventWireup="true" ... Theme="FunkyTheme" %>
```

(2).You can make this change by hand, or you can select the DOCUMENT object in the Properties window at design time and set the Theme property (which provides a handy drop-down list of all your web application's themes). Visual Studio will modify the Page directive accordingly.

When you apply a theme to a page, ASP.NET considers each control on your web page and checks your skin files to see whether they define any properties for that control. If ASP.NET finds a matching tag in the skin file, the information from the skin file overrides the current properties of the control. (And if you have a CSS style that affects the same control, it can override the theme, because the theme is set on the server, and the style is applied after, in the browser. However, it's best to avoid this sort of overlapping formatting when possible.)

Figure 12-11 shows the result of applying the FunkyTheme to a simple page. You'll notice that conflicting settings (such as the existing background for the list box) are overwritten.

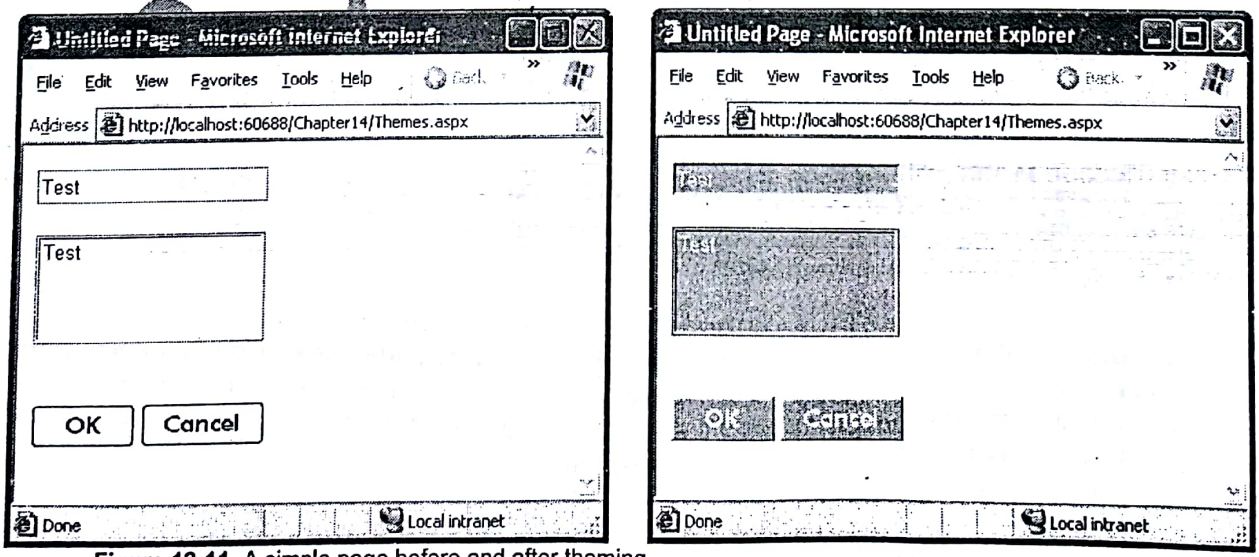


Figure 12-11. A simple page before and after theming

APPLYING A THEME TO AN ENTIRE WEBSITE

(3).Using the Page directive, you can bind a theme to a single page. However, you might decide that your theme is ready to be rolled out for the entire web application. The cleanest way to apply this theme is by configuring the <pages> element in the web.config file for your application, as shown here:

```
<configuration>
<system.web>
<pages theme="FunkyTheme">
...
</pages>
</system.web>
</configuration>
```

If you want to use the style sheet behavior so that the theme doesn't overwrite conflicting control properties, use the StyleSheetTheme attribute instead of Theme:

```
<configuration>
<system.web>
<pages styleSheetTheme="FunkyTheme">
...
</pages>
</system.web>
</configuration>
```

Master Page Basics

If you decide to update your navigation bar or change its position later, you'll need to modify every web page to apply the same change.
So how can you deal with the complexity of different pages that need to look and act the same? One option is to subdivide the page into *frames*. Frames are an HTML feature that lets the browser show more than one web page alongside another. Unfortunately, frames have problems of their own, including that each frame is treated as a separate document and requested separately by the browser.

This makes it difficult to create code that communicates between frames. A better choice is to use ASP.NET's master pages feature, which allows you to define page templates and reuse them across your website.

Master pages are similar to ordinary ASP.NET pages. Like ordinary pages, master pages are text files that can contain HTML, web controls, and code. However, master pages have a different file extension (.master instead of .aspx), and they can't be viewed directly by a browser. Instead, master pages must be used by other pages, which are known as *content pages*. Essentially, the master page defines the page structure and the common ingredients. The content pages adopt this structure and just fill it with the appropriate content.

For example, if a website such as http://www.amazon.com were created using ASP.NET, a single master page might define the layout for the entire site. Every page would use that master page, and as a result, every page would have the same basic organization and the same title, footer, and so on. However, each page would also insert its specific information, such as product descriptions, book reviews, or search results, into this template.

A Simple Master Page and Content Page

To see how this works, it helps to create a simple example. To create a master page in Visual Studio, select Website ➤ Add New Item from the menu. Select Master Page, give it a file name (such as SiteTemplate.master, used in the next example), and click Add. When you create a new master page in Visual Studio, you start with a blank page that includes a ContentPlaceHolder control (see Figure 12-14). The ContentPlaceHolder is the portion of the master page that a content page can change. Or, to look at it another way, everything else that's set in the master page is unchangeable in a content page. If you add a header, that header appears in every content page. If you want to give the content page the opportunity to supply content in a specific section of the page, you need to add a ContentPlaceHolder.

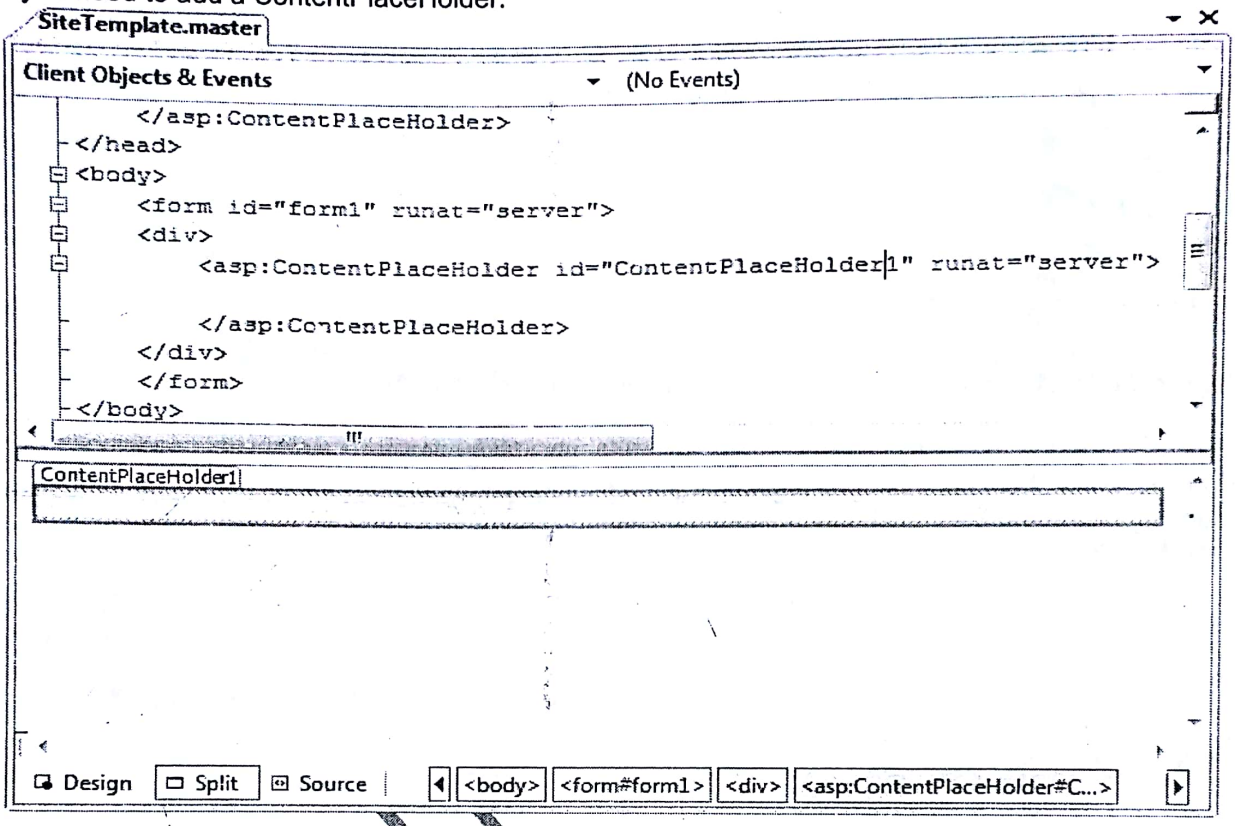


Figure 12-14. *A new master page*

When you first create a master page, you'll start with two ContentPlaceHolder controls. **One is** defined in the <head> section, which gives content pages the add page metadata, such as search keywords and style sheet links. **The second,** more important ContentPlaceHolder is defined in the <body> section, and represents the displayed content of the page. It appears on the page as a faintly outlined box. If you click inside it or hover over it, the name of ContentPlaceHolder appears in a tooltip, as shown in Figure 12-14.

To make this master page example more practical, try adding a header before the ContentPlaceHolder (using an <img> tag) and a footer after it (using some static text), as shown in Figure 12-15. You'll notice that the content area of the page looks very small, but this appearance is deceptive. The content section will expand to fit the content you place inside.

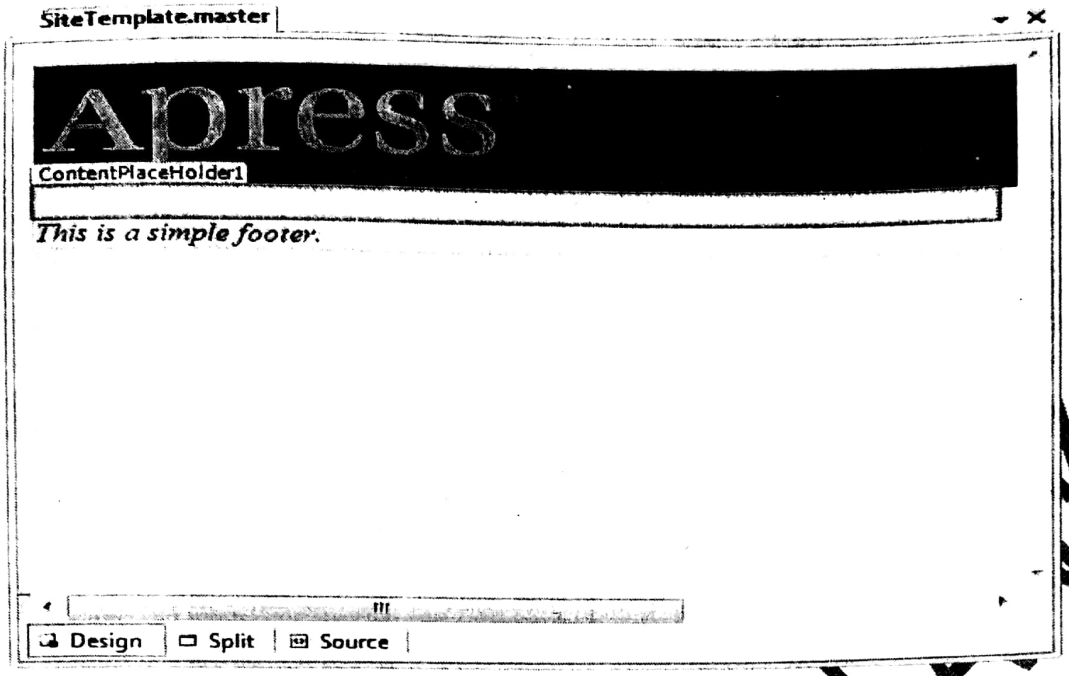


Figure 12-15. *A simple master page with a header and footer*

Now you're ready to create a content page based on this master page. To take this step, select Website ➤ Add New Item from the menu. Select Web Form, and choose to select a master page (see Figure 12-16). Click Add. When you're prompted to choose a master page, use the one you created with the header and footer.

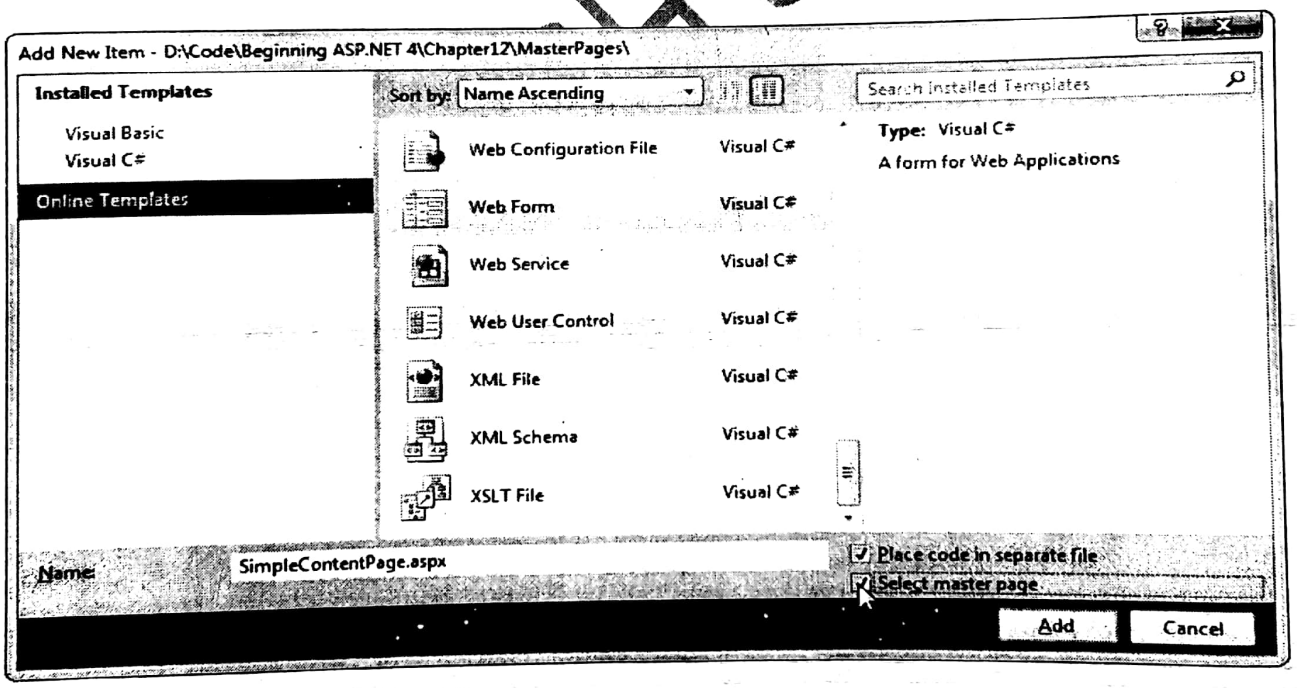


Figure 12-16. Creating a content page

Now you'll see something a little more interesting. Your content page will have all the elements of the master page, but the elements will be shaded in gray, indicating that you can't select or change them in any way. However, you can add content or drag and drop new controls into the ContentPlaceHolder region to create a page like the one shown in Figure 12-17. In fact, this is the only editable portion of your page.

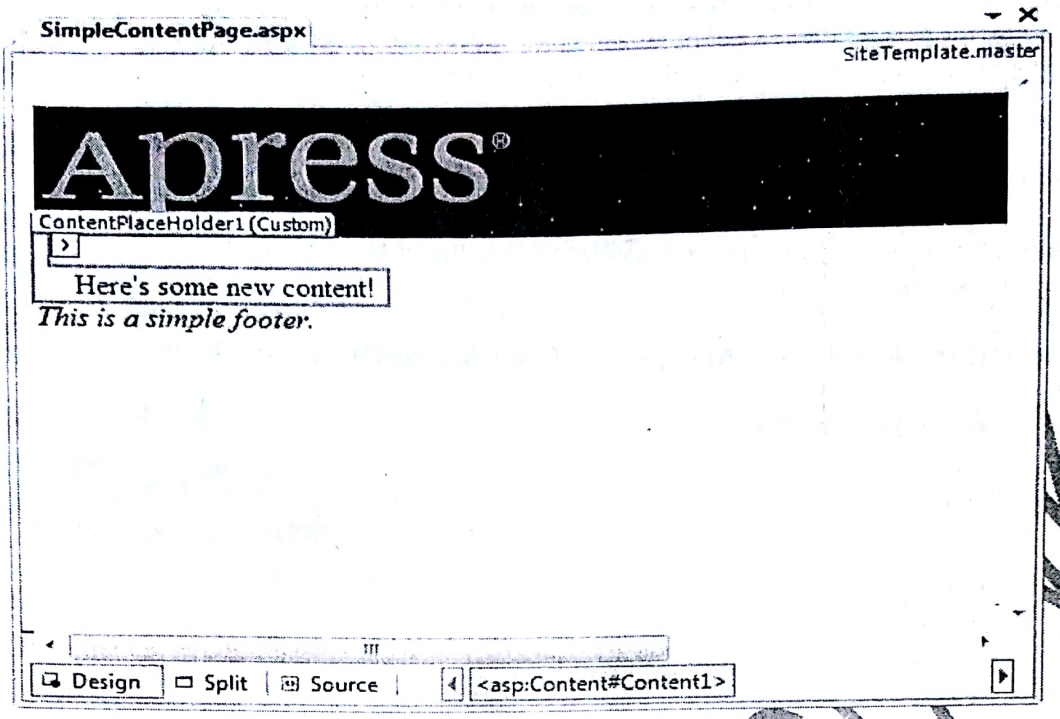


Figure 12-17. *A simple content page at design time*

The ContentPlaceHolder section will expand or collapse to fit the content you place in it. If you've added volumes of text, the footer won't appear until the end. If you've included only a single line of text, you'll see something more compact, as in Figure 12-17. To get a clearer look at your web page, you can run it in the browser. Figure 12-18 shows the content page that's being designed in Figure 12-17.

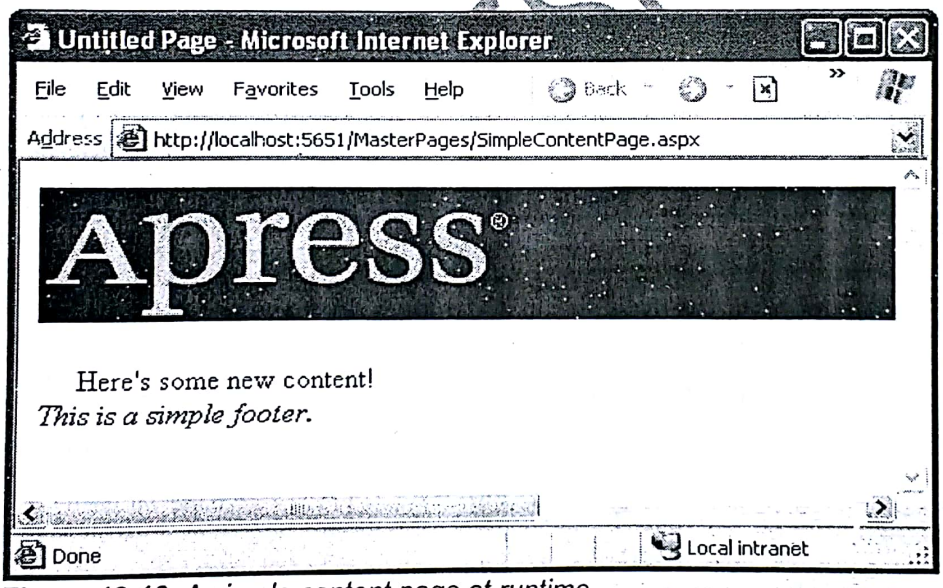


Figure 12-18. *A simple content page at runtime*

The real magic starts when you create multiple pages that use the same master page. Now each page will have the same header and footer, creating a seamless look across your entire website. How Master Pages and Content Pages Are Connected Now that you've seen a master page example, it's worth taking a look behind the scenes to see how you implement the master page.

When you create a master page, you're building something that looks much like an ordinary

ASP.NET web form. The key difference is that although web forms start with the Page directive, a master page starts with a Master directive that specifies the same information. Here's the Master directive for the simple master page shown in the previous example:

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

The ContentPlaceHolder is less interesting. You declare it like any ordinary control. Here's the complete code for the simple master page:

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="SiteTemplate.master.cs"
Inherits="SiteTemplate" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Untitled Page</title>
</head>
<body>
<form id="form1" runat="server">
<img src="apress.jpg" /><br />
<asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">
</asp:ContentPlaceHolder>
<i>This is a simple footer.</i>
</form>
</body>
</html>
```

Note For simplicity's sake, this example doesn't include a ContentPlaceHolder in the <head> section. Although this ContentPlaceHolder isn't required (and it's isn't used in this example), Visual Studio adds it by default to all new master pages.

When you create a content page, ASP.NET links your page to the master page by adding an attribute to the Page directive. This attribute, named MasterPageFile, indicates the associated master page. Here's what it looks like:

```
<%@ Page Language="C#" MasterPageFile="~/SiteTemplate.master" AutoEventWireup="true"
CodeFile="SimpleContentPage.aspx.cs" Inherits="SimpleContentPage" Title="Untitled Page" %>
```

Notice that the MasterPageFile attribute begins with the path ~/ to specify the root website folder. Using the ~/ syntax is better because it indicates unambiguously where ASP.NET can find your master page.

```
<%@ Page Language="C#" MasterPageFile="~/SiteTemplate.master" AutoEventWireup="true"
CodeFile="SimpleContentPage.aspx.cs" Inherits="SimpleContentPage" Title="Content Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
<br />
Here's some new content!
<br />
</asp:Content>
```

For ASP.NET to process this page successfully, the ContentPlaceHolderID attribute in the <Content> tag must match the ContentPlaceHolder specified in the master page exactly. This is how ASP.NET knows where it should insert your content in the master page template.

Event in ASP.NET:

Events are generated in response to a user action.

Example: When user clicks on button, then click event is generated. Events are messages that contain details about the user action, such as cause of the event and the control on which the event occurred.
In ASP.NET all events pass two arguments.....

1. Object
2. Event Object

1. Object:
This is the first argument and is an instance of system .object class. This represents the control that raised the events.

2. Event Object:
This i is an instance of system .EventArgs class. This contain information about the event is generated.

❖ ASP.NET Server Control Event Model:

In ASP.NET, Events can be generated either **on the server side** or **on the client side** but the processing of events happens only on the server.
ASP.NET supports only two types of Events:

- PostBack Events
- NonPostBack Events

PostBack Events:
Events that can cause the roundtrip to the server are called postback event.

Example: The click event of a button causes a postback.

NonPostBack Events:

Events that do not cause the roundtrip to the server.

Example: The change event is a NonPostBack Event.

NonPostBack Events are cached and send to the server. during a postback via the post system. the Autopostback property of a web server control can be used to generate a postback event for a Nonpostback event.

❖ AutoPostBack and Web Control Events:

One of the main limitations of Html server controls are their limited set of useful events-exactly two. HTML controls that trigger a postback, such as buttons, raise a ServerClick event. Input controls provide a ServerChange event that won't be detected until the page is posted back.

Server controls are really an ingenious illusion. You will recall that the code in an ASP.NET page is processed on the server. It's then sent to the user as ordinary HTML webpage.

The question is, how can you write server code that will react to an event that occurs on the client?

The answer is a new innovation called the automatic postback.
Automatic postback submits a page back to the server when it detects a specific user action. This gives your code the chance to run again and create a new, updated page. Controls that support automatic postbacks include almost all input controls. A basic list of web controls and events are in the following table: - .

Web Control Events

Event	Web Controls that Provide It	Always Posts Back
Click	Button, ImageButttton	True
TextChange	Textbox	False
CheckChanged	Checkbox, Radio Button	False
SelectedIndexChanged	DropDownList, Listbox, CheckBoxList, RadioButtonList	False

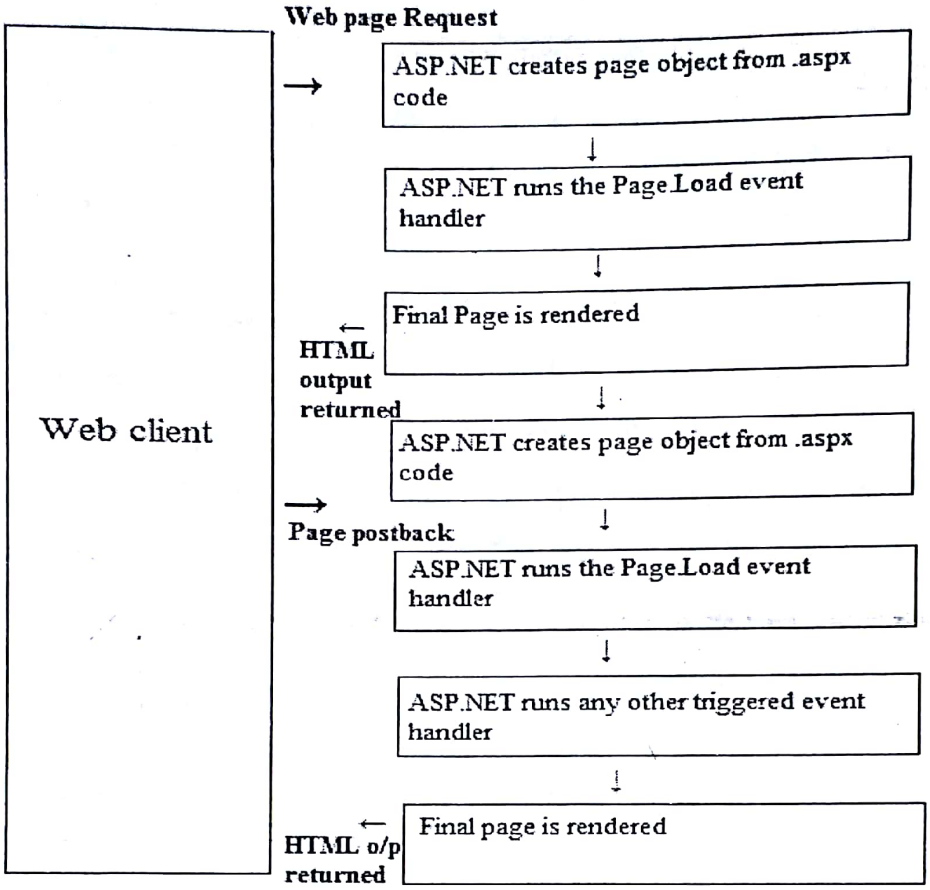
The Page Class

Every web page is a custom class that inherits from the System.Web.UI.Page class. By inheriting from this class, your page class acquires a number of properties that your code can use. These include properties for enabling caching, validation, and tracing.

Some of fundamental properties including the traditional built in objects that you probably used in asp programming, such as Response, Request, and session.

Properties	Description
Application and Session	These collections are used to hold state information on the server.
Cache	This collection allows you to store objects for reuse in other pages or for other clients.
Controls	Provides a collection of all the controls contained on the web page. You can also use the methods of this collection to add new controls dynamically.
EnableViewState	When set to false, this overrides the EnableViewState property of the contained controls, ensuring that no controls will maintain state information.
IsPostBack	This Boolean property indicates whether this is the first time the page is being run(false),or whether the page is being resubmitted in response to a control event, typically with stored viewstate information(true).This property is often used in the page. Load event handler, ensuring that basic setup is only performed once for controls that maintain viewstate.
Request	Refers to an HttpRequest object that contains information about the current web request, including client certificates, cookies and values submitted through HTML form elements. It supports the same features as the built in ASP Request object.
Response	Refers to an HTTPRespnse object that allows you to set the web response or redirect the user to another web page. It supports the same features as the built in ASP response object, although it's used much less in .NET development.
Server	Refers to an HTTPServer utility object that allows you to perform some miscellaneous tasks, such as URL and HTML encoding. It supports the same features as the built in ASP server object.
User	If the user has been authenticated, this property will be intialized with user information. This property is described in more detail when we examine security.

The Page Processing Sequence



If you want to capture a change event for a web control, you need to set its AutoPostBack property to True. That means that when the user clicks a radio button or checkbox, the page will be resubmitted to the server. The server examines the page, loads all the current information, and then allows your code to perform some extra processing before returning the page back to the user.

In other words, every time you need to update the web page, it is actually being sent to the server and recreated. However, ASP.NET makes this process so transparent that your code can treat your web page like a continuously running program that fires events.

Some events, such as the Click clicked, the button posts back the page. This is a basic convention of HTML forms. However, other actions *do* cause events but *don't* trigger a postback. An example is when the user changes the text in a text box (which triggers the TextChanged event) or chooses a new item in a list (the SelectedIndexChanged event). You might want to respond to these events, but without a postback you code has no way to run.

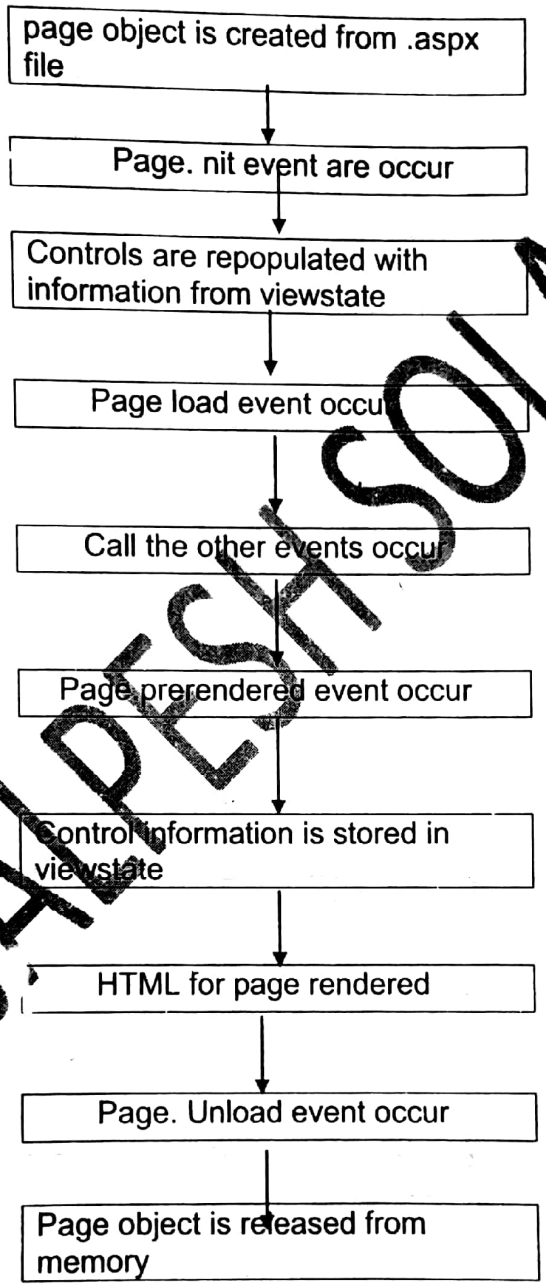
ASP.NET handles this by giving you two options

• You can wait until the next postback to react to the event. For example, imagine you want to react to the SelectedIndexChanged event in a list. If the user selects an item in a list, nothing happens immediately. However, if the user then clicks a button to post back the page, *two* events fire: Button.Click followed by ListBox.SelectedIndexChanged. And if you have several controls, it's quite

possible for a single postback to result in several change events, which fire one after the other, in an undetermined order.

• You can use the *automatic postback* feature to force a control to post back the page immediately when it detects a specific user action. In this scenario, when the user clicks a new item in the list, the page is posted back, your code executes, and a new version of the page is returned. The option you choose depends on the result you want.

PostBack Processing Sequence:



This postback system is not ideal for all events.

Example:

Some events that you may be familiar with from windows programs, such as mouse movement,events or Keypress events, are not practical in an ASP.NET application. Resubmit the page every time a key is pressed or the mouse is moved would make the application slowed and unresponsive.

The Page Life Cycle

To understand how web control events work, you need to have a solid understanding of the page life cycle. Consider what happens when a user changes a control that has the AutoPostBack property set to true:

1. On the client side, the JavaScript __doPostBack function is invoked, and the page is resubmitted to the server.
2. ASP.NET re-creates the Page object using the .aspx file.
3. ASP.NET retrieves state information from the hidden view state field and updates the controls accordingly.
4. The Page.Load event is fired.
5. The appropriate change event is fired for the control. (If more than one control has been changed, the order of change events is undetermined.)
6. The Page.PreRender event fires, and the page is rendered (transformed from a set of objects to an HTML page).
7. Finally, the Page.Unload event is fired.
8. The new page is sent to the client.

To watch these events in action, it helps to create a simple event tracker application. All this application does is write a new entry to a list control every time one of the events it's monitoring occurs.

This allows you to see the order in which events are triggered. Figure 6-14 shows what the window looks like after it's been loaded once, posted back (when the text box content was changed), and posted back again (when the check box state was changed).

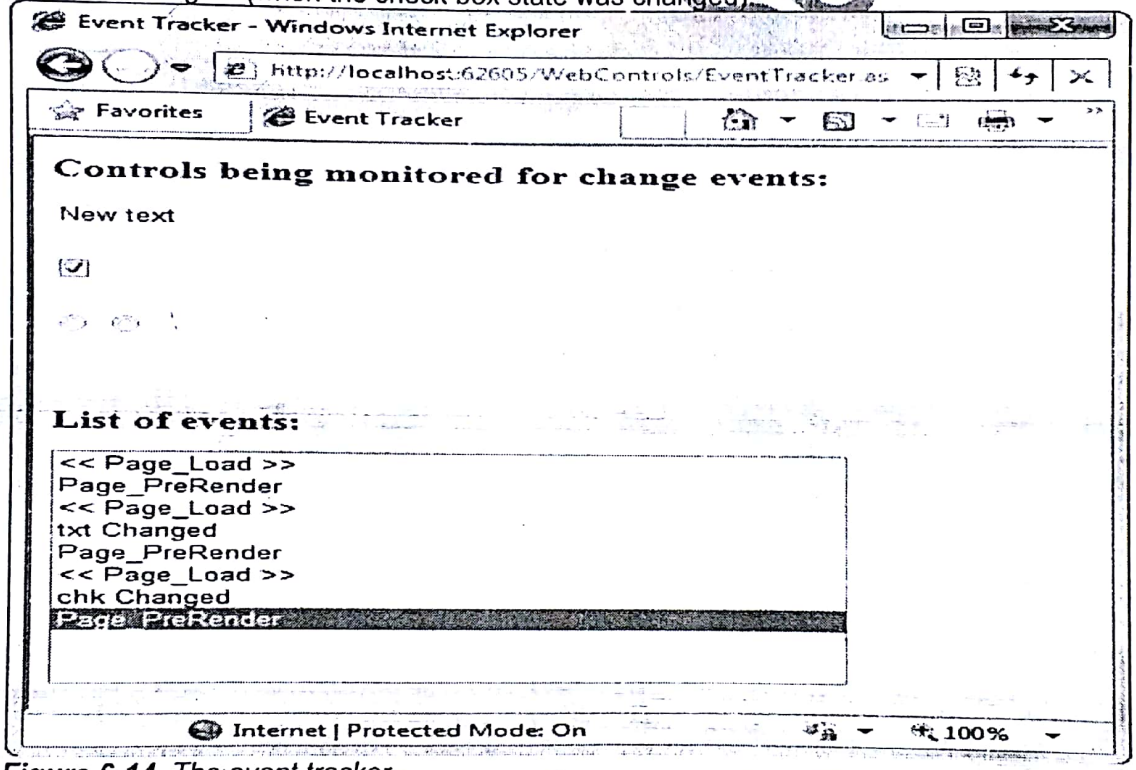


Figure 6-14. *The event tracker*

Listing 6-1 shows the markup code for the event tracker, and Listing 6-2 shows the code-behind class that makes it work.

Listing 6-1. *EventTracker.aspx*

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="EventTracker.aspx.cs"
Inherits="EventTracker" %>
<head runat="server">
<title>Event Tracker</title>
</head>
```

```
<body>
<form runat="server">
<div>
<h1>Controls being monitored for change events:</h1>
<asp:TextBox ID="txt" runat="server" AutoPostBack="true" OnTextChanged="CtrlChanged" />
<br /><br />
<asp:CheckBox ID="chk" runat="server" AutoPostBack="true"
OnCheckedChanged="CtrlChanged"/>
<br /><br />
<asp:RadioButton ID="opt1" runat="server" GroupName="Sample" AutoPostBack="true"
OnCheckedChanged="CtrlChanged"/>
<asp:RadioButton ID="opt2" runat="server" GroupName="Sample"
AutoPostBack="true" OnCheckedChanged="CtrlChanged"/>
<h1>List of events:</h1>
<asp:ListBox ID="lstEvents" runat="server" Width="355px"
Height="150px" /><br />
<br /><br /><br />
</div>
</form>
</body>
</html>
```

Listing 6-2. *EventTracker.aspx.cs*

```
public partial class EventTracker : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{
Log("<< Page_Load >>");
}
protected void Page_PreRender(object sender, EventArgs e)
{
// When the Page.PreRender event occurs, it is too late
// to change the list.
Log("Page_PreRender");
}
protected void CtrlChanged(Object sender, EventArgs e)
{
// Find the control ID of the sender.
// This requires converting the Object type into a Control class.
string ctrlName = ((Control)sender).ID;
Log(ctrlName + " Changed");
}
private void Log(string entry)
{
lstEvents.Items.Add(entry);
// Select the last item to scroll the list so the most recent
// entries are visible.
lstEvents.SelectedIndex = lstEvents.Items.Count - 1;
```

********** END**********