Unit 6

6.0 Web Form Fundamentals

(Text Book Chapter 05)

➢6.0 Web Form Fundamentals

6.1 ASP.NET Application Model

ASP.NET provides a unified Web development model that includes the services necessary for you to build enterprise-class Web applications. While ASP.NET is largely syntax compatible with Active Server Pages (ASP), it provides a new programming model and infrastructure that allow you to create a powerful new class of applications. ASP.NET is part of the .NET Framework and allows you to take full advantage of the features of the common language runtime, such as type safety, inheritance, language interoperability, and versioning.

It may be that you find it difficult to comprehend what's going on inside of web applications. Web applications are not like normal stand alone applications that run on a machine - normally an .exe file.

ASP.NET applications are configured into several web pages. The web page you on your browser is the "RESULT" after the a application was executed by the Web Server. Therefore a Web application is also called a "Server Side Application"

A Web Application has its own domain. Within each applications' domain, there are several components – Web Pages, Web Services, Configuration Files and Session Data. (See Figure 6.1-1)

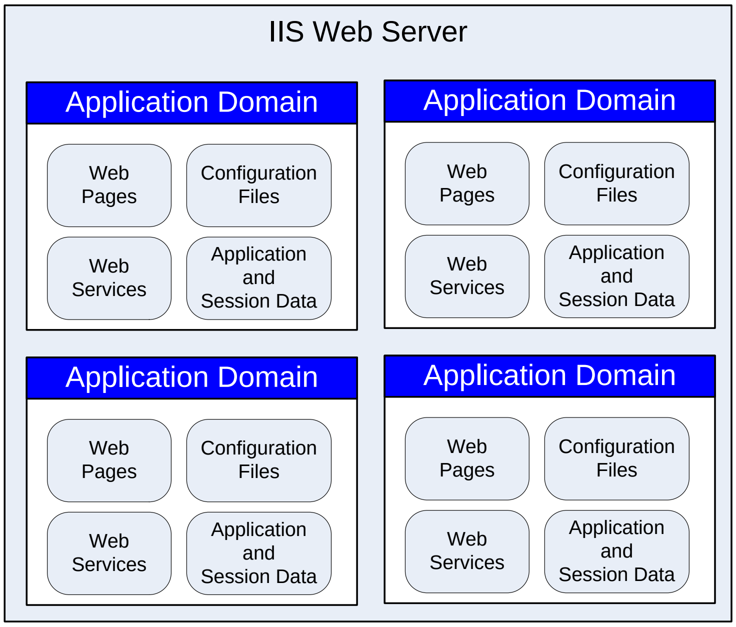


Figure 6.1-1 –ASP.NET Application Domains

Each of these components contains it's own set of files that do specific jobs. The chart in Figure 6.2 next page explains the type of files in each of these components and what they do.

➢6.0 Web Form Fundamentals

File Type	Description
.aspx files	.aspx file is a server script file like .asp or .php (if you are familiar with these). These files require appropriate hosts to interpret its contents and dynamically generate HTMLs. .aspx files require ASP.NET runtime which is typically on microsoft windows web server with .NET framework installed, IIS (Internet Information Services). If the .aspx file is hosted on an IIS web server, which will be on windows (XP, Server etc). So these are essentially Web pages typically a mix of HTML content as well as some code which generates the HTML content dynamically.
.ascx files	.ascx files are associated with user controls. **user controls** can be requested from a server that hosts an ASP.NET Web application. The file must be called from a Web Forms page. Internally, the UserControl class is associated with files that have .ascx extensions. These files are compiled at run time as UserControl objects and cached in server memory. User controls are contained in ASP.NET Web Forms pages, and offer Web developers an easy way to capture commonly used Web UI.
.asmx files	.asmx file serves as the end point for the **web service.** When a client calls the .asmx file, the ASP.NET runtime processes the file. Usually, each .asmx page has a language directive at the top of the page, as displayed here: <%@ WebService Language="C#" Class= IFCEBrokerage.SecuritiesExchange" %> The first time the web service is called, the ASP.NET runtime uses the Language directive to compile the code
web.config	Web.config file, as it sounds, is a configuration file for the Asp .net web application. An Asp .net application has one web.config file which keeps the configurations required for the corresponding application. Web.config file is written in XML with specific tags having specific meanings.
Global.asax	The Global.asax file (also known as the ASP.NET application file) is an optional file that is located in the application's root directory and is the ASP.NET counterpart of the Global.asa of ASP. This file exposes the application and session level events in ASP.NET and provides a gateway to all the application and the session level events in ASP.NET. This file can be used to implement the important application and session level events such as Application_Start, Application_End, Session_Start, Session_End, etc. This article provides an overview of the Global.asax file, the events stored in this file and how we can perform application wide tasks with the help of this file.
.cs files	These are C# source code files. Normally these are used for "code Behind" files. These files are also part of the 'Business Logic' which normally separates from the presentation logic.

Figure 6.1-2 - Web Application Domain File Types

➤6.0 Web Form Fundamentals

6.2 ASP.NET Web Application Directory Structure

ASP.NET uses a new set of predefined **Directory names for special work.** In general, the ASP.NET directory structure can be determined by the developer's preferences. But there are a few reserved directory names.

Here is the list of special directory names used by ASP.NET.

Folder Name	Description
bin	folder is used to store the compiled assemblies of the application and also the referenced assemblies which are private assemblies (*.dll files). Assemblies in the Bin folder are automatically referenced by your application.
App_Code	is the directory used to store the raw code. The classes in this folder are automatically compiled into one assembly which is accessible in the code in every page in the site.
App_GlobalResources	holds the resx files for the localized resource available to every page in the site.
App_LocalResources	folder contains the localized resource files for each page in the site.
App_WebReferences	folder is used to store the discovery and WSDL files of the references of the web services consumed by site.
App_Data	is the default director for the data storage or database like Access, SQL server express etc. This directory has write permission so that data can be inserted in the data.
App_Themes	folder is used to store the themes of the site.

Figure 6.2-1 –ASP.NET Directories

Example Directory is shown below.

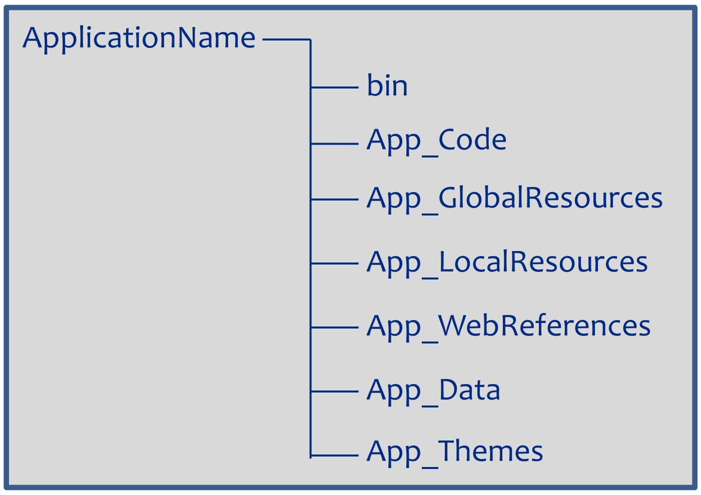


Figure 6.2-2 –Example ASP.NET Directory

6.3 ASP.NET Server Controls

ASP.NET Web server controls are objects on ASP.NET Web pages that run when the page is requested and that render markup to the browser. Many Web server controls are similar to familiar HTML elements, such as buttons And text boxes. Other controls encompass complex behavior, such as a calendar controls, and controls that you can use to connect to data sources and display data.

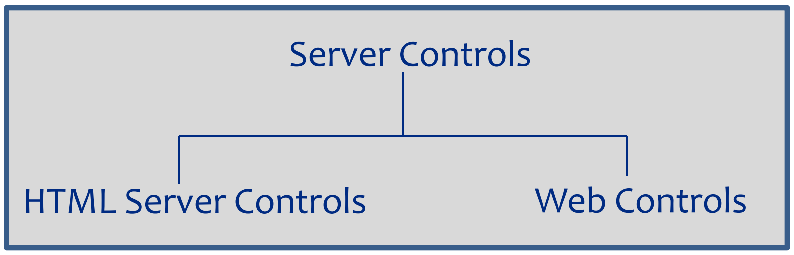


Figure 6.3-1 –ASP.NET Server Control Types

6.3.1 HTML Server Controls

HTML server controls are HTML elements (or elements in other supported markup, such as XHTML) containing attributes that make them programmable in server code. By default, HTML elements on an ASP.NET Web page are not available to the server. Instead, they are treated as opaque text and passed through to the browser. However, by converting HTML elements to HTML server controls, you expose them as elements you can program on the server.

The object model for HTML server controls maps closely to that of the corresponding elements. For example, HTML attributes are exposed in HTML server controls as properties.

Any HTML element on a page **can be converted to an HTML server control by adding the attribute runat="server".** During parsing, the ASP.NET page framework creates instances of all elements containing the **runat="server"** attribute. If you want to reference the control as a member within your code, **you should also assign an id attribute to the control**.

The page framework provides predefined HTML server controls for the HTML elements most commonly used dynamically on a page: the form element, the input elements (text box, check box, Submit button), the select element, and so on. These predefined HTML server controls share the basic properties of the generic control, and in addition, each control typically provides its own set of properties and its own event.

Note: All HTML server controls must be within a <form> tag with the runat="server" attribute!

Note: ASP.NET requires that all HTML elements must be properly closed and properly nested.

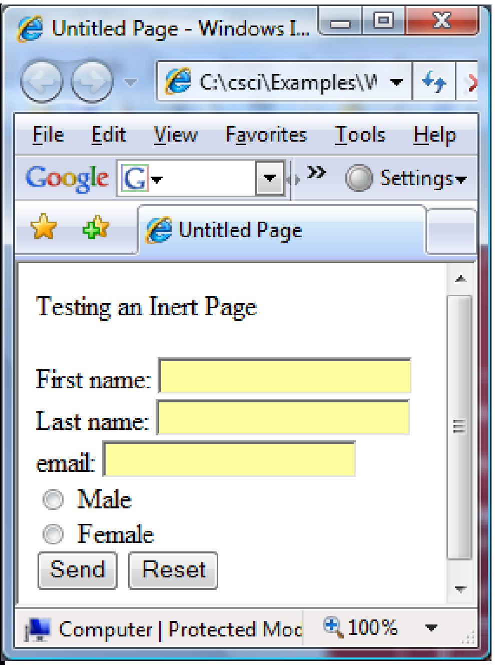
6.3-2 HTML to ASP.NET Conversion

Inert HTML pages can be converted to ASP.NET HTML Server Controls pretty easily. Let's look at an example:

The following code is just pure HTML and is inert. The result is shown in Figure 6.3-2

```
<!-- Example 6.3-2 -->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
   <title>Untitled Page</title>
</head>
<body>
   Testing an Inert Page
<FORM method="post">
   <P>
   First name: <INPUT type="text" name="firstname"><BR>
   Last name: <INPUT type="text" name="lastname"><BR>
   email: <INPUT type="text" name="email"><BR>
   <INPUT type="radio" name="sex" value="Male"> Male<BR>
   <INPUT type="radio" name="sex" value="Female"> Female<BR>
   <INPUT type="submit" value="Send"> <INPUT type="reset">
   </P>
 </FORM>
</Body>
</html>
```

Example 6.3-1 – Traditional HTML Controls



Testing an Inert Page

First name:
Last name:
email:
Male
Female
Send Reset

Figure 6.3-2 – Outputs

➢6.0 Web Form Fundamentals

First step to convert the **HTML code to HTML Server Control.** The converted code shall have the following line at the top of the file. This is the standard line inserted by Visual Studio.

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

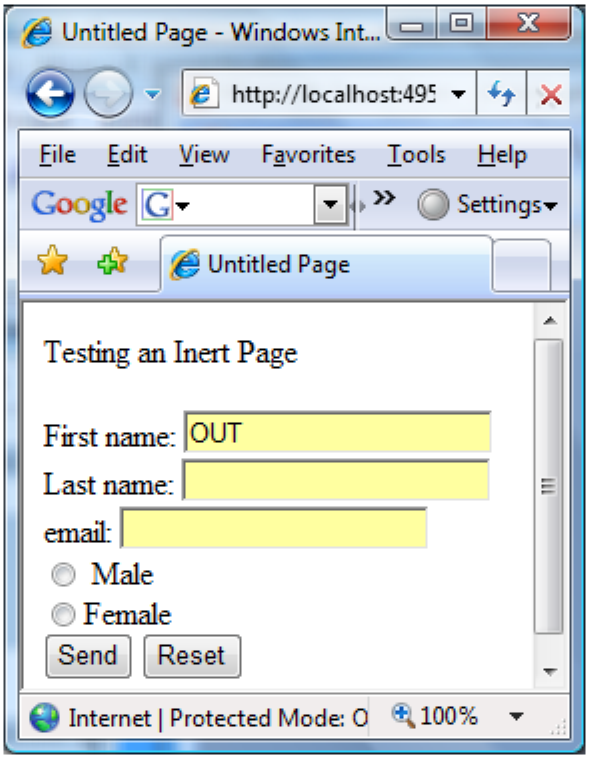
Modified code shown below.

```
<!-- Example 6.2 -->
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="InertToLife._Default" %>

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

<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
   <title>Untitled Page</title>
</head>
<body>
   Testing an Inert Page
<form runat="server">
   <p>
   First name: <input type="text" ID="FirstName"  runat="server"/> <br />
   Last name: <input type="text" ID="LastName" runat="server"/> <br />
   email: <input type="text" ID="Email" runat="server"/> <br />
   <input type="radio"  value="Male"  ID="rbMale" runat="server" /> Male <br />
   <input type="radio"  value="Female" ID="rbFeMale" runat="server" />Female <br />
   <input type="submit" value="Send" ID="btnSend" OnServerClick='ClickedSubmit' runat="server" />
   <input type="reset" value="Reset" id="btnReset" runat="server" /> <br />
   </p>
 </form>
</body>
</html>
```

Example 6.3-2 – HTML Server Controls

See next page for the rest of the code.

➢6.0 Web Form Fundamentals


Untitled Page - Windows Int...
http://localhost:495
File Edit View Favorites Tools Help
Google
Settings
Untitled Page
Internet | Protected Mode: O
100%

Testing an Inert Page

First name: OUT
Last name:
email:
Male
Female
Send Reset

```
//Example 6.3
using System;
using System.Collections;
.
.
using System.Xml.Linq;

namespace InertToLife
{
 public partial class _Default : System.Web.UI.Page
 {
  protected void Page_Load(object sender, EventArgs e)
  {
  }
 }

 public partial class _Default : System.Web.UI.Page
 {
  protected void ClickedSubmit(object sender, EventArgs e)
  {
   FirstName.Value = "OUT";
  }
 }
}
```

Figure 6.3-3 – Outputs Example 6.3-3 – Code Behind

The ability to process the output when the "Send" button was clicked is taken care of by the **"Code Behind"** C# code show above. As you see the "ClickedSubmit method is putting the "OUT" value in the "First Name" text box. This ability was added into the original HTML file by making the FirstName HTML control an HTML Server Control.

6.3.3 The Source View

After you ran the example in the previous page, you can use the right mouse button on the page and select the "View Source" option to view the response the browser received from the Web Server. Here is a dump of the source. (See next page example 6.3.4)

Notice that the source you see is not the source you typed while building the application. Several attributes are stripped out. There is simply no meaning at all to show that in the response. Response needs the result. Several hidden fields are added. There fields stores information in a compressed form. These are the states abut each component.

➢6.0 Web Form Fundamentals

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
   <title>Untitled Page</title>
</head>
<body>
   Testing an Inert Page
<form name="ctl00" method="post" action="Default.aspx" id="ctl00">
<div>
<input type="hidden" name="__EVENTTARGET" id="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" id="__EVENTARGUMENT" value="" />
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUKMTA2NDE0NjYwN2QYAQUeX19Db250cm9sc1JlcXVpcmVQb3N0QmFja0tleV9fFgIFBnJiTWFsZQUIcmJGZ
U1hbGXDCcBqLeko0XFyWWc8cBqLKq09Xg==" />
</div>
<script type="text/C#script">
//<![CDATA[
var theForm = document.forms['ctl00'];
if (!theForm) {
   theForm = document.ctl00;
}
function __doPostBack(eventTarget, eventArgument) {
   if (!theForm.onsubmit || (theForm.onsubmit() != false)) {
      theForm.__EVENTTARGET.value = eventTarget;
      theForm.__EVENTARGUMENT.value = eventArgument;
      theForm.submit();
   }
}
//]]>
</script>
<div>
 <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
value="/wEWBwLj4uulBwKHh6WhBALyvoPgAgKS18aDDQKB0NaIBgLNsJvQDgKFzrr8ARI8/vgGOfyQ/PR+v4JHa7+hvQgr" />
</div>
   <p>
   First name: <input name="FirstName" type="text" id="FirstName" /> <br />
   Last name: <input name="LastName" type="text" id="LastName" /> <br />
   email: <input name="Email" type="text" id="Email" /> <br />
   <input value="Male" name="" type="radio" id="rbMale" /> Male <br />
   <input value="Female" name="" type="radio" id="rbFeMale" />Female <br />
   <input name="btnSend" type="submit" id="btnSend" value="Send" />
   <input name="btnReset" type="reset" id="btnReset" value="Reset" /> <br />
   </p>
 </form>
</body>
</html>
```

Example 6.3-4 – Client side source (emitted via "View Source")

Note : It is worth looking at the examples given on pages 130-134 in your textbook.

➢6.0 Web Form Fundamentals

6.4 HTML Control Classes

Internally, the HTML tags are bound to a C# class. These classes are defined In the **System,Web.UI.HtmlControls** namespace. The Figure 6.4-1 shows all classes defined In the **HtmlControls** namespace.

[Also see text book pages 135-140]

HTML Server Control	Description
HtmlAnchor	Controls an <a> HTML element
HtmlButton	Controls a <button> HTML element
HtmlForm	Controls a <form> HTML element
HtmlGeneric	Controls other HTML element not specified by a specific HTML server control, like <body>, <div>, <span>, etc.
HtmlImage	Controls an <image> HTML element
HtmlInputButton	Controls <input type="button">, <input type="submit">, and <input type="reset"> HTML elements
HtmlInputCheckBox	Controls an <input type="checkbox"> HTML element
HtmlInputFile	Controls an <input type="file"> HTML element
HtmlInputHidden	Controls an <input type="hidden"> HTML element
HtmlInputImage	Controls an <input type="image"> HTML element
HtmlInputRadioButton	Controls an <input type="radio"> HTML element
HtmlInputText	Controls <input type="text"> and <input type="password"> HTML elements
HtmlSelect	Controls a <select> HTML element
HtmlTable	Controls a <table> HTML element
HtmlTableCell	Controls <td>and <th> HTML elements
HtmlTableRow	Controls a <tr> HTML element
HtmlTextArea	Controls a <textarea> HTML element

Figure 6.4-1– Class in HtmlControls namespace

➢6.0 Web Form Fundamentals

6.5 How the files of an ASP.NET application are stored and compiled

- When you build an ASP.NET application, the .NET compiler compiles the files that contain C# code to create an assembly. The assembly is stored in a DLL file on the web server.
- The **aspx** files that contain the HTML code are not compiled along with the C# code. Instead, those files are stored in their original format on the web server.
- When the user requests an ASP.NET page, ASP.NET: creates a class file from the **aspx** file for the page
- Compiles that class file and the code it inherits from the C# assembly into a single assembly that's stored on disk in a DLL file
- Executes the DLL to generate the page

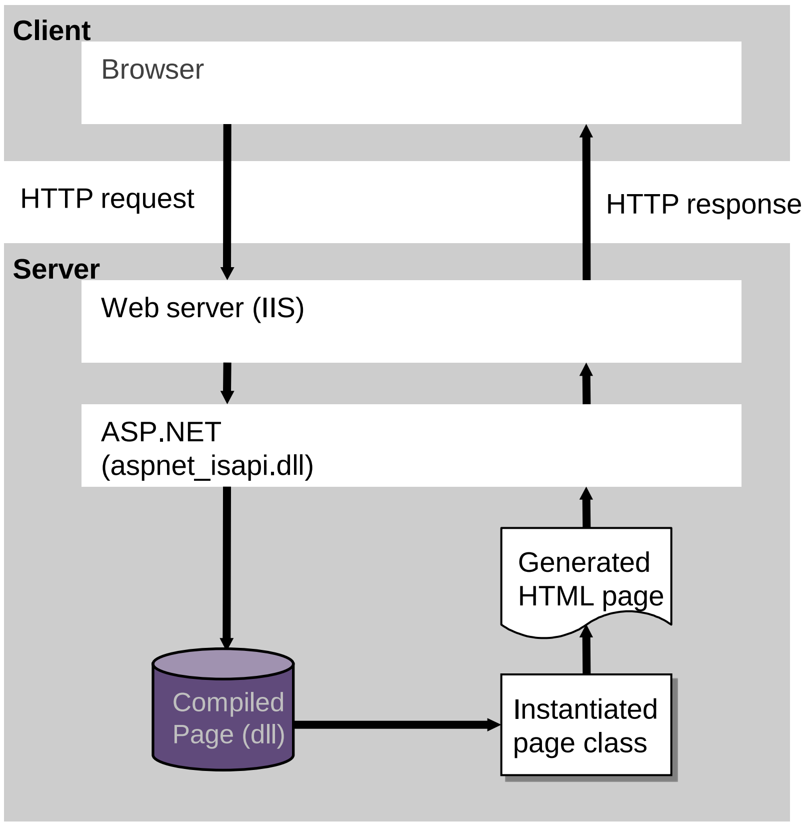


Figure 6.5-1– ASP.NET High Level Architecture

➢6.0 Web Form Fundamentals

6.6 ASP.NET Page class Overview

When an ASP.NET page is requested and renders markup to a browser, the code that runs is not solely the code that you created for your page. Instead, at run time, ASP.NET generates and compiles one or more classes that actually perform the tasks required to run the page. This topic provides an overview of the code that is generated at run time.

6.6.1Generating and Running the Page Class Code

An ASP.NET page runs as a unit, combining the server-side elements in a page, such as controls, with the event-handling code you have written. You do not have to precompile pages into assemblies. ASP.NET dynamically compiles pages and runs them the first time they are requested by a user. If there are any changes to the page or resources the page depends on, the page is automatically recompiled. The class or classes that the compiler creates depends on whether the page uses the single-file model or the code-behind model.

6.6.2 Single-File Pages

In a single-file page, the markup, server-side elements, and event-handling code are all in a single .aspx file. When the page is compiled, the compiler generates and compiles a new class that derives from the base Page class or a custom base class defined with the Inherits attribute of the @ Page directive. For example, if you create a new ASP.NET Web page named SamplePage1 in your application's root directory then a new class named ASP.SamplePage1_aspx is derived from the Page class. For pages inside of application subfolders, the subfolder name is used as part of the generated class.

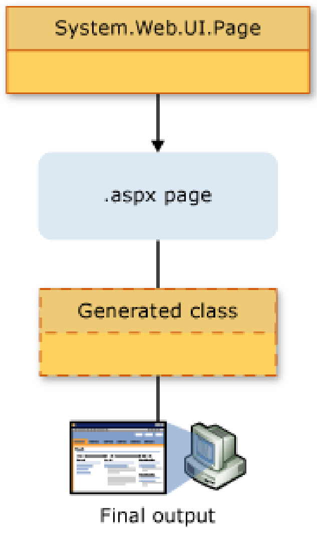


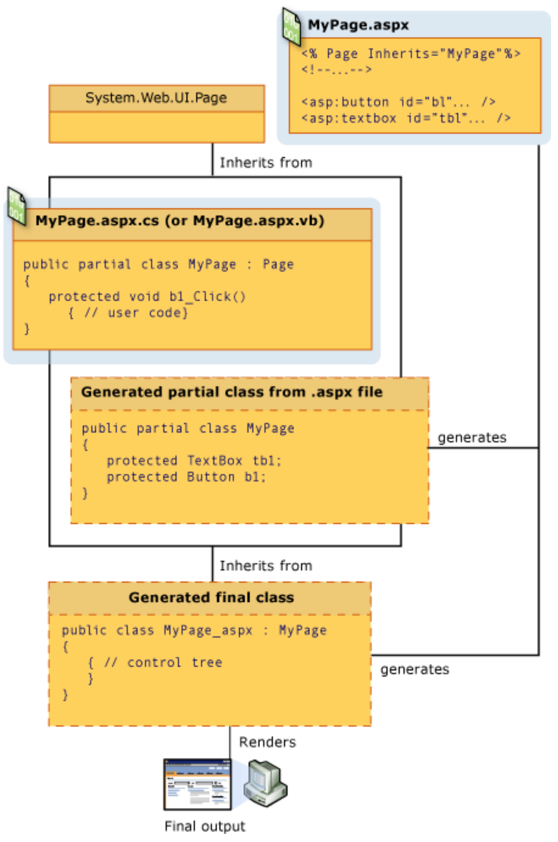
Figure 6.6-1 Inheritance Model

The generated class contains declarations for the controls in the .aspx page and contains your event handlers and other custom code. After page generation, the generated class is compiled into an assembly, the assembly is loaded into the application domain, and then the page class is instantiated and executed to render output to the browser. If you make changes to the page that would affect the generated class—whether by adding controls or modifying your code—the compiled class code is

invalidated and a new class is generated. Figure 6.6 -1 illustrates the inheritance model for the page class in a single-file ASP.NET Web page.

6.6.3 Code Behind Pages

In the code-behind model, the page's markup and server-side elements, Including control declarations, are in an .aspx file, while your page code is in a separate code file. The code file contains a partial class—that is, a class declaration with the keyword partial indicating that it contains only some of the total code that makes up the full class for the page. In the partial class, you add the code that your application requires for the page. This typically consists of event handlers, but can include any methods or properties that you need.



```
<% Page Inherits="MyPage"%>
<!--...-->

<asp:button id="bl"... />
<asp:textbox id="tbl"... />
```

```
public partial class MyPage : Page
{
   protected void b1_Click()
     { // user code}
}
```

```
public partial class MyPage
{
   protected TextBox tbl;
   protected Button b1;
}
```

```
public class MyPage_aspx : MyPage
{
   { // control tree
   }
}
```

Figure 6.6-2

The inheritance model for code-behind pages is slightly more complex than that for single-f file pages. The model is this:

The code-behind file contains a partial class that inherits from a base page class.

The base page class can be the Page class, or it can be another class that derives from Page.

The .aspx file contains an Inherits attribute in the @ Page directive that points to the code-behind partial class.

➢6.0 Web Form Fundamentals

When the page is compiled, ASP.NET generates a partial class based on the .aspx file; this class is a partial class of the code-behind class file. The generated partial class file contains declarations for the page's controls. This partial class enables your code-behind file to be used as part of a complete class without requiring you to declare the controls explicitly.

Finally, ASP.NET generates another class that inherits from the class generated in Step 3. This second generated class contains the code required to build the page. The second generated class and the code-behind class are compiled into an assembly that runs to render output to the browser. Figure 6.7 shows the inheritance model for the page class in A code-behind ASP.NET Web page:.

6.6.4 Page Properties

There are a series of page properties that you can use for various purposes. The listing is too large to explain here. Visit the following web site. All you need is discussed in there. Page Properties

6.6.5 Page Switching (or redirecting)

The traditional HTML style of page switching is to use a anchor <a> tag. The general syntax is :

```
<a href="#identifier"> words go here</a>
```

Example

```
<a href="MyOtherPage.aspx"> Go to the other
page</a>
```

This tag will 'hyperlink' the Go to the other page and when clicks it will load the MyOtherPage.aspx page.

Another way of re-directing to another page is to use the
`Response.Redirect()` method or the **`Server.Transfer()`**
method. `Response.Redirect()` simply sends a message down to the

➢6.0 Web Form Fundamentals

browser, telling it to move to another page. So, you may run code like:

```
Response.Redirect("MyOtherPage.aspx")      or
Server.Transfer(“MyOtherPage.aspx");
        or
Response.Redirect("http://www.microsoft.com/") or
Server.Transfer(“http://www.microsoft.com “);
```

to send the user to another page.

Lets look at an example:

```
!Example 6.6 -1a
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="PageReDirectTest.aspx.cs"
Inherits="PageReDirectTest._Default" %>

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

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Untitled Page</title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
  <center>
  <asp:Label  text="WELCOME THE MAIN PAGE"
runat="server" /> <br />
  <asp:Label  text="Hello there. You are now in Main
Page”
              runat="server" /> <br />
  <asp:Button id="btnSwitchPage" text="Go To
Another Page" OnClick="goToClicked"
runat="server" /> <br />
  </center>
  </div>
  </form>
</body>
</html>
```

Example 6.6-1a – Presentation Code (Main)

```
//This is the Code Behind for PageReDirectTest.aspx
//Example 6.6- 1b
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 PageReDirectTest
{
 public partial class _Default : System.Web.UI.Page
 {
  protected void Page_Load(object sender, EventArgs e)
  {
  }

  protected void goToClicked(object sender, EventArgs e)
  {
   Server.Transfer("Redirect1.aspx");
  }
 }
}
```

Example 6.6-1b– Code Behind

➢6.o Web Form Fundamentals

```
!Example 6.6-2a
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Redirect1.aspx.cs"
Inherits="PageReDirectTest.WebForm1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Untitled Page</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
   <center>
    <asp:Label  text="You are now in switched page"
runat="server" /> <br />
    <asp:Button id="btnGotoMain" text="Go To Main Page"
OnClick="goToMainPage" runat="server" /> </center>
   </div>
   </form>
</body>
</html>
```

Example 6.6-2a – Presentation Code (Redirect)

```
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 PageReDirectTest
{public partial class WebForm1 : System.Web.UI.Page
 {
  protected void goToClicked(object sender, EventArgs e)
  {  }
  protected void goToMainPage(object sender, EventArgs e)
  { Server.Transfer("PageReDirectTest.aspx"); }
 }}
```

Example 6.6-2b– Code Behind

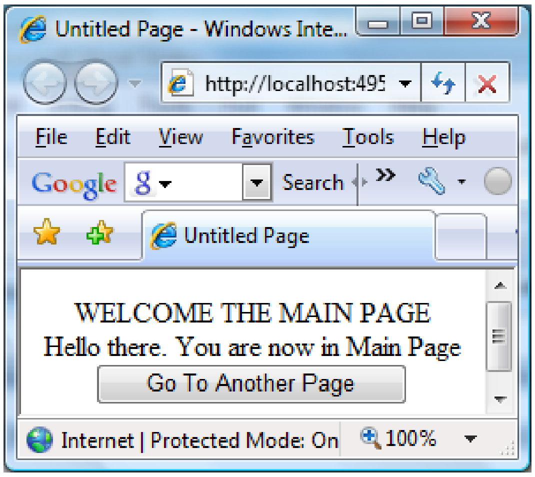

WELCOME THE MAIN PAGE
Hello there. You are now in Main Page
Untitled Page - Windows Inte...
http://localhost:495
Go To Another Page
Internet | Protected Mode: On
100%

Figure 6.6-3a : Start -> Click the "Go To Another Page" Button

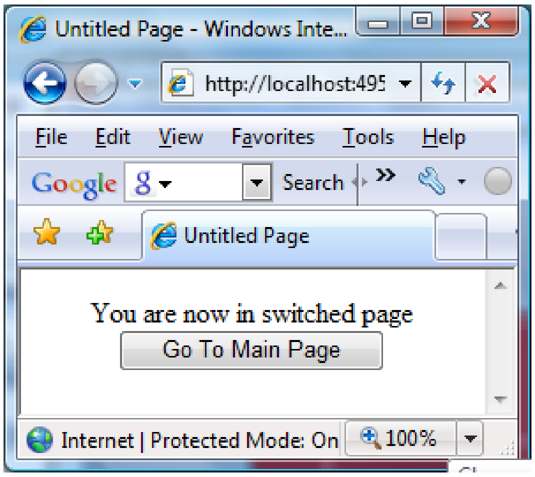
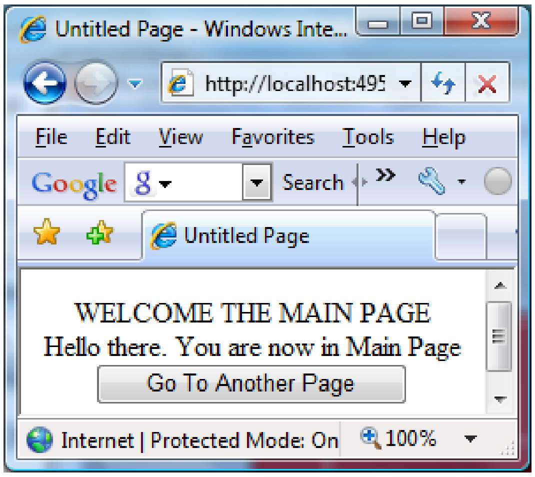

You are now in switched page
Untitled Page - Windows Inte...
http://localhost:495
Go To Main Page
Internet | Protected Mode: On
100%

Figure 6.6-3b Re-Directed page on the browser. Click the "Go To Main Page" Button


WELCOME THE MAIN PAGE
Hello there. You are now in Main Page
Untitled Page - Windows Inte...
http://localhost:495
Go To Another Page
Internet | Protected Mode: On
100%

6.6-3c Main Page appears again.

➤6.0 Web Form Fundamentals

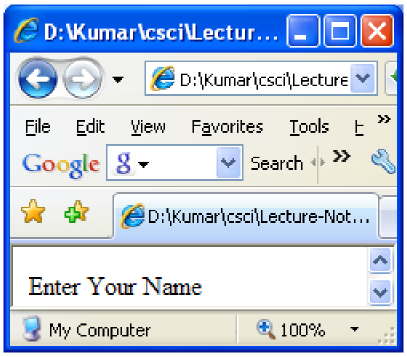
6.7 HTML Encoding

The need for encoding becomes apparent when the look at a simple HTML problem. Say we want to display:

Enter Your Name <Here>

You write a small html snippet like this, expecting to set the result you wanted.

```
<html>

  Enter Your Name <Here>

</html>
```



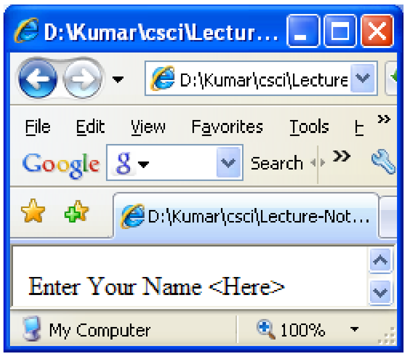
Enter Your Name

Figure 6.7-1

Figure 6.7-1 shows the output of this snippet.

What you need to to display like this. Enter Your Name <Here> .The problem here Is the angle brackets. When you use the angle brackets, HTML thinks that it is a tag. But there is no such <Here> tag in HTML. So that part is considered an error, and nothing is displayed. A solution is to use HTML encoding. Here is the solution.

```
<html>

  Enter Your Name &lt;Here&gt;

</html>
```



Enter Your Name <Here>

Figure 6.7-2

Note : lt = less than   gt = greater than

The expected output is shown in figure 6.7-2

➢6.0 Web Form Fundamentals

The HTML Encoder to display **<** is **&lt** and to display **>** is **&gt**. Similarly there is a set of HTML Encoders you could use for various purposes. Here is a commonly used HTML Encoder list.

Result	Description	Encorder Entity
	Nonbreaking Space	&nbsp;
<	Less-than Symbol	&lt;
>	Greater-than Symbol	&gt;
&	Ampersand	&amp;
"	Double Quote	&quot;

Figure 6.7-3 Common HTML Encoders

A full list of HTML Encorders can be viewed here (http://www.zytrax.com/tech/web/entities.html).

6.7.1 Server.HTMLEncode Method

The HTMLEncode method applies HTML encoding to a specified string. This is useful as a quick method of encoding form data and other client request data before using it in your Web application. Encoding data converts potentially unsafe characters to their HTML-encoded equivalent.

HTMLEncode converts characters as follows:

```
The less-than character (<) is converted to &lt;.
The greater-than character (>) is converted to &gt;.
The ampersand character (&) is converted to &amp;.
The double-quote character (") is converted to &quot;.
```

The following script:

```
Copy <%= Server.HTMLEncode("The paragraph tag: <P>") %>
```

Produces the following HTML output :

```
copy The paragraph tag: &lt;P&gt;
```

The preceding output will be displayed by a Web browser as:

```
Copy The paragraph tag: <P>
```

Via “View Source” you will be able to see the encoded HTML.

➢6.0 Web Form Fundamentals

6.8 Global .asax file

The **Global.asax** file, also known as the ASP.NET application file, is an optional file that contains code for responding to application-level events raised by ASP.NET or by HttpModules. The **Global.asax** file resides in the root directory of an ASP.NET-based application. At run time, Global.asax is parsed and compiled into a dynamically generated .NET Framework class derived from the HttpApplication base class. The **Global.asax** file itself is configured so that any direct URL request for it is automatically rejected; external users cannot download or view the code written within it.

You can use the **Global.asax** file to handle any event exposed by the modules in the request. For example, you might create a custom authentication module for your ASP.NET Web application in which you might expose an OnAuthenticateRequest or **OnEndRequest** event.

The **Global.asax** file looks similar to a normal **.aspx** file. However it cannot contain and HTML or ASP.NET tags. Instead it should contain event handlers.

See the example below.

```
<Script language="C#" runat="server">
 void Application_OnEndRequest(Object Source, EventArgs Details)
 {
  Response.Write("<ht /> This page was served at " + DateTime.Now.ToString());
 }
</script>
```

This will use the **Write()** method of the built in **Response** object to write a footer at the bottom of the page.

6.8 Web.config file

Web.config file, as it sounds like is a configuration file for the ASP.NET web application. An ASP.NET application has one **web.config** file which keeps The configurations required for the corresponding application. **web.config** file is written in XML with specific tags having specific meanings.

There are number of important settings that can be stored in the configuration file. Here are some of the most frequently used configurations, stored conveniently inside **web.config** file..

- Database connections
- Session States
- Error Handling
- Security

Here is an example of a connection string which is stored in the **web.config** file.

```
<configuration>
 <appSettings>
   <add key="ConnectionString"
     value="server=localhost;uid=sa;pwd=;database=DBPerson" />
 </appSettings>
</configuration>
```

Now then a ASP.NET web application can access the connection string like this:

```
using System.Configuration;

string connectionString = (string )
ConfigurationSettings.AppSettings["ConnectionString"];
```