DOT NET

Server Controls

- Created and configured as *objects*.
- Run on the web server
- Automatically provide their own HTML output
- State is maintained and events are triggered

HTML server controls:

- Server-based equivalents for standard HTML elements.
- Useful when migrating ordinary HTML pages or classic ASP pages to ASP.NET, because they require the fewest changes.

Web controls:

- Similar to the HTML server controls, but they provide
  - richer object model
  - variety of properties for style and formatting details.
  - More events
- Web controls also feature some user interface elements that have no direct HTML equivalent, such as the GridView, Calendar, and validation controls.

HTML Server Controls

HTML server controls provide an object interface for standard HTML elements. They provide three key features:

- ***They generate their own interface***: You set properties in code, and the underlying HTML tag is created automatically when the page is rendered and sent to the client.

- ***They retain their state***: Because the Web is stateless, ordinary web pages need to do a lot of work to store information between requests. HTML server controls handle this task automatically. For example, if the user selects an item in a list box, that item remains selected the next time the page is generated. Or if your code changes the text in a button, the new text sticks the next time the page is posted back to the web server.

- ***They fire server-side events***: For example, buttons fire an event when clicked, text boxes fire an event when the text they contain is modified, and so on. Your code can respond to these events, just like ordinary controls in a Windows application. If a given event doesn't occur, the event handler won't be executed.

Control - Most Important Properties

- HtmlAnchor - HRef, Target
- HtmlImage - Src, Alt, Width, Height
- HtmlInputCheckBox - Checked
- HtmlInputRadioButton - Checked
- HtmlInputText - Value
- HtmlSelect - Items (collection)
- HtmlGenericControl - InnerText InnerHtml

Sample Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
public partial class CurrencyConverter : System.Web.UI.Page
{
            protected void Convert_ServerClick(object sender, EventArgs e)
            {
            decimal USAmount = Decimal.Parse(US.Value);
            decimal euroAmount = USAmount * 0.85 M;
            Result.InnerText = USAmount.ToString() + " U.S. dollars = ";
            Result.InnerText + = euroAmount.ToString() + " Euros.";
            }
}
```

- It starts with several *using* statements. This provides access to all the important namespaces. This is a typical first step in any code-behind file.

- The page class is defined with the partial keyword. That’s because your class code is merged with another code file that you never see. This extra code, which ASP.NET generates automatically, defines all the server controls that are used on the page. This allows you to access them by name in your code.

- The page defines a single event handler. You’ll notice that the event handler accepts two parameters (sender and e). It allows your code to identify the control that sent the event (through the sender parameter) and retrieve any other information that may be associated with the event (through the e parameter).

- The event handler is connected to the control event by using the OnServerClick attribute in the <input> tag for the button.

Behind the Scenes

1. The request for the page is sent to the web server. The request is sent to the built-in test server.

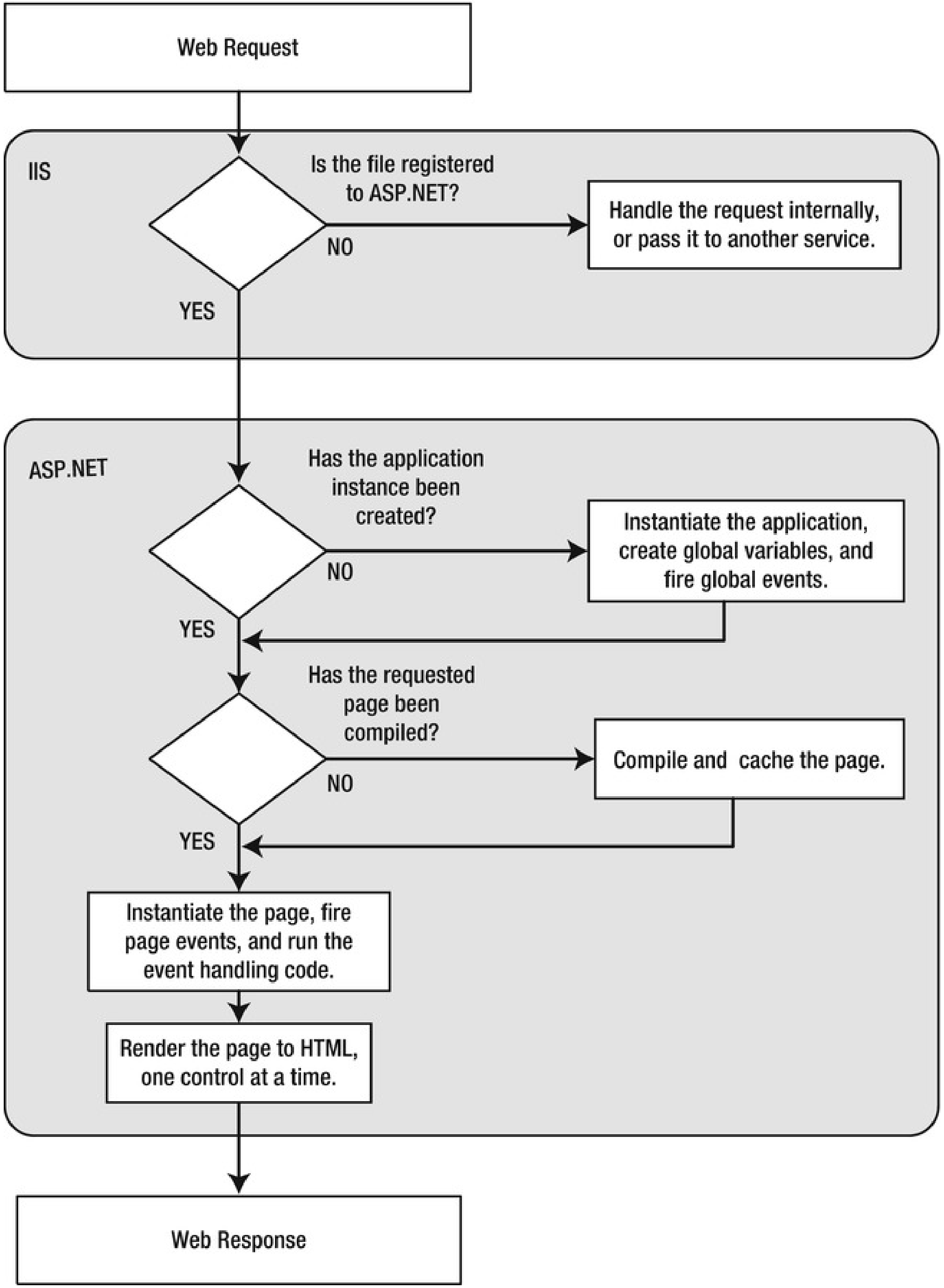
2. The web server determines that the .aspx file extension is registered with ASP.NET. If the file extension belonged to another service (as it would for .html files, for example),ASP.NET would never get involved.

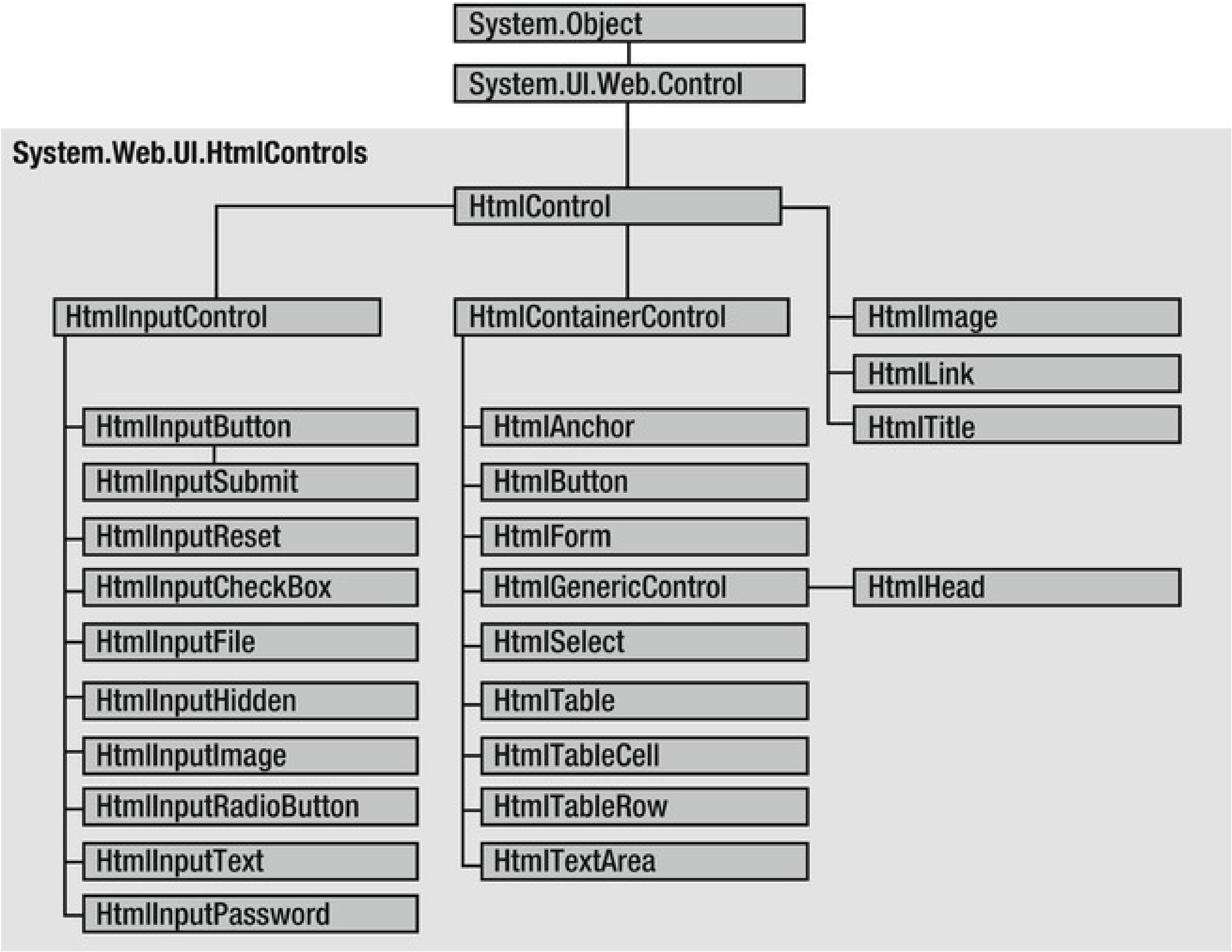
3. If this is the first time a page in this application has been requested, ASP.NET automatically creates the application domain. It also compiles all the web page code for optimum performance, and caches the compiled files. If this task has already been performed, ASP.NET will reuse the compiled version of the page.

4. The compiled CurrencyConverter.aspx page acts like a miniature program. It starts firing events (most notably, the Page.Load event). However, you haven't created an event handler for that event, so no code runs. 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 into the corresponding HTML markup.

6. The final page is sent to the user, and the application ends.

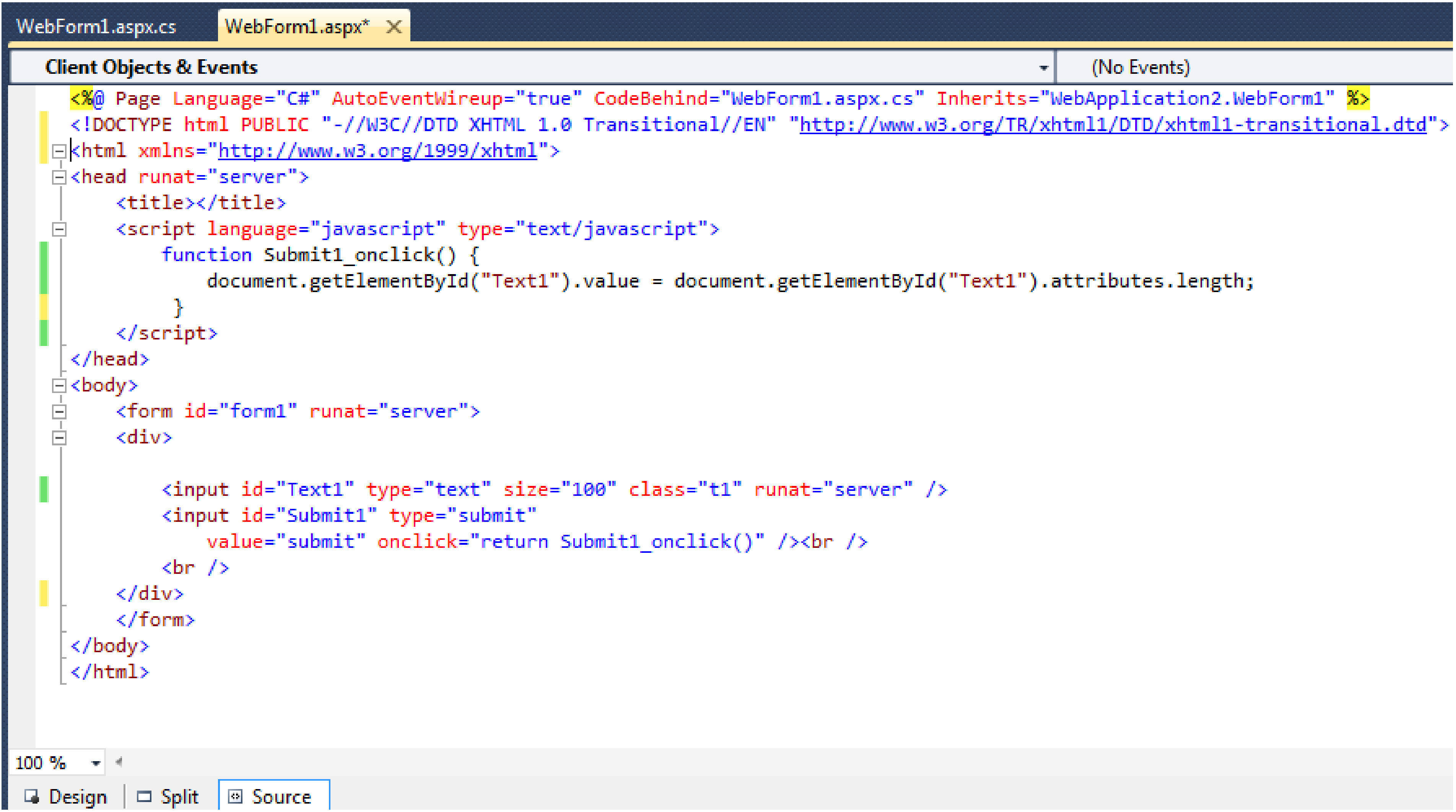

Web Request
IIS
Is the file registered to ASP.NET?
NO
Handle the request internally, or pass it to another service.
YES
ASP.NET
Has the application instance been created?
NO
Instantiate the application, create global variables, and fire global events.
YES
Has the requested page been compiled?
NO
Compile and cache the page.
YES
Instantiate the page, fire page events, and run the event handling code.
Render the page to HTML, one control at a time.
Web Response


System.Object
System.UI.Web.Control
System.Web.UI.HtmlControls
HtmlControl
HtmlInputControl
HtmlContainerControl
HtmlImage
HtmlLink
HtmlTitle
HtmlInputButton
HtmlInputSubmit
HtmlInputReset
HtmlInputCheckBox
HtmlInputFile
HtmlInputHidden
HtmlInputImage
HtmlInputRadioButton
HtmlInputText
HtmlInputPassword
HtmlAnchor
HtmlButton
HtmlForm
HtmlGenericControl
HtmlHead
HtmlSelect
HtmlTable
HtmlTableCell
HtmlTableRow
HtmlTextArea

HTML Control Events

- **Event Controls That Provide It**
- ServerClick - HtmlAnchor, HtmlButton, HtmlInputButton, HtmlInputImage, HtmlInputReset
- ServerChange - HtmlInputText, HtmlInputCheckBox, HtmlInputRadioButton, HtmlSelect

- **HtmlControl Properties**
- **Attributes** - Provides a collection of all the attributes that are set in the control tag, and their values. Rather than reading or setting an attribute through the Attributes, it's better to use the corresponding property in the control class.
- **Controls** - Provides a collection of all the controls contained inside the current control. (Forexample, a <div> server control could contain an <input> server control.) Each object is provided as a generic System.Web.UI.Control object so that you may need to cast thereference to access control-specific properties.
- **Disabled** - Disables the control when set to true, thereby ensuring that the user cannot interact with it, and its events will not be fired.
- **EnableViewState** - Disables the automatic state management for this control when set to false. 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 default), the control stores its state in a hidden input field on the page, thereby 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 (rather than inside another control), 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 underlying HTML element (for example, img or div).
- **Visible** - Hides the control when set to false and will not be rendered to the final HTML page that is sent to the client.



```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="WebApplication2.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></title>
    <script language="javascript" type="text/javascript">
        function Submit1_onclick() {
            document.getElementById("Text1").value = document.getElementById("Text1").attributes.length;
         }
    </script>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <input id="Text1" type="text" size="100" class="t1" runat="server" />
        <input id="Submit1" type="submit"
            value="submit" onclick="return Submit1_onclick()" /><br />
        <br />
    </div>
    </form>
</body>
</html>
```

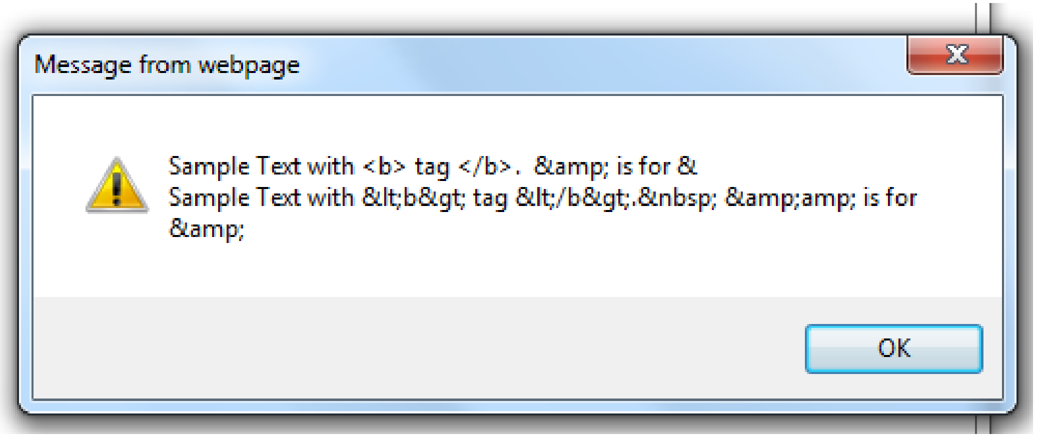
HtmlInputControl Properties

- **Type** - Provides the type of input control.
- **Value** - Returns the contents of the control as a string.

HtmlContainerControl Properties

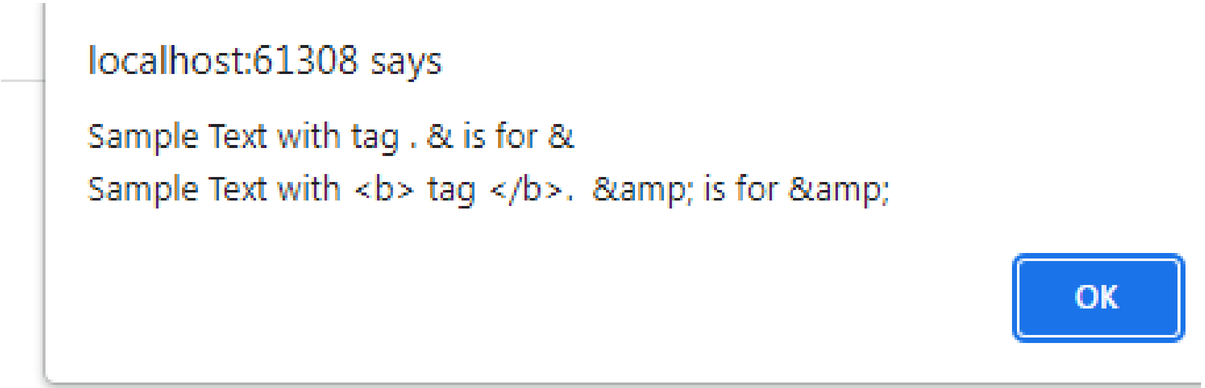
- **InnerText** - The text content between the opening and closing tags of the control. Special characters will be automatically converted to HTML entities and displayed as text (for example, the less-than character (<) will be converted to &lt; and will be displayed as <in the web page).
- **InnerHtml** - 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.

```
document.getElementById("div2").innerText = "Sample Text with <b> tag </b>.  &amp; is for &";
minnerText = document.getElementById("div2").innerText;
minnerHtml = document.getElementById("div2").innerHTML;
alert(minnerText + "\n" + minnerHtml);
```



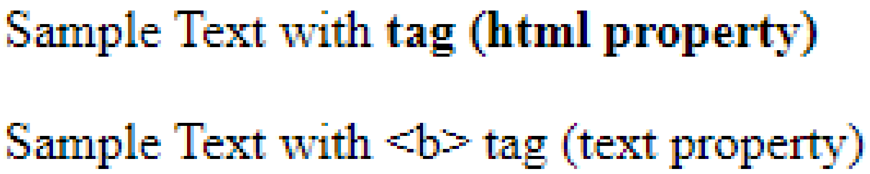
```
Sample Text with <b> tag </b>.  &amp; is for &
Sample Text with &lt;b&gt; tag &lt;/b&gt;.&nbsp; &amp;amp; is for
&amp;
```

```
document.getElementById("div2").innerHTML = "Sample Text with <b> tag </b>.  &amp; is for &";
```



```
Sample Text with tag . & is for &
Sample Text with <b> tag </b>.  &amp; is for &amp;
```

```
$("#p1").html("Sample Text with <b> tag (html property");
$("#p2").text("Sample Text with <b> tag (text property");
```



Sample Text with **tag (html property)**

Sample Text with <b> tag (text property)

Page Properties

Property	Description
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 view state information (true).
EnableViewState	When set to false, this overrides the EnableViewState property of the contained controls, thereby ensuring that no controls will maintain state information.
Application	This collection holds information that’s shared between all users in your website
Session	This collection holds information for a single user, so it can be used in different pages.
Cache	This collection allows you to store objects that are time-consuming to create so they can be reused in other pages or for other clients.
Request	This refers to an HttpRequest object that contains information about the current web request.
Response	This refers to an HttpResponse object that represents the response ASP.NET will send to the user’s browser.
Server	This refers to an HttpServerUtility object that allows you to perform a few miscellaneous tasks. For example, it allows you to encode text so that it’s safe to place it in a URL or in the HTML markup of your page.
User	If the user has been authenticated, this property will be initialized with user information.

EnableViewState and ViewStateMode

- The ViewStateMode property of a page or a control has an effect only if the EnableViewState property is set to true.

- The default value of the ViewStateMode property for a page is Enabled. The default value of the ViewStateMode property for a Web server control in a page is Inherit.

- You can use the ViewStateMode property to enable view state for an individual control even if view state is disabled for the page.

- Disabled - will disable the viewstate for that page or control(i.e. if the page level property is disabled and control level property is enabled, view state will work for the control).

- Enabled - will enable the viewstate for that page or control(i.e. if the page level property is enabled and control level property is disabled, view state will not work for the control).

- Inherit - will inherit the page viewstate property and apply it to the control viewstate property.

- When EnableViewState of Page is set to True

EnableViewState-ViewStateMode	Status
True – Inherit	Content Preserved
True – Enabled	Content preserved
True - Disabled	Content is not preserved
False – Inherit	Content is not preserved
False – Enabled	Content is not preserved
False – Disabled	Content is not preserved

- When EnableViewState of Page is set to False, Content Preserved irrespective of control's EnableViewState / ViewStateMode

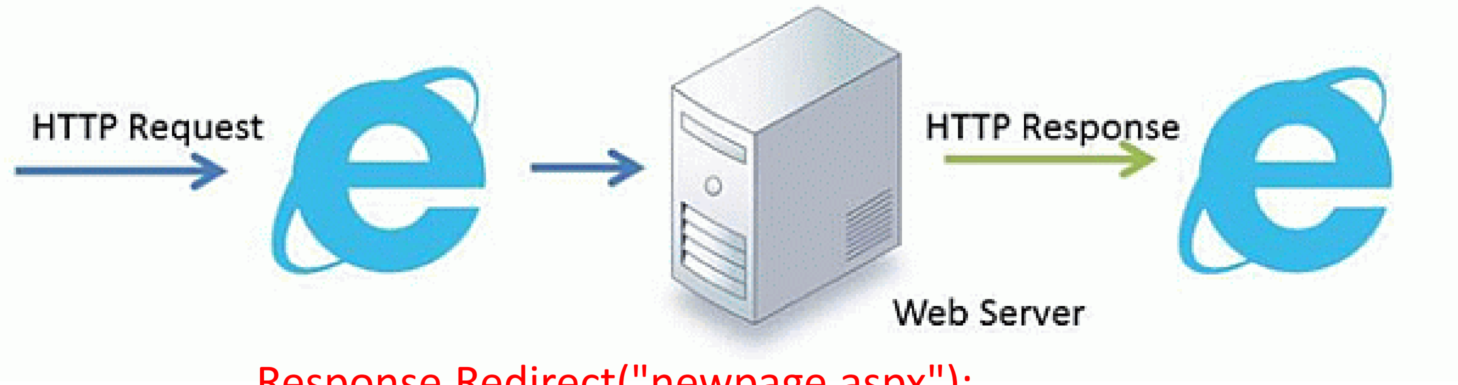
Sending the User to a New Page

- Click <a href = "newpage.aspx" > here</a > to go to newpage.aspx

- Response.Redirect("newpage.aspx");

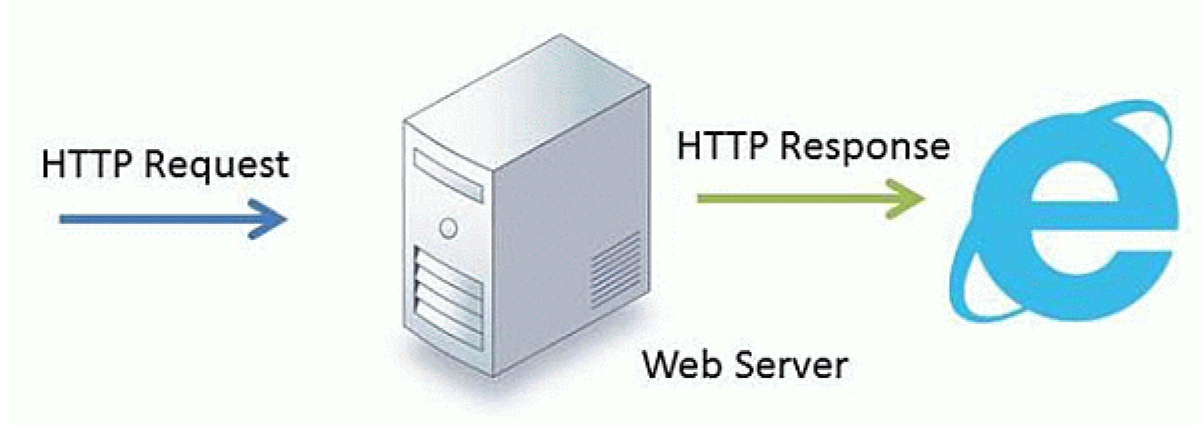
- Server.Transfer("newpage.aspx");

Response.Redirect()	Server.Transfer()
ASP.NET immediately stops processing the page and sends a redirect message back to the browser. When the browser receives the redirect message, it sends a request for the new page.	Doesn’t involve the browser. Instead of sending a redirect message back to the browser, ASP.NET simply starts processing the new page as though the user had originally requested that page.
browser shows the new URL	browser will still show the original URL.
used to redirect a user to an external websites	used only on sites running on the same server
the previous page is removed from server memory and loads a new page in memory	previous page also exists in server memory



Response.Redirect("newpage.aspx");

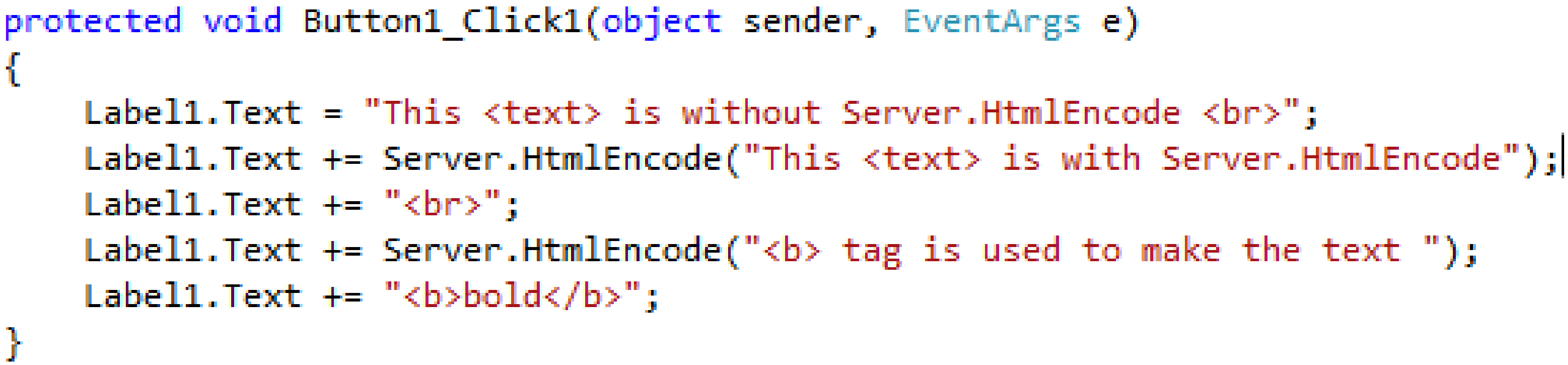
Response.Redirect sends an HTTP request to the browser, then the browser sends that request to the web server, then the web server delivers a response to the web browser.



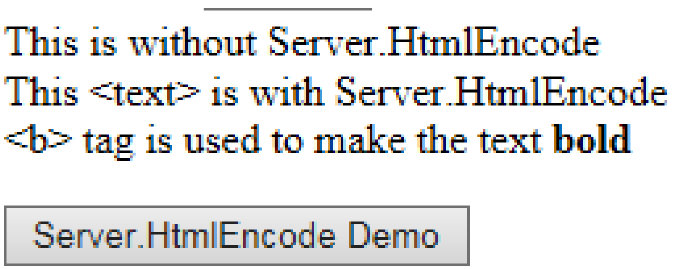
Server.Transfer("newpage.aspx");

Server.Transfer sends a request directly to the web server and the web server delivers the response to the browser.

Working with HTMLEncoding

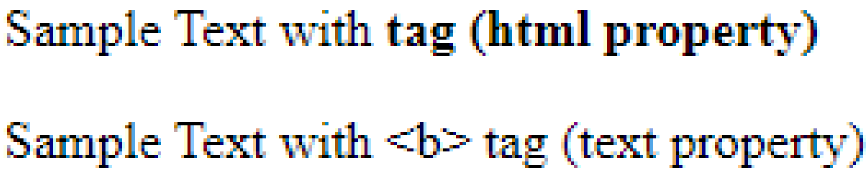


```
protected void Button1_Click1(object sender, EventArgs e)
{
    Label1.Text = "This <text> is without Server.HtmlEncode <br>";
    Label1.Text += Server.HtmlEncode("This <text> is with Server.HtmlEncode");
    Label1.Text += "<br>";
    Label1.Text += Server.HtmlEncode("<b> tag is used to make the text ");
    Label1.Text += "<b>bold</b>";
}
```



This is without Server.HtmlEncode
This <text> is with Server.HtmlEncode
<b> tag is used to make the text **bold**

```
$("#p1").html("Sample Text with <b> tag (html property");
$("#p2").text("Sample Text with <b> tag (text property");
```



Sample Text with **tag (html property)**

Sample Text with <b> tag (text property)

Context Object

- HttpContext object will hold information about the current http request.
- HttpContext object will be constructed newly for every request given to an ASP.Net application
- This holds current request specific information like Request, Response, Server, Session, Cache, User and etc.

```
using System;
public partial class UserRegister : System.Web.UI.Page
{
   protected void btn_Detail_Click(object sender, EventArgs e)
    {
        Context.Items.Add(“Name", “Dev");
        Context.Items.Add("Email", “dev.mehta@gmail.com");
       Server.Transfer("UserDetail.aspx");
    }
}
```

```
using System;
public partial class UserDetail : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
    {   try
        {
           Response.Write( Context.Items["Name"].ToString(),  Context.Items["Email"].ToString()));
        }
       catch (NullReferenceException ex)
        {
           Response.Write(ex.Message);
        }
```

Using Application Events

- By using application events, code can be written that runs every time a request is received, no matter what page is being requested.
- Basic ASP.NET features such as session state and authentication use application events to plug into the ASP.NET processing pipeline.
- code-behind for a web form is not used for this. Instead, we need the help of another ingredient: the global.asax file.

global.asax File

- The global.asax file allows you to write code that responds to global application events. These events fire at various points during the lifetime of a web application, including when the application domain is first created

Table 5-11. Basic Application Events

Event-Handling Method	Description
Application_Start()	Occurs when the application starts, which is the first time it receives a request from any user. It doesn't occur on subsequent requests. This event is commonly used to create or cache some initial information that will be reused later.
Application_End()	Occurs when the application is shutting down, generally because the web server is being restarted. You can insert cleanup code here.
Application_BeginRequest()	Occurs with each request the application receives, just before the page code is executed.
Application_EndRequest()	Occurs with each request the application receives, just after the page code is executed.
Session_Start()	Occurs whenever a new user request is received and a session is started. Sessions are discussed in detail in Chapter 8.
Session_End()	Occurs when a session times out or is programmatically ended. This event is raised only if you are using in-process session-state storage (the InProc mode, not the StateServer or SQLServer modes).
Application_Error()	Occurs in response to an unhandled error. You can find more information about error handling in Chapter 7.

```
protected void Application_Start(object sender, EventArgs e)
        {
            Application["appCtr"] = 0;
            Application["noOfUsers"] = 0;
        }
        protected void Session_Start(object sender, EventArgs e)
        {
            Application.Lock();
            Application["noOfUsers"] = (int)Application["noOfUsers"] + 1;
            Application.UnLock();
        }
        protected void Session_End(object sender, EventArgs e)
        {

            Application.Lock();

            Application["noOfUsers"] = (int)Application["noOfUsers"] - 1;
            Application.UnLock();
        }
        protected void Application_BeginRequest(object sender, EventArgs e)
        {
            Response.Write(" Application BeginRequest method is triggered<hr/><br>");
            Application.Lock();
            Application["appCtr"] = (int)Application["appCtr"] + 1;
            Application.UnLock();
        }
        protected void Application_EndRequest(object sender, EventArgs e)
        {
            Response.Write(" <hr/>This page was served at " + DateTime.Now.ToString() + "<br>");
            Response.Write("No of Visitors - " + Application["appCtr"] + "<br>");
            Response.Write("No of Users Currently Active - " + Application["noOfUsers"]);
        }
```

Point to note

- *The Session_End event doesn't fire when the browser is closed, it fires when the server hasn't gotten a request from the user in a specific time persion (by default 20 minutes). That means that if you use Session_End to remove users, they will stay in the chat for 20 minutes after they have closed the browser.*

```
<sessionState timeout="1" mode="InProc"/>
```

State Management

Chapter – 8

Traditional desktop application	web application
Single user	***Thousands of users can simultaneously run the same application on the same computer (the web server), each one communicating over a stateless HTTP connection.***
users interact with a continuously running application	***Illusion of continuously running application. clients need to be connected for only a few seconds at most, a web server can handle a huge number of nearly simultaneous requests without a performance hit.***
A portion of memory on the desktop computer is allocated to store the current set of working information.	***In a typical web request, the client connects to the web server and requests a page. When the page is delivered, the connection is severed, and the web server discards all the page objects from memory. By the time the user receives a page, the web page code has already stopped running, and there's no information left in the web server's memory.***

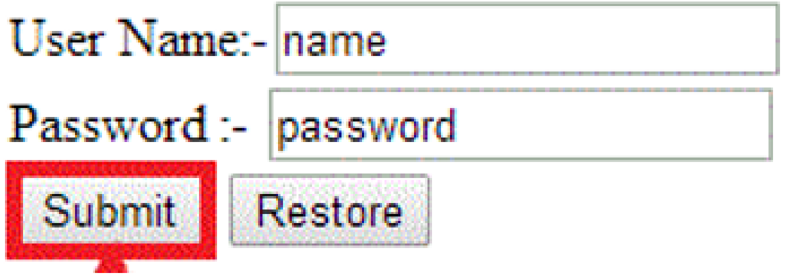
However, if you want to retain information for a longer period of time so it can be used over multiple postbacks or on multiple pages, you need to take additional steps.

Using View State

- One of the most common ways to store information is in view state.
- View State is the method to preserve the Value of the Page and Controls between round trips. It is a Page-Level State Management technique.
- View state uses a hidden field that ASP.NET automatically inserts in the final, rendered HTML of a web page.
- It’s a perfect place to store information that’s used for multiple postbacks in a single web page.

Code without ViewState

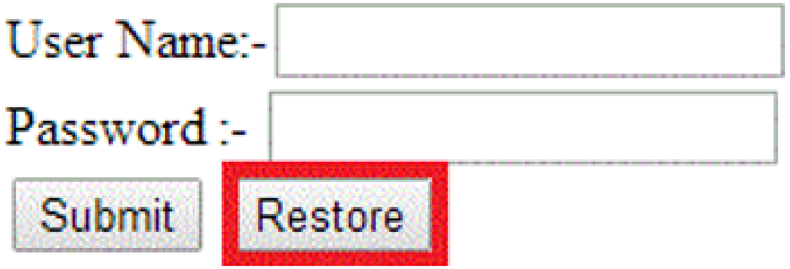
```
public string a, b;
protected void Button1_Click(object sender, EventArgs e)
{
    a = TextBox1.Text;
    b = TextBox2.Text;
    TextBox1.Text = TextBox2.Text = " ";
}
```



User Name:- name

Password :- password

```
protected void Button2_Click(object sender, EventArgs e)
{
    TextBox1.Text = a;
    TextBox2.Text = b;
}
```

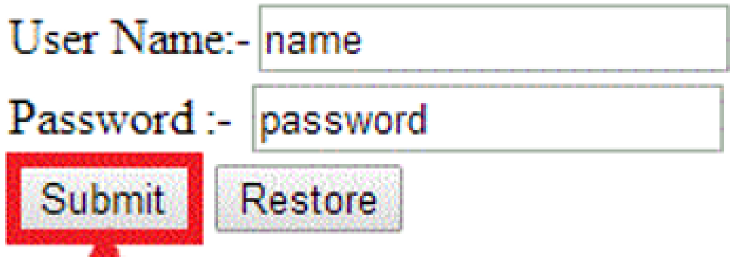


User Name:-

Password :-

Code with ViewState

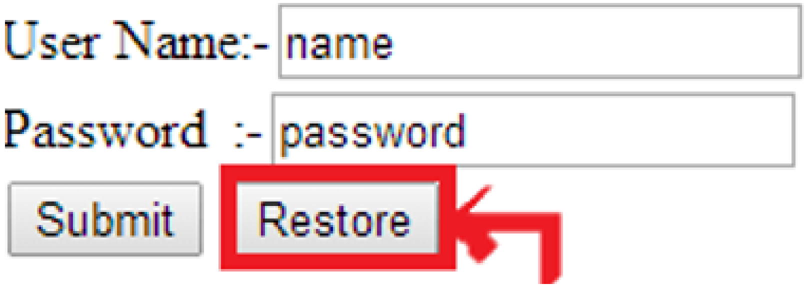
```
protected void Button1_Click(object sender, EventArgs e)
{
    ViewState["name"] = TextBox1.Text;
    ViewState["password"] = TextBox2.Text;
    TextBox1.Text = TextBox2.Text = ””;
}
```



User Name:- name

Password :- password

```
----------------------------------------------------------------------------------------------------
protected void Button2_Click(object sender, EventArgs e)
{
    if (ViewState["name"] != null)
    {
        TextBox1.Text = ViewState["name"].ToString();
    }
    if (ViewState["password"] != null)
    {
         TextBox2.Text = ViewState["password"].ToString();
    }
}
```



User Name:- name

Password :- password

Limitations

- One of the most significant limitations with view state is that it’s tightly bound to a specific page.
- If the user navigates to another page, this information is lost

Cross-Page Posting

- A cross-page postback is a technique that extends the postback mechanism you’ve already learned about so that one page can send the user to another page, complete with all the information for that page.
- To use cross-posting, you simply set PostBackUrl to the name of another web form. When the user clicks the button, the page will be posted to that new URL with the values from all the input controls on the current page.
- Interaction between pages take place using Page.PreviousPage property

Code to Demonstrate Cross Page Posting

```
protected void page_load( object sender , eventArgs e)
{
   if (PreviousPage !=null)
         {
            Button btn = (Button)(PreviousPage.FindControl("Button1"));
             TextBox txt = (TextBox)(PreviousPage.FindControl("TextBox1"));
             TextBox1.Font.Size = 12;
             TextBox1.Font.Italic = true;
             TextBox1.ForeColor = System.Drawing.Color.Blue;
             TextBox1.Text="Previous Page is " + PreviousPage.Title + " \nYou 
clicked "+ btn.Text + "\nWelcome  " + txt.Text;
         }
}
```

Query String

- This approach is commonly found in search engines. Here’s an example:

http://www.google.com/search?q=dotnet+framework

- Well suited in database applications in which you present the user with a list of items that correspond to records in a database, such as products. The user can then select an item and be forwarded to another page with detailed information about the selected item.

- Information is limited to simple strings, which must contain URL-legal characters.
- Information is clearly visible to the user and to anyone else who cares to eavesdrop on the Internet.
- The enterprising user might decide to modify the query string and supply new values, which your program won’t expect and can’t protect against.
- Many browsers impose a limit on the length of a URL (usually from 1 KB to 2 KB).

Demo Code for Query String :

- Default1.aspx :

```
Response.Redirect("newpage.aspx?recordID=10&mode=full");
```

(or)

```
string url = “newpage.aspx?";
url += "Item=" + lstItems.SelectedItem.Text + "&";
url += "Mode=" + chkDetails.Checked.ToString();
```

- newpage.aspx :

```
string ID = Request.QueryString["recordID"];
```

URL Encoding

- One potential problem with the query string is that some characters aren’t allowed in a URL. All characters must be alphanumeric or one of a small set of special characters (including $-_.+!*'(),).
- Furthermore, some characters have special meaning. For example, the ampersand (&) is used to separate multiple query string parameters, the plus sign (+) is an alternate way to represent a space, and the number sign (#) is used to point to a specific bookmark in a web page. If you try to send query string values that include any of these characters, you’ll lose some of your data.

Demo Code for Query String

Default.aspx : (list box, checkbox and button)

Button Click Event :

```
 protected void cmdGo_Click(Object sender,
EventArgs e)
 {
if (lstItems.SelectedIndex == -1)
          {          lblError.Text = "You must select
an item.";       }
else
{
string url = "QueryStringRecipient.aspx?";
url += "Item=" + lstItems.SelectedItem.Text + "&";
url += "Mode=" + chkDetails.Checked.ToString();
Response.Redirect(url);
 }
 }
```

Newpage.aspx :

```
protected void Page_Load(Object sender, EventArgs e)
{
lblInfo.Text = "Item: " + Request.QueryString["Item"];
lblInfo.Text += "<br />Show Full Record: ";
lblInfo.Text += Request.QueryString["Mode"];
}
```

```
string url = “newpage.aspx?";
url += "Item=" + Server.UrlEncode(lstItems.SelectedItem.Text)
+ "&";
url += "Mode=" + chkDetails.Checked.ToString();
Response.Redirect(url);
```

Cookies

- Cookies are small files that are created in the web browser’s memory (if they’re temporary) or on the client’s hard drive (if they’re permanent).
- Can be easily used by any page in the application and even be retained between visits, which allows for truly long-term storage.
- Same drawbacks that affect query strings—namely,
- Limited to simple string information
- Easily accessible and readable if the user finds and opens the corresponding file.
- Poor choice for complex or private information or large amounts of data.
- Some users disable cookies on their browsers, which will cause problems for web applications that require them.
- Also, users might manually delete the cookie files stored on their hard drives.
- But for the most part, cookies are widely adopted and used extensively on many websites.
- using System.Net;

Demo Code for Creating Cookies

```
HttpCookie userInfo = new HttpCookie("userInfo");
userInfo["UserName"] = “TestUser";
userInfo["UserCity"] = “Mumbai";
userInfo.Expires =  DateTime.Now.AddMinutes(10);
Response.Cookies.Add(userInfo);
```

Demo Code for Retriving Cookies

```
string User_Name = string.Empty;
string User_City = string.Empty;
HttpCookie reqCookies = Request.Cookies["userInfo"];
if (reqCookies != null)
{
    User_Name = reqCookies["UserName"].ToString();
    User_City = reqCookies["UserCity"].ToString();
}
```

Session State Variables

To create session state variables :

```
Session["username"] = TextBox1.Text;
Session["Email"] = TextBox2.Text;
 Session["Subject"] = ListBox1.SelectedValue;
```

To retrieve the session variables:

```
String uname=Session[“username”].ToString();
```

Validation Controls

Chapter – 9

Types of User Errors during Data Entry
Leaving a field blank
Entering invalid data by mistake
Entering invalid data on purpose to break the code

Validation Controls

Control Class	Description
RequiredFieldValidator	Validation succeeds if input control doesn’t contain an empty string.
RangeValidator	Validation succeeds if the input control contains a value within a specific numeric, alphabetic, or date range.
CompareValidator	Validation succeeds if the input control contains a value that matches the value in another input control, or a fixed value that is specified.
RegularExpressionValidator	Validation succeeds if the value in an input control matches a specified regular expression.
CustomValidator	Validation is performed by a user-defined function.

Server-Side Validation

- Validator controls can be used to verify a page automatically when the user submits it or manually in your code.
- When using automatic validation, the user receives a normal page and begins to fill in the input controls.
- When finished, the user clicks a button to submit the page. Every button has a CausesValidation property, which can be set to true or false

Client-Side Validation

- In modern browsers, ASP.NET automatically adds JavaScript code for client-side validation. In this case, when the user clicks a CausesValidation button, the same error messages will appear without the page needing to be submitted and returned from the server. This increases the responsiveness of your web page.
- However, even if the page validates successfully on the client side, ASP.NET still revalidates it when it’s received at the server.

BaseValidator Class

Property	Description
ControlToValidate	Identifies the control that this validator will check.
ErrorMessage and ForeColor	If validation fails, the validator control can display a text message (set by the ErrorMessage property). By changing the ForeColor, you can make this message stand out in angry red lettering.
Display	Static / Dynamic
IsValid , Enabled, EnableClientScript	True / False

Validator-Specific Properties

Property	Description
RangeValidator	MaximumValue, MinimumValue, Type
CompareValidator	ControlToCompare, Operator, Type, ValueToCompare
RegularExpressionValidator	ValidationExpression
CustomValidator	ClientValidationFunction, ValidateEmptyText, ServerValidate event

Manual Validation

Disable validation and perform the work on your own

You can create manual validation in one of three ways:

- Use your own code to verify values. In this case, you won’t use any of the ASP.NET validation controls.
- Disable the EnableClientScript property for each validation control. This allows an invalid page to be submitted, after which you can decide what to do with it depending on the problems that may exist.
- Add a button with CausesValidation set to false. When this button is clicked, manually validate the page by calling the Page.Validate() method. Then examine the IsValid property and decide what to do.

Validation with Regular Expression

Character	Description
*	Zero or more occurrences of the previous character or subexpression. For example, 7*8 matches 7778 or just 8.
+	One or more occurrences of the previous character or subexpression. For example, 7+8 matches 7778 but not 8.
{ }	Groups a subexpression that will be treated as a single element. For example, (78)+ matches 78 and 787878.
{m,n}	Previous character (or subexpression) can occur from *m* to *n* times. A{1,3}
|	Either of two matches
[ ]	Matches one character in a range of valid characters.
[^]	Matches a character that isn’t in the given range. [^A-B]
.	Any character except a newline.
\s	Any whitespace character (such as a tab or space).
\S	Any non-whitespace character.
\d	Any digit character
\D	Any character that isn’t a digit.
\w	Any “word” character (letter, number, or underscore).
\W	Any character that isn’t a “word” character (letter, number, or underscore).

Master Page

- Chapter 12 – Page 369

- [a-z][A-Z]+   - one lower case character followed by one or more uppercase character
- [a-zA-Z]+   - one or more lowercase/uppercase character
- [a-zA-Z]+[ ] [a-zA-Z]+ - one or more lowercase/uppercase character, followed by a space, followed by one or more lowercase/uppercase character