enosis learning

Creating Software Professionals

ASP.NET

www.enosislearning.com

course@enosislearning.com

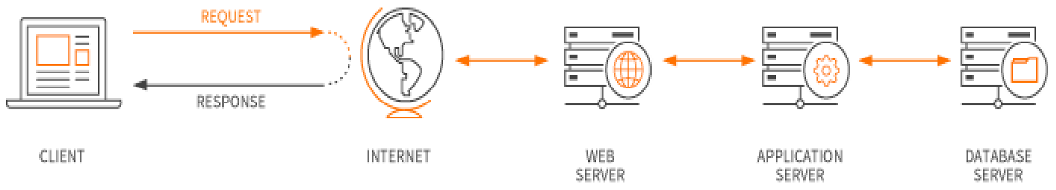
Contents

Chapter 1

INTRODUCTION

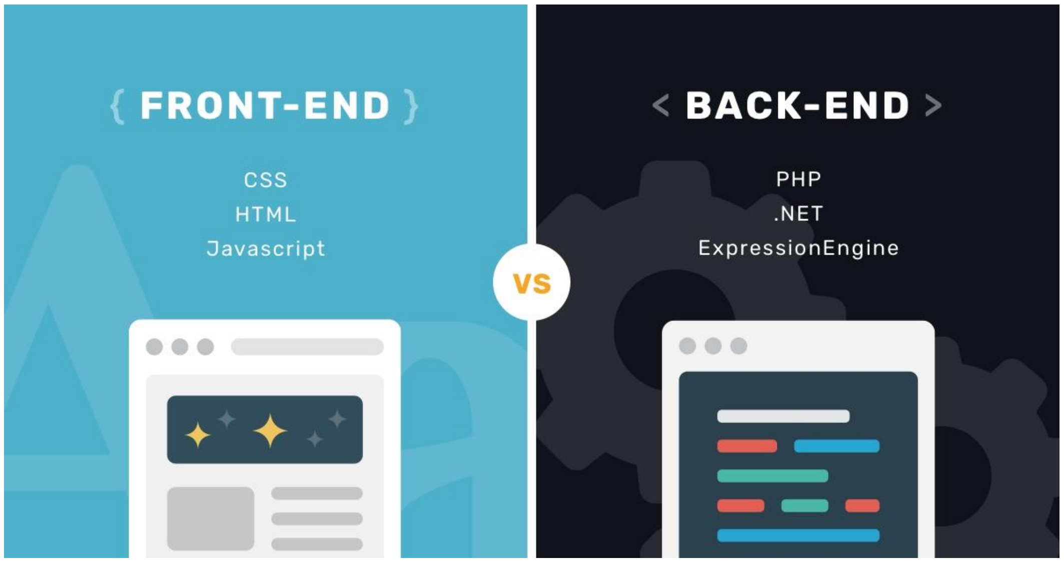
What is Web Application?

- Web application or web app is a client-server program which run on web browser.
- Example: - online shopping website.



- Web application uses a combination of server-side scripts i.e php, ASP etc to handle the storage and retrieval of the information and client-side scripts i.e JavaScript and HTML to present information to users.
- Web Application can be static (no processing is required) or dynamic (which required server side processing).

- Two Broad division of web Development are
  - Front End Development
  - Back End Development



What is Front End Development?

- It is also known as client-side development.
- What user sees when they load a web application like content, design is known as front end development.
- Done using html, css and JS.

What is Back End Development?

- It is also known as Server-side development.
- What goes behind the scenes of web application is known as Back end development.
- It uses database to generate the front end.
- They are written in PHP, ASP.NET, and Java etc.

What is ASP?

- Server-side scripting technology.
- Microsoft created to ease the development of interactive Web applications.
- ASP you can use client-side scripts as well as server-side scripts.
- ASP provides solutions for transaction processing and managing session state.
- Asp is one of the most successful languages used in web development.

Problem with ASP

There are many problems with ASP if you think of needs for Today's powerful Web applications.

- **Interpreted and Loosely-Typed Code**
  ASP scripting code is usually written in languages such as Script or VBScript. The script-execution engine that Active Server Pages relies on interprets code line by line, every time the page is called.

- **Mixes layout (HTML) and logic (scripting code)**
  ASP files frequently combine script code with HTML.

- **Limited Development and Debugging Tools**
  Microsoft Visual InterDev, Macromedia Visual UltraDev, and other tools have attempted to increase the productivity of ASP programmers by providing graphical development environments.

- **No real state management**
  Session state is only maintained if the client browser supports cookies. Session state information can only be held by using the ASP Session object. And you have to implement additional code if you, for example, want to identify a user.

- **Update files only when server is down**
  If your Web application makes use of components, copying new files to your application should only be done when the Web server is stopped.
- **Obscure Configuration Settings**
  The configuration information for an ASP web application (such as session state and server timeouts) is stored in the IIS metabase. Because the metabase is stored in a proprietary format, it can only be modified on the server machine with utilities such as the Internet Service Manager. With limited support for programmatically manipulating or extracting these settings, it is often an arduous task to port an ASP application from one server to another.

What is ASP.NET?

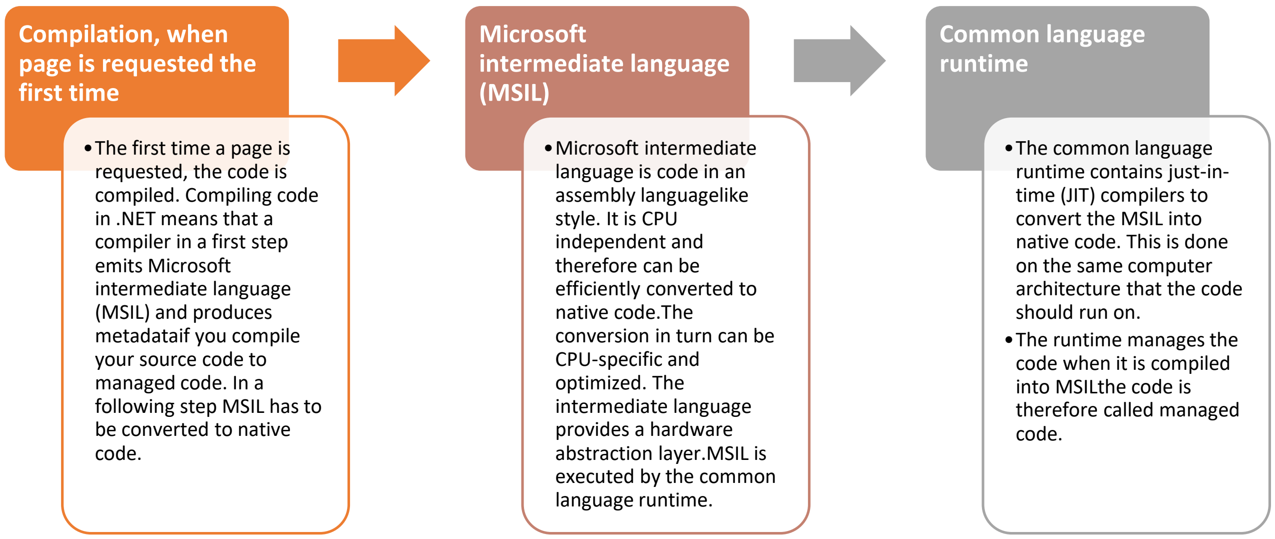
- Open Source Server side web application.
- Framework designed for web development to produce dynamic web pages.
- Developed by Microsoft in 2002.
- Because it has evolved from ASP, ASP.NET looks very similar to its predecessor but only at first sight. Some items look very familiar, and they remind us of ASP. But concepts like Web Forms, Web Services, or Server Controls gives ASP.NET the power to build real Web applications.

Advantages of ASP.NET

- Separation of Code from HTML
- Support for compiled languages
- Use services provided by the .NET Framework
- Graphical Development Environment
- State management
- Update files while the server is running!
- XML-Based Configuration Files

ASP.NET Execution Cycle

- A request for an .aspx file causes the ASP.NET runtime to parse the file for code that can be compiled.
- It then generates a page class that instantiates and populates a tree of server control instances. This page class represents the ASP.NET page.
- Now an execution sequence is started in which, for example, the ASP.NET page walks its entire list of controls, asking each one to render itself.
- The controls paint themselves to the page. This means they make themselves visible by generating HTML output to the browser client.



Compilation, when page is requested the first time

- The first time a page is requested, the code is compiled. Compiling code in .NET means that a compiler in a first step emits Microsoft intermediate language (MSIL) and produces metadataif you compile your source code to managed code. In a following step MSIL has to be converted to native code.

Microsoft intermediate language (MSIL)

- Microsoft intermediate language is code in an assembly languagelike style. It is CPU independent and therefore can be efficiently converted to native code.The conversion in turn can be CPU-specific and optimized. The intermediate language provides a hardware abstraction layer.MSIL is executed by the common language runtime.

Common language runtime

- The common language runtime contains just-in-time (JIT) compilers to convert the MSIL into native code. This is done on the same computer architecture that the code should run on.
- The runtime manages the code when it is compiled into MSILthe code is therefore called managed code.

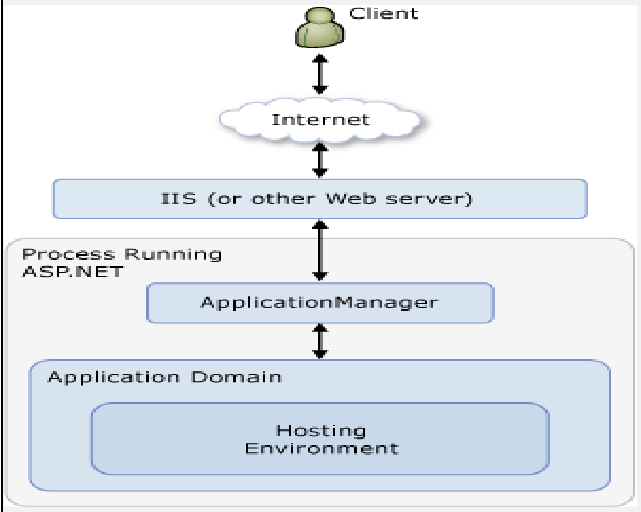
Chapter 2

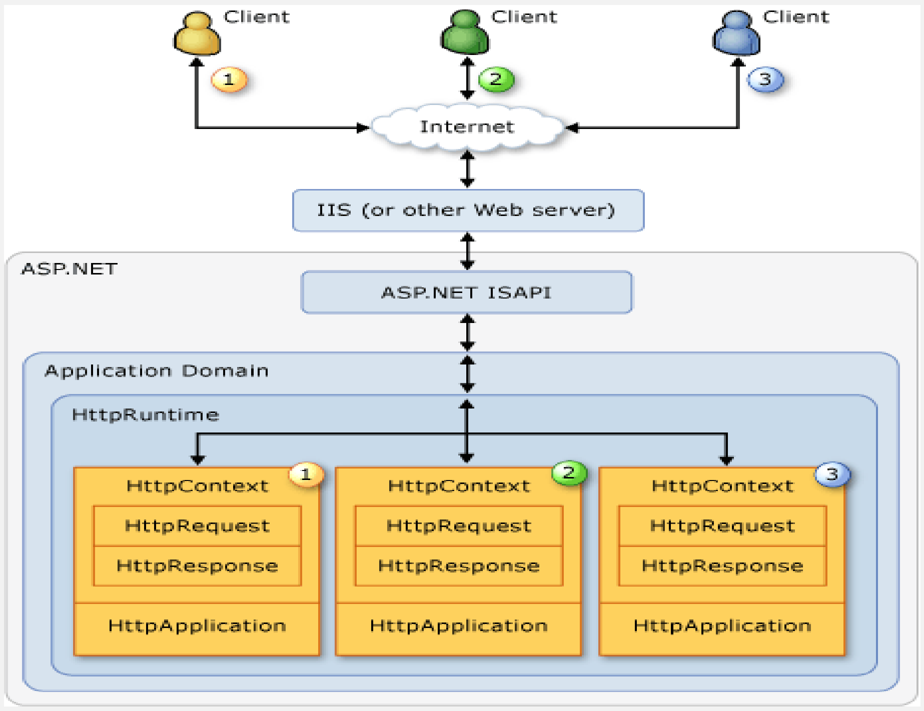
ASP.NET LIFE CYCLE

ASP.NET Life Cycle

- The following table describes the stages of the ASP.NET application life cycle.

STAGE	DESCRIPTION
User requests an application resource from the Web server.	The life cycle of an ASP.NET application starts with a request sent by a browser to the Web server (for ASP.NET applications, typically IIS). ASP.NET is an ISAPI extension under the Web server. When a Web server receives a request, it examines the file name extension of the requested file, determines which ISAPI extension should handle the request, and then passes the request to the appropriate ISAPI extension. ASP.NET handles file name extensions that have been mapped to it, such as .aspx, .ascx, .ashx, and .asmx. **Note** If a file name extension has not been mapped to ASP.NET, then ASP.NET will not receive the request. This is important to understand for applications that use ASP.NET authentication. For example, because .htm files are typically not mapped to ASP.NET, ASP.NET will not perform authentication or authorization checks on requests for .htm files. Therefore, even if a file contains only static content, if you want ASP.NET to check authentication, create the file using a file name extension mapped to ASP.NET, such as .aspx. **Note** If you create a custom handler to service a particular file name extension, you must map the extension to ASP.NET in IIS and also register the handler in your application's Web.config file.
ASP.NET receives the	When ASP.NET receives the first request for any resource in an application, a class named ApplicationManager creates an application domain. Application domains

first request for the application.	provide isolation between applications for global variables and allow each application to be unloaded separately. Within an application domain, an instance of the class named HostingEnvironment is created, which provides access to information about the application such as the name of the folder where the application is stored. The following diagram illustrates this relationship:  Client Internet IIS (or other Web server) Process Running ASP.NET ApplicationManager Application Domain Hosting Environment ASP.NET also compiles the top-level items in the application if required, including application code in the App_Code folder. For more information, see "Compilation Life Cycle" later in this topic.
ASP.NET core objects are created for each request.	After the application domain has been created and the **HostingEnvironment** object instantiated, ASP.NET creates and initializes core objects such as HttpContext, HttpRequest, and HttpResponse. The **HttpContext** class contains objects that are specific to the current application request, such as the **HttpRequest** and **HttpResponse** objects. The **HttpRequest** object contains information about the current request, including cookies and browser information. The **HttpResponse** object contains the response that is sent to the client, including all rendered output and cookies.
An HttpApplication object is	After all core application objects have been initialized, the application is started by creating an instance of the **HttpApplication** class. If the application has a Global.asax

assigned to the request	file, ASP.NET instead creates an instance of the Global.asax class that is derived from the **HttpApplication** class and uses the derived class to represent the application. **Note** The first time an ASP.NET page or process is requested in an application, a new instance of **HttpApplication** is created. However, to maximize performance, **HttpApplication** instances might be reused for multiple requests. When an instance of **HttpApplication** is created, any configured modules are also created. For instance, if the application is configured to do so, ASP.NET creates a SessionStateModule module. After all configured modules are created, the **HttpApplication** class's Init method is called. The following diagram illustrates this relationship:  Client Client Client 1 2 3 Internet IIS (or other Web server) ASP.NET ASP.NET ISAPI Application Domain HttpRuntime HttpContext HttpRequest HttpResponse HttpApplication

The request is processed by the HttpApplication pipeline.	The following events are executed by the **HttpApplication** class while the request is processed. The events are of particular interest to developers who want to extend the **HttpApplication** class. 1. Validate the request, which examines the information sent by the browser and determines whether it contains potentially malicious markup. For more information, see ValidateRequest and Script Exploits Overview. 2. Perform URL mapping, if any URLs have been configured in the UrlMappingsSection section of the Web.config file. 3. Raise the BeginRequest event. 4. Raise the AuthenticateRequest event. 5. Raise the PostAuthenticateRequest event. 6. Raise the AuthorizeRequest event. 7. Raise the PostAuthorizeRequest event. 8. Raise the ResolveRequestCache event. 9. Raise the PostResolveRequestCache event. 10. Based on the file name extension of the requested resource (mapped in the application's configuration file), select a class that implements IHttpHandler to process the request. If the request is for an object (page) derived from the Page class and the page needs to be compiled, ASP.NET compiles the page before creating an instance of it. 11. Raise the PostMapRequestHandler event. 12. Raise the AcquireRequestState event. 13. Raise the PostAcquireRequestState event. 14. Raise the PreRequestHandlerExecute event. 15. Call the ProcessRequest method (or the asynchronous version BeginProcessRequest) of the appropriate **IHttpHandler** class for the request. For example, if the request is for a page, the current page instance handles the request. 16. Raise the PostRequestHandlerExecute event. 17. Raise the ReleaseRequestState event. 18. Raise the PostReleaseRequestState event.

	19. Perform response filtering if the Filter property is defined. 20. Raise the UpdateRequestCache event. 21. Raise the PostUpdateRequestCache event. 22. Raise the EndRequest event.

ASP.NET Life Cycle Events and Global.asax file

- During the application life cycle, the application raises events that you can handle and calls particular methods that you can override. To handle application events or methods, you can create a file named Global.asax in the root directory of your application.
- If you create a Global.asax file, ASP.NET compiles it into a class derived from the HttpApplication class, and then uses the derived class to represent the application.
- An instance of HttpApplication processes only one request at a time. This simplifies application event handling because you do not need to lock non-static members in the application class when you access them. This also allows you to store request-specific data in non-static members of the application class. For example, you can define a property in the Global.asax file and assign it a request-specific value.
- ASP.NET automatically binds application events to handlers in the Global.asax file using the naming convention Application_event, such as Application_BeginRequest. This is similar to the way that ASP.NET page methods are automatically bound to events, such as the page's Page_Load event.
- TheApplication_Start and Application_End methods are special methods that do not represent HttpApplication events. ASP.NET calls them once for the lifetime of the application domain, not for each HttpApplication instance.

Application Life Cycle Events and Methods

Application_Start

- Called when the first resource (such as a page) in an ASP.NET application is requested. The Application_Start method is called only one time during the life cycle of an application. You can use this method to perform startup tasks such as loading data into the cache and initializing static values.

Application_event

- Raised at the appropriate time in the application life cycle, as listed in the application life cycle table earlier in this topic.
- Application_Error can be raised at any phase in the application life cycle.
- Application_EndRequest is the only event that is guaranteed to be raised in every request, because a request can be short-circuited. For example, if two modules handle the Application_BeginRequest event and the first one throws an exception, the Application_BeginRequest event will not be called for the second module. However, the Application_EndRequest method is always called to allow the application to clean up resources.

Init

- Called once for every instance of the HttpApplication class after all modules have been created.

Dispose

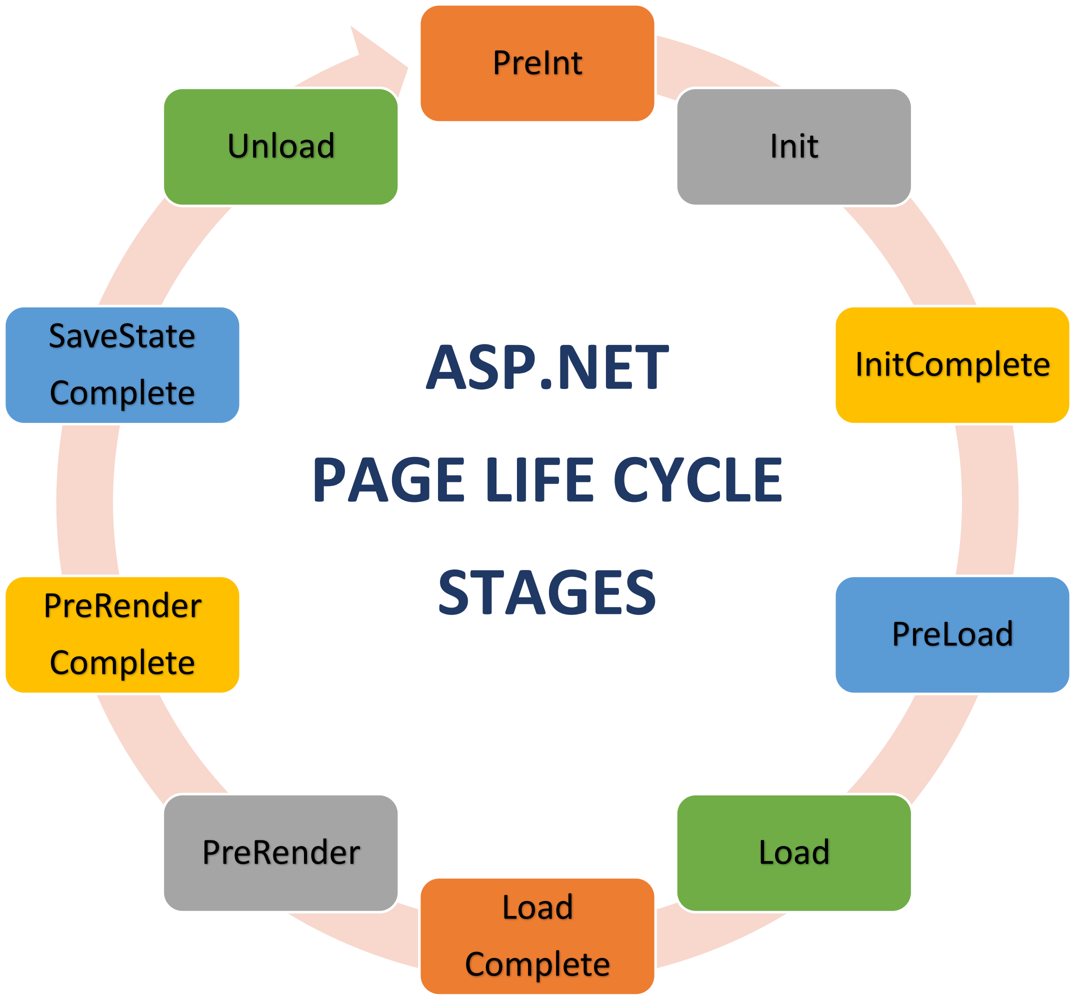
- Called before the application instance is destroyed. You can use this method to manually release any unmanaged resources.

Application_End

- Called once per lifetime of the application before the application is unloaded.

ASP.NET Page Life Cycle

- The requesting of an ASP.NET page triggers a sequence of events that encompass the page life cycle. The Web browser sends a post request to the Web server. The Web server recognizes the ASP.NET file extension for the requested page and sends the request to the HTTP Page Handler class. The following list is a sampling of these events, numbered in the order in which they are triggered.



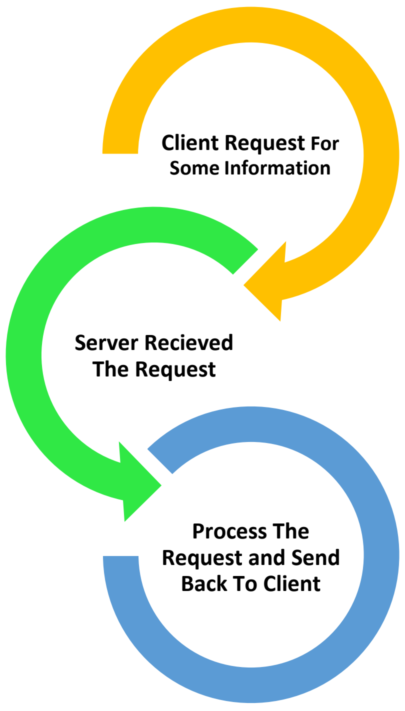
- **PreInt:** This is the entry point of the ASP.NET page life cycle - it is the pre-initialization, so you have access to the page before it is initialized. Controls can be created within this event. Also, master pages and themes can be accessed. You can check the IsPostBack property here to determine if it is the first time a page has been loaded.
- **Init:** This event fires when all controls on the page have been initialized and skin settings have been applied. You can use this event to work with control properties. The Init event of the page is not fired until all control Init events have triggered - this occurs from the bottom up.
- **InitComplete:** This event fires once all page and control initializations complete. This is the last event fired where ViewState is not set, so ViewState can be manipulated in this event.
- **PreLoad:** This event is triggered when all ViewState and Postback data have been loaded for the page and all of its controls - ViewState loads first, followed by Postback data.
- **Load:** This is the first event in the page life cycle where everything is loaded and has been set to its previous state (in the case of a Postback). The page Load event occurs first followed by the Load event for all controls (recursively). This is where most coding is done, so you want to check the IsPostBack property to avoid unnecessary work.
- **LoadComplete:** This event is fired when the page is completely loaded. Place code here that requires everything on the page to be loaded.
- **PreRender:** This is the final stop in the page load cycle where you can make changes to page contents or controls. It is fired after all Postback events and before ViewState has been saved. Also, this is where control databinding occurs.
- **PreRenderComplete:** This event is fired when PreRender is complete. Each control raises this event after databinding (when a control has its DataSourceID set).
- SaveStateComplete: This is triggered when view and control state have been saved for the page and all controls within it. At this point, you can make changes in the rendering of the page, but those changes will not be reflected on the next page Postback since view state is already saved.
- **Unload:** This event fires for each control and then the page itself. It is fired when the HTML for the page is fully rendered. This is where you can take care of cleanup tasks, such as properly closing and disposing database connections.

Chapter 3

WEB SERVER

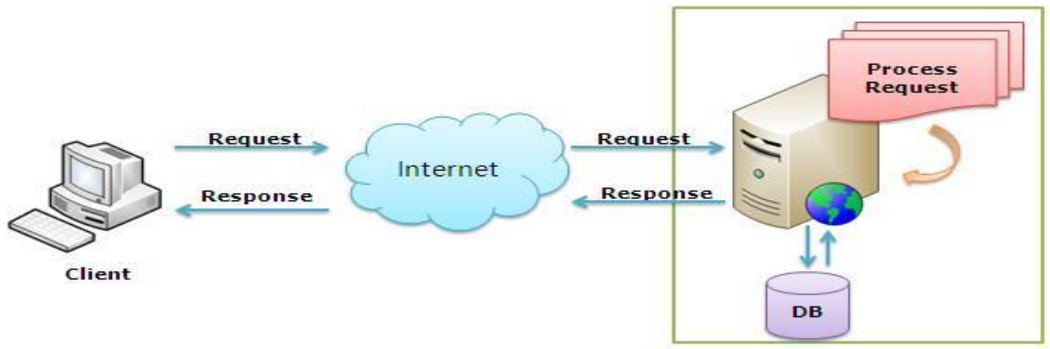
What is Web Server?

- When we run our ASP.NET Web Application from visual studio IDE, VS Integrated ASP.NET Engine is responsible to execute all kind of asp.net requests and responses. The process name is "WebDev.WebServer.Exe" which actually take care of all request and response of an web application which is running from Visual Studio IDE.
- Now, the name “Web Server” come into picture when we want to host the application on a centralized location and wanted to access from many locations. Web server is responsible for handle all the requests that are coming from clients, process them and provide the responses.


Client Request For Some Information
Server Recieved The Request
Process The Request and Send Back To Client

What is IIS?

- IIS (Internet Information Server) is one of the most powerful web servers from Microsoft that is used to host your ASP.NET Web application. IIS has its own ASP.NET Process Engine to handle the ASP.NET request. So, when a request comes from client to server, IIS takes that request and process it and send response back to clients.



How Asp.Net Request Is Process at IIS

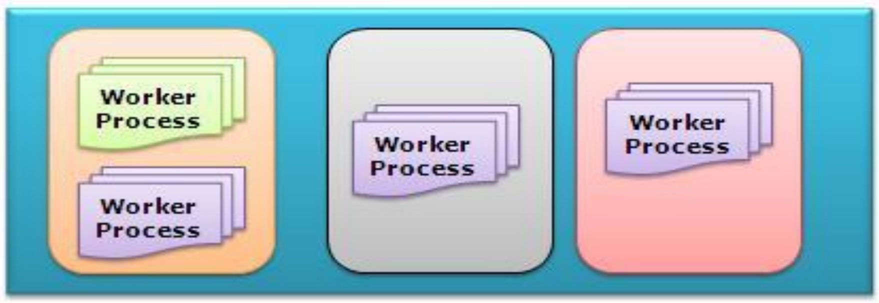
- Now let's have a look how they do things internally. Before we move ahead, you have to know about two main concepts
    - Worker Process
    - *Application Pool*

Worker Process

- Worker Process (w3wp.exe) runs the ASP.Net application in IIS.
- This process is responsible to manage all the request and response that are coming from client system.
- All the ASP.Net functionality runs under the scope of worker process.
- When a request comes to the server from a client worker process is responsible to generate the request and response. In a single word we can say worker process is the heart of ASP.NET Web Application which runs on IIS.

Application Pool

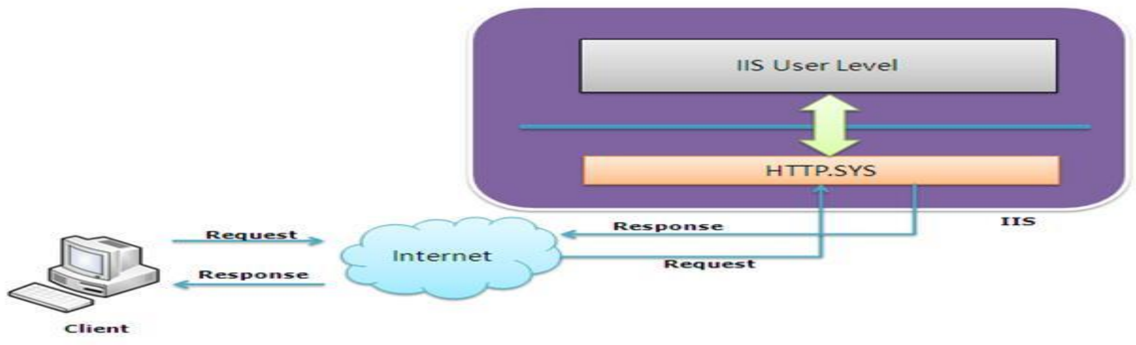
- Application pool is the container of worker process.
- Application pools is used to separate sets of IIS worker processes that share the same configuration.
- Application pools enables a better security, reliability, and availability for any web application.
- The worker process serves as the process boundary that separates each application pool so that when one worker process or application is having an issue or recycles, other applications or worker processes are not affected. This makes sure that a particular web application doesn't not impact other web application as they are configured into different application pools.



- Application Pool with multiple worker process is called "Web Garden".
- Now, I have covered all the basic stuff like Web server, Application Pool, Worker process. Now let's have look how IIS process the request when a new request comes up from client.

- If we look into the IIS 6.0 Architecture, we can divided them into Two Layer
  - *Kernel Mode*
  - *User Mode*

Now, ***Kernel mode*** is introduced with IIS 6.0, which contains the **HTTP.SYS**. So whenever a request comes from Client to Server, it will hit *HTTP.SYS* First.



Chapter 4

ASP.NET CONTROLS

Controls in Web Development

- Controls are small building blocks of the graphical user interface, which include text boxes, buttons, check boxes, list boxes, labels, and numerous other tools. Using these tools, the users can enter data, make selections and indicate their preferences.
- Controls are also used for structural jobs, like validation, data access, security, creating master pages, and data manipulation.
- Asp.Net has several types of web controls which are as follows:
    - Html Controls
    - ASP.NET Html Server Controls
    - ASP.NET Server Controls
    - ASP.NET Validation Controls

HTML Controls

Basic Tags

<html></html>	Creates an HTML document
<head></head>	Sets off the title and other information that isn’t displayed on the web page itself
<body></body>	Sets off the visible portion of the document

Body Attributes

<body bgcolor="pink">	Sets the background color, using name or hex value
<body text="black">	Sets the text color, using name or hex value
<body link="blue">	Sets the color of links, using name or hex value

<body vlink="#ff0000">	Sets the color of followed links, using name or hex value
<body alink="#00ff00">	Sets the color of links on click
<body ondragstart="return false" onselectstart="return false">	Disallows text selection with the mouse and keyboard

Text Tags

<pre></pre>	Creates preformatted text
<hl></hl>	Creates the largest headline
<h6></h6>	Creates the smallest headline
<b></b>	Creates bold text
<i></i>	Creates italic text
<tt></tt>	Creates teletype, or typewriter-style text
<cite></cite>	Creates a citation, usually italic
<em></em>	Emphasizes a word (with italic or bold)
<strong></strong>	Emphasizes a word (with italic or bold)
<font size="3"></font>	Sets size of font, from 1 to 7
<font color="green"></font>	Sets font color, using name or hex value

Links

<a href="URL"></a>	Creates a hyperlink
<a href="mailto:EMAIL"></a>	Creates a mailto link
<a href="URL"><imgsrc="URL"></a>	Creates an image/link
<a name="NAME"></a>	Creates a target location within a document

<a href="#NAME"></a>	Links to that target location from elsewhere in the document

Formatting

<p></p>	Creates a new paragraph
<p align="left">	Aligns a paragraph to the left (default), right, or center.
<br>	Inserts a line break
<blockquote></blockquote>	Indents text from both sides
<dl></dl>	Creates a definition list
<dt>	Precedes each definition term
<dd>	Precedes each definition
<ol></ol>	Creates a numbered list
<ul></ul>	Creates a bulleted list
<li></li>	Precedes each list item, and adds a number or symbol depending upon the type of list selected
<div align="left">	A generic tag used to format large blocks of HTML, also used for stylesheets
<imgsrc="name">	Adds an image
<imgsrc="name" align="left">	Aligns an image: left, right, center; bottom, top, middle
<imgsrc="name" border="1">	Sets size of border around an image
<hr />	Inserts a horizontal rule
<hr size="3" />	Sets size (height) of rule
<hr width="80%" />	Sets width of rule, in percentage or absolute value
<hrnoshade />	Creates a rule without a shadow

Tables

<table></table>	Creates a table
<tr></tr>	Sets off each row in a table
<td></td>	Sets off each cell in a row
<th></th>	Sets off the table header (a normal cell with bold, centered text)

Table Attributes

<table border="1">	Sets width of border around table cells
<table cellspacing="1">	Sets amount of space between table cells
<table cellpadding="1">	Sets amount of space between a cell's border and its contents
<table width="500" or "80%">	Sets width of table, in pixels or as a percentage of document width
<tr align="left"> or <td align="left">	Sets alignment for cell(s) (left, center, or right)
<tr valign="top"> or <td valign="top">	Sets vertical alignment for cell(s) (top, middle, or bottom)
<td colspan="2">	Sets number of columns a cell should span (default=1)
<td rowspan="4">	Sets number of rows a cell should span (default=1)
<tdnowrap>	Prevents the lines within a cell from being broken to fit

Frames

<frameset></frameset>	Replaces the <body> tag in a frames document; can also be nested in other framesets

<frameset rows="value,value">	Defines the rows within a frameset, using number in pixels, or percentage of width
<frameset cols="value,value">	Defines the columns within a frameset, using number in pixels, or percentage of width
<frame>	Defines a single frame — or region — within a frameset
<noframes></noframes>	Defines what will appear on browsers that don't support frames

Frames Attributes

<frame src="URL">	Specifies which HTML document should be displayed
<frame name="name">	Names the frame, or region, so it may be targeted by other frames
<frame marginwidth="value">	Defines the left and right margins for the frame; must be equal to or greater than 1
<frame marginheight="value">	Defines the top and bottom margins for the frame; must be equal to or greater than 1
<frame scrolling="value">	Sets whether the frame has a scrollbar; value may equal "yes," "no," or "auto." The default, as in ordinary documents, is auto.
<frame noresize="noresize">	Prevents the user from resizing a frame

Forms

For functional forms, you'll have to run a script. The HTML just creates the appearance of a form.

<form></form>	Creates all forms

<select multiple name="NAME" size=?></select>	Creates a scrolling menu. Size sets the number of menu items visible before you need to scroll.
<select name="NAME"></select>	Creates a pulldown menu
<option>	Sets off each menu item
<textarea name="NAME" cols=40 rows=8></textarea name>	Creates a text box area. Columns set the width; rows set the height.
<input type="checkbox" name="NAME">	Creates a checkbox. Text follows tag.
<input type="radio" name="NAME" value="x">	Creates a radio button. Text follows tag
<input type="text" name="NAME" size=20>	Creates a one-line text area. Size sets length, in characters.
<input type="submit" value="NAME">	Creates a Submit button
<button type="submit">Submit</button>	Creates an actual button that is clicked
<input type="image" border=0 name="NAME" src="name.gif">	Creates a Submit button using an image
<input type="reset">	Creates a Reset button

HTML 5 adds a few (and HTML5 seems to not use some of the ones above as well):

```
<audio src=url>
<button formaction=url>
<command icon=url>
<embed src=url>
<html manifest=url>
<input formaction=url>
<source src=url>
<video poster=url> and <video src=url>
```

These aren't necessarily simple URLs:

<object archive=url> or <object archive="url1 url2 url3">

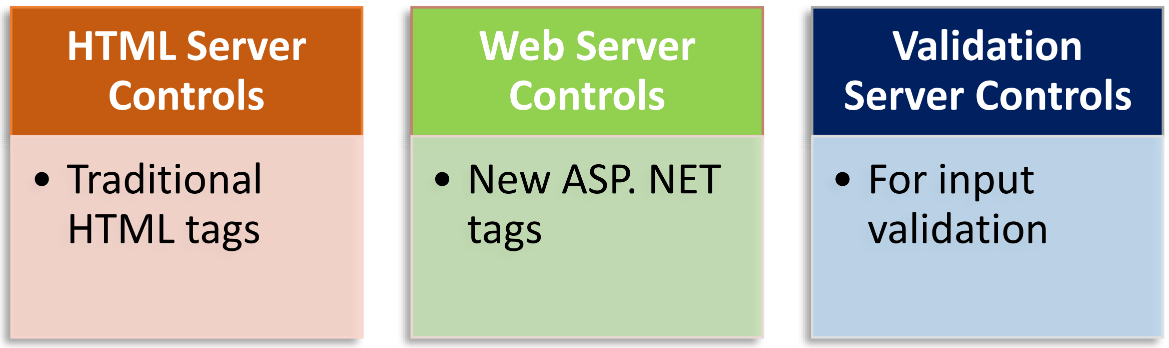
<applet archive=url> or <applet archive=url1,url2,url3>

<meta http-equiv="refresh" content="seconds; url">

In addition, the style attribute can contain css declarations with one or several urls. For example: <div style="background: url(image.png)">

ASP.NET Controls

- Server Controls are the tags that are understood by the server. There are basically three types of server controls



.

ASP.NET HTML Server Controls

- ASP.NET provides a way to work with HTML Server controls on the server side; programming with a set of controls collectively is called HTML Controls.
- These controls are grouped together in the Visual Studio Toolbox in the HTML Control tab. The markup of the controls are similar to the HTML control.
- These controls are basically the original HTML controls but enhanced to enable server side processing.
- HTML elements in ASP. NET files are, by default, treated as text. To make these elements programmable, add a runat="server" attribute to the HTML element. This attribute indicates that the element should be treated as a server control.

Note: All HTML server controls must be within a <form> tag with the runat="server" attribute. The runat="server" attribute indicates that the form should be processed on the server. It also indicates that the enclosed controls can be accessed by server scripts.

- The System.Web.UI.HtmlControls.HtmlControl base class contains all of the common properties. HTML server controls derive from this class.

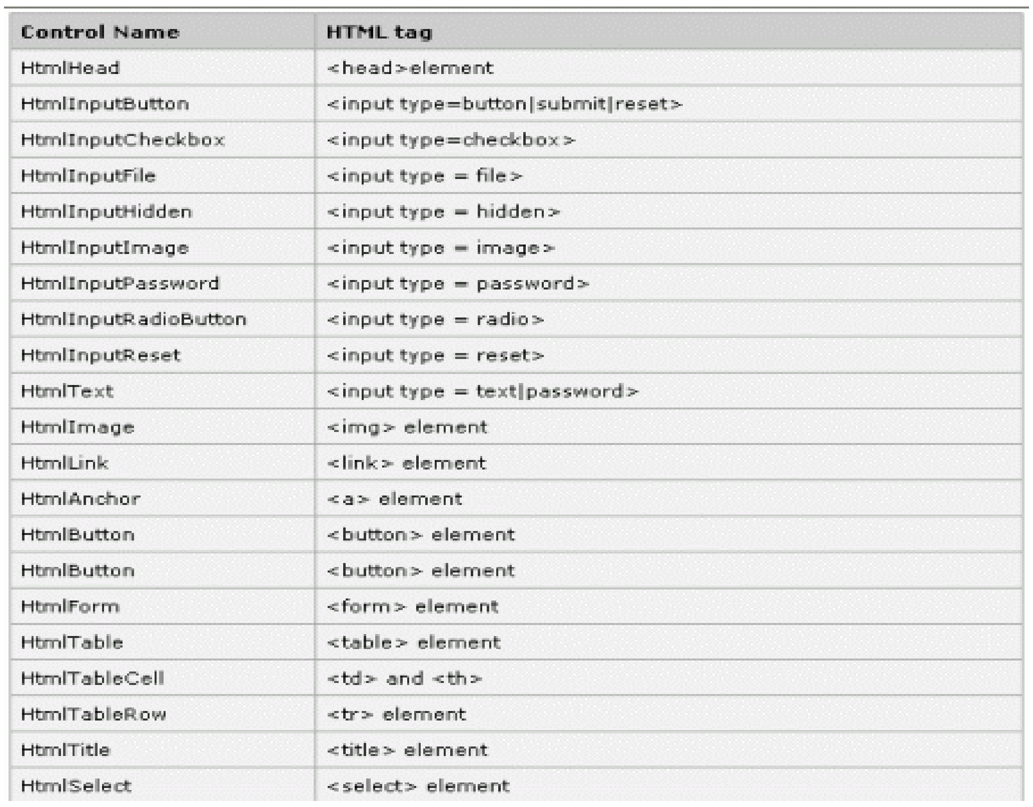
For example, consider the HTML input control:

<input type=”text” size=”40”/>

It could be converted to a server control, by adding the runat and id attribute:

<input type=”text” id=”testtext” size=”40” runat=”server”>

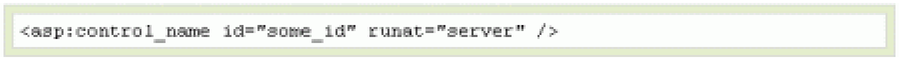
- The following table describes the HTML se



Control Name	HTML tag
HtmlHead	<head>element
HtmlInputButton	<input type=button|submit|reset>
HtmlInputCheckbox	<input type=checkbox>
HtmlInputFile	<input type = file>
HtmlInputHidden	<input type = hidden>
HtmlInputImage	<input type = image>
HtmlInputPassword	<input type = password>
HtmlInputRadioButton	<input type = radio>
HtmlInputReset	<input type = reset>
HtmlText	<input type = text|password>
HtmlImage	<img> element
HtmlLink	<link> element
HtmlAnchor	<a> element
HtmlButton	<button> element
HtmlButton	<button> element
HtmlForm	<form> element
HtmlTable	<table> element
HtmlTableCell	<td> and <th>
HtmlTableRow	<tr> element
HtmlTitle	<title> element
HtmlSelect	<select> element

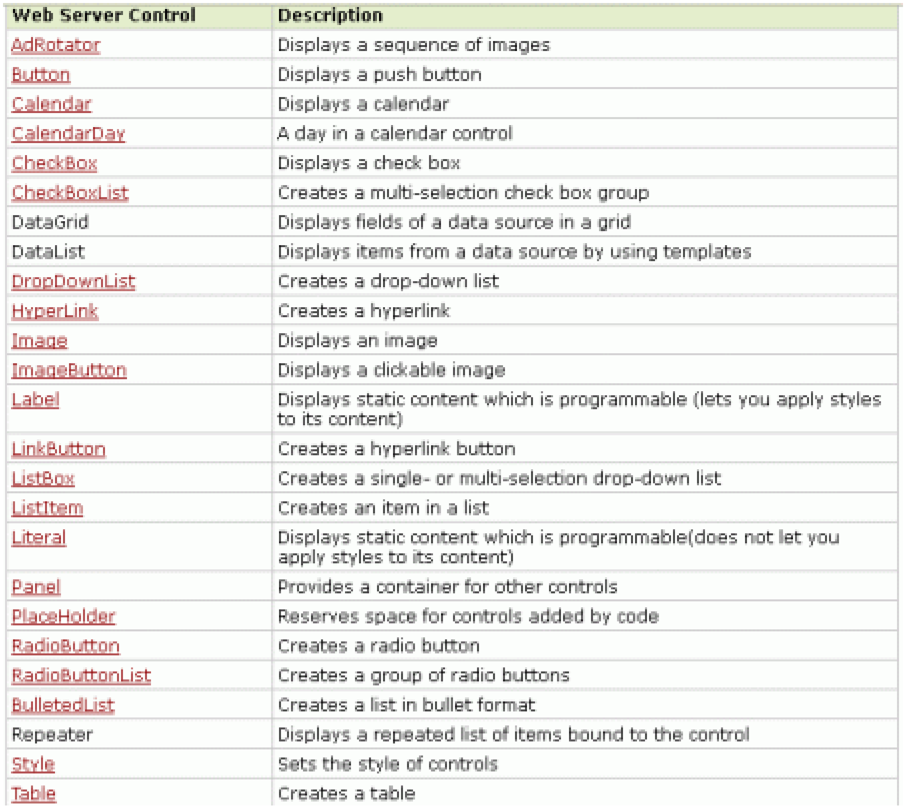
ASP.NET Web Server Controls

- Web server controls are special ASP. NET tags understood by the server.
- Like HTML server controls, Web server controls are also created on the server and they require a runat="server" attribute to work.
- However, Web server controls do not necessarily map to any existing HTML elements and they may represent more complex elements.
- Mostly all Web Server controls inherit from a common base class, namely the WebControl class defined in the System.Web.UI.WebControls namespace.
- The syntax for creating a Web server control is:



```
<asp:control_name id="some_id" runat="server" />
```

- The following table describes the WEB server controls:

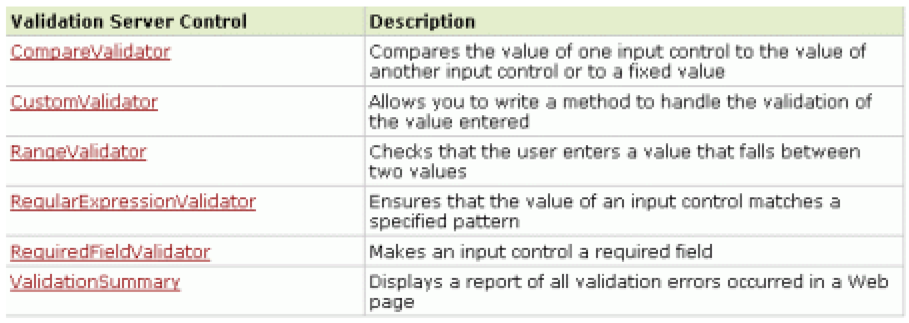


Web Server Control	Description
AdRotator	Displays a sequence of images
Button	Displays a push button
Calendar	Displays a calendar
CalendarDay	A day in a calendar control
CheckBox	Displays a check box
CheckBoxList	Creates a multi-selection check box group
DataGrid	Displays fields of a data source in a grid
DataList	Displays items from a data source by using templates
DropDownList	Creates a drop-down list
HyperLink	Creates a hyperlink
Image	Displays an image
ImageButton	Displays a clickable image
Label	Displays static content which is programmable (lets you apply styles to its content)
LinkButton	Creates a hyperlink button
ListBox	Creates a single- or multi-selection drop-down list
ListItem	Creates an item in a list
Literal	Displays static content which is programmable(does not let you apply styles to its content)
Panel	Provides a container for other controls
PlaceHolder	Reserves space for controls added by code
RadioButton	Creates a radio button
RadioButtonList	Creates a group of radio buttons
BulletedList	Creates a list in bullet format
Repeater	Displays a repeated list of items bound to the control
Style	Sets the style of controls
Table	Creates a table

ASP.NET Validation Server Controls

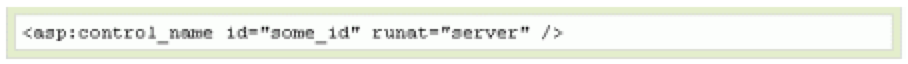
- After you create a web form, you should make sure that mandatory fields of the form elements such as login name and password are not left blank; data inserted is correct and is within the specified range. Validation is the method of scrutinizing (observing) that the user has entered the correct values in input fields.
- A Validation server control is used to validate the data of an input control. If the data does not pass validation, it will display an error message to the user.
- In ASP. NET you can use ASP. NET Validation Controls while creating the form and specify what ASP. NET Validation Controls you want to use and to which server control you want bind this.
- Validation Controls are derived from a common base class and share a common set of properties and methods. You just have to drag and drop the ASP. NET Validation Control in the web form and write one line of code to describe its functionality.
- This reduces the developer time from writing JavaScript for each type of validation. Moreover, through ASP. NET Validation Controls if any invalid data is entered the browser itself detects the error on the client side and displays the error without requesting the server. This is another advantage because it reduces the server load.

Some Server Validation controls are:



Validation Server Control	Description
CompareValidator	Compares the value of one input control to the value of another input control or to a fixed value
CustomValidator	Allows you to write a method to handle the validation of the value entered
RangeValidator	Checks that the user enters a value that falls between two values
RegularExpressionValidator	Ensures that the value of an input control matches a specified pattern
RequiredFieldValidator	Makes an input control a required field
ValidationSummary	Displays a report of all validation errors occurred in a Web page

The syntax for creating a Validation server control is:



```
<asp:control_name id="some_id" runat="server" />
```

Validation Controls in Asp.Net

- Validation server controls are a series of controls that help you validate the data that the user enters into the other controls that are provided with ASP.NET. They determine whether the form can be processed based upon the rules that you define in the validation server controls.

Understanding the Difference between Server-Side and Client-Side Validation

- After the user enters data into a Web form, clicks the Submit button, and sends the form data to the server as a request, you can perform server-side validation on the data. If the data is incorrect or not valid, you can send back a response stating this.
- If, however, when the user clicks the Submit button, a scripting language that is part of the overall HTML page is initiated to check the validity of the data before it is sent to the server, this is client-side validation.
- The bad thing about server-side validation is that it requires trips back and forth to the server. This takes a lot of resources and makes for a slower-paced form for the user.
- The other option for form validation is to put some client-side JavaScript or VBScript at the top of the ASP page that checks if the information in the fields is correct. This takes care of the problem of making unnecessary trips to the server, but it requires another language to learn and manage.

Listing 1: Client-side JavaScript for form validation

```
<script language="javascript">
<!--   Function CheckForm(form)
{
for(varintCtr = 0; intCtr<= (form.elements.length - 5); ++intCtr)
  {
var temp = form.elements[intCtr];
if(temp.type == "text" &&temp.value == "")
    {
alert("Please Enter All Information!");
temp.focus();
return false;
    }
  }
return true;
}//-->
</script>
```

.Net Validation Controls

- You can also customize validation for your own needs. Then, if there are any errors in the form data, these validation server controls enable you to customize the display of error information on the browser.
- Available validation server controls

Validation Server Control	Description
RequiredFieldValidator	Ensures that the user does not skip a form entry field
CompareValidator	Allows for comparisons between the user's input and another item using a comparison operator (equals, greater than, less than, and so on)
RangeValidator	Checks the user's input based upon a lower- and upper-level range of numbers or characters
RegularExpressionValidator	Checks that the user's entry matches a pattern defined by a regular expression. This is a good control to use to check e-mail addresses and phone numbers
ValidationSummary	Displays all the error messages from the validators in one specific spot on the page

Not all button clicks are equal

- Normally, in a series of HTML form elements, there is some sort of Button, ImageButton, or LinkButton control on the page that submits the form and causes the validation to initiate.
- You may not want each and every button on the ASP.NET page to initiate validation.
- For instance, you might have a Cancel or Reset button on the Web page. If the user clicks one of these buttons, you don't want that button click to validate the contents contained in the Web form.

To prevent this, you have to set the CausesValidation property for the control to False.

```
<asp:Button id="Button1" runat="server" Text="Button"
CausesValidation="False"></asp:Button>
```

[illegible]

- The RequiredFieldValidator server control makes sure that the user enters something into the field that it is associated with in the form. You need to tie the RequiredFieldValidator to each control that is a required field in the form.

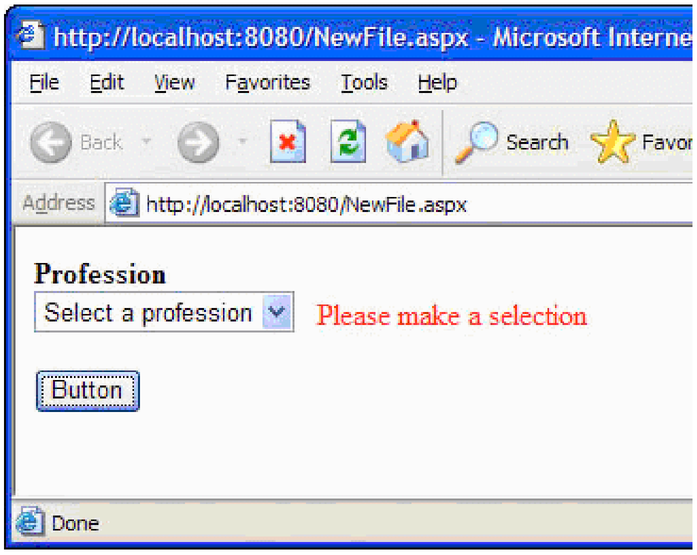
```
<asp:TextBox id="TextBox1" runat="server"></asp:TextBox>
&nbsp;
<asp:RequiredFieldValidator id="RequiredFieldValidator1" runat="server"
ErrorMessage="Required!" ControlToValidate="TextBox1"></asp:RequiredFieldValidator>
</p>
```

Using the InitialVale Property with a DropDownList Control

- A good way of using the InitialValue property is with the DropDownList server control. For instance, a drop-down list might have a default value that is not an empty value (by default, some text appears in it). An example of a drop-down list with the RequiredFieldValidator that uses the InitialValue property is illustrated in Listing 5.

Listing 5: Using the RequiredFieldValidator server control with the Drop-DownList server Control

```
<asp:DropDownList id="DropDownList1" runat="server">
<asp:ListItem Selected="True">Select a profession</asp:ListItem>
<asp:ListItem>Programmer</asp:ListItem>
<asp:ListItem>Lawyer</asp:ListItem>
<asp:ListItem>Doctor</asp:ListItem>
<asp:ListItem>Artist</asp:ListItem>
</asp:DropDownList>
&nbsp;
<asp:RequiredFieldValidator id="RequiredFieldValidator1"
runat="server" ErrorMessage="Please make a selection"
ControlToValidate="DropDownList1"
InitialValue="Select a profession">
</asp:RequiredFieldValidator>
```


http://localhost:8080/NewFile.aspx - Microsoft Interne
File Edit View Favorites Tools Help
Back
Search
Favor
Address http://localhost:8080/NewFile.aspx
Button
Done

Profession

Select a profession

Please make a selection

Figure 3: Validation error based upon the DropDownList control

The CompareValidator Control

- The CompareValidator server control compares the value entered into the form field to another field, a database value, or other value that you specify. When comparing against data types, you just set the Operator—DataTypeCheck. After that is done, you can set the Type attribute to String, Integer, Double, Date, or Currency in the CompareValidator control to make sure that the user's input into the field is the specified type.
- Using the CompareValidator server control, you can make comparisons between different controls within a form on your ASP.NET page. For example, if you want to compare what the user enters in the Password field to the entry in the Confirm Password field to see whether they are the same, you can use the CompareValidator server control.

Listing 6: Using the CompareValidator server control

```
<p>
    Password<br>
<asp:TextBox id="TextBox1" runat="server" TextMode="Password"></asp:TextBox>
&nbsp;
<asp:CompareValidator id="CompareValidator1" runat="server" ErrorMessage="Passwords
do not match!" ControlToValidate="TextBox2" ControlToCompare="TextBox1"></asp:Compa
reValidator>
</p>
<p>
            Confirm Password<br>
```

```
<asp:TextBox id="TextBox2" runat="server" TextMode="Password"></asp:TextBox>
</p>
```

Validating Against Constants

- You can also use the CompareValidator server control to make sure that the value typed into the form field by a user is valid when compared against a constant, some dynamic value that you retrieve, or that it adheres to a specific data type. Listing 7 shows an example of this.

Listing 7: Checking to make sure value entered is of a specific data type

```
Age:
<asp:TextBox id="TextBox1" runat="server" MaxLength="3"></asp:TextBox>
&nbsp;
<asp:CompareValidator id="CompareValidator1" runat="server"
ErrorMessage="You must enter a number"  ControlToValidate="TextBox1" Type="Integer"
  Operator="DataTypeCheck"></asp:CompareValidator>
```

- In this example, the user must enter an integer in the text box; otherwise, the CompareValidator server control fires and displays an error message on the page. By giving the Type property of the CompareValidator server control a value of Integer, you ensure that the value entered in the text box conforms to this .NET Framework data type.
- You also have the option of not only comparing values against specific data types, but also ensuring that values are valid when compared against certain constants (see Listing 8).

Listing 8: Comparing values against constants for validity

```
Age:
<asp:TextBox id="TextBox1" runat="server"></asp:TextBox>

&nbsp;

<asp:CompareValidator id="CompareValidator1" runat="server" Operator="GreaterThan"
```

```
ValueToCompare="18" ControlToValidate="TextBox1" ErrorMessage="You must be older th
an

  18 to join" Type="Integer">

</asp:CompareValidator>
```

The RangeValidator Control

- The RangeValidator server control is similar to the CompareValidator server control, but the RangeValidator server control compares what is entered into the form field with two values and makes sure that what was entered by the user is between these two specified values.

```
Marks:

<asp:TextBoxID="txtMarks"runat="server"/>

<asp:RangeValidatorID="rvMarks"runat="server"

ErrorMessage="Value is out of range"ControlToValidate="txtMarks"

Type="Integer"MinimumValue="0"MaximumValue="100"/>
```

- For instance, imagine that you have a text box where you want end users to enter their ages

Listing 9: Using the RangeValidator server control to work with a range of numbers

```
Age:
<asp:TextBox id="TextBox1" runat="server"></asp:TextBox>
&nbsp;
<asp:RangeValidator id="RangeValidator1" runat="server"
ControlToValidate="TextBox1" Type="Integer"
ErrorMessage="You must be between 30 and 40"
MaximumValue="40" MinimumValue="30"></asp:RangeValidator>
```

- In this case, the user should enter a value between 30 and 40 in the text box. If some number is entered that is outside of this range, the RangeValidator server control fires an error message and considers the form submission invalid.
- The Type property enables you to make comparisons against many different .NET Framework types, such as String, Integer, Double, Date, and Currency. These choices enable you to do a number of range comparisons. You can also use the Date value for the Type property to make sure that the entry is between specific date ranges.

Listing 10: Comparing an entry to a range of characters

- For example, if the user is entering her last name, and you want only people with last names starting with M and P to proceed, you can easily do this by using the RangeValidator server control, as illustrated in Listing 10.
- Last name:

```
<asp:TextBox id="TextBox1" runat="server"></asp:TextBox>

&nbsp;

<asp:RangeValidator id="RangeValidator1" runat="server"

ControlToValidate="TextBox1"

ErrorMessage="Your last name needs to be between M and P"

MaximumValue="Q" MinimumValue="M"></asp:RangeValidator>
```

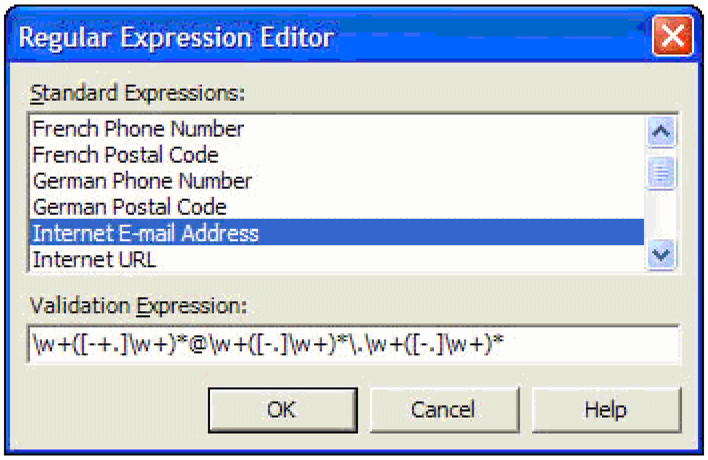
- In the example in Listing 10, the value is being checked against a range that is specified by using the MaximumValue and MinimumValue properties. In this example, that the Type property is not specified. In this case, it doesn't need to be specified because the default value of the Type property is String.

```
Age:

<asp:TextBoxid="TextBox4"runat="server"></asp:TextBox>

&nbsp;
```

```
<asp:RangeValidatorid="RangeValidator1"runat="server"

ControlToValidate="TextBox4"Type="Integer"

ErrorMessage="You must be between 30 and 40"

MaximumValue="40"MinimumValue="30"></asp:RangeValidator>

<br/>


  Last name:

<asp:TextBoxid="TextBox5"runat="server"></asp:TextBox>

&nbsp;

<asp:RangeValidatorid="RangeValidator2"runat="server"

ControlToValidate="TextBox5"

ErrorMessage="Your last name needs to be between M and P"

MaximumValue="Q"MinimumValue="M"></asp:RangeValidator>
```

The RegularExpressionValidator Control

- The RegularExpressionValidator server control is a validation control that enables you to check the user's input based on a pattern defined by a regular expression. In the past, these kinds of validations took a considerable amount of JavaScript coding. The RegularExpressionValidator control with ASP.NET saves coding time.



```
\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*
```

Figure 5: The Regular Expression Editor

- For an example of using the RegularExpressionValidator server control to make sure that a value entered in a text box is an e-mail address, look at Listing 11.

Listing 11: Validating an e-mail address

Email:

```
<asp:TextBox id="TextBox1" runat="server"></asp:TextBox>

&nbsp;

<asp:RegularExpressionValidator id="RegularExpressionValidator1"

runat="server" ControlToValidate="TextBox1"

ErrorMessage="You must enter an email address"

  ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">

</asp:RegularExpressionValidator>
```

Using Images for Your Error Messages

- One interesting way of showing your error messages when using validation controls is to use images along with text for identifying errors on your ASP.NET pages. This secret is not limited to the RegularExpressionValidator server control, but can be used with all the validation server controls.
- To use an image instead of text for your error messages, you create something similar to the code in Listing 13.

Listing 13: Using images for your validation messages

```
Email:
<asp:TextBox id="TextBox1" runat="server"></asp:TextBox>
&nbsp;
<asp:RegularExpressionValidator id="RegularExpressionValidator1"
runat="server" ControlToValidate="TextBox1"
ErrorMessage='<imgsrc="error.gif">'
  ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
</asp:RegularExpressionValidator>
```

- To use an image instead of text for your error messages, you use an HTML string that points to the image that you want to display for the value of the ErrorMessage property (see Figure 7).

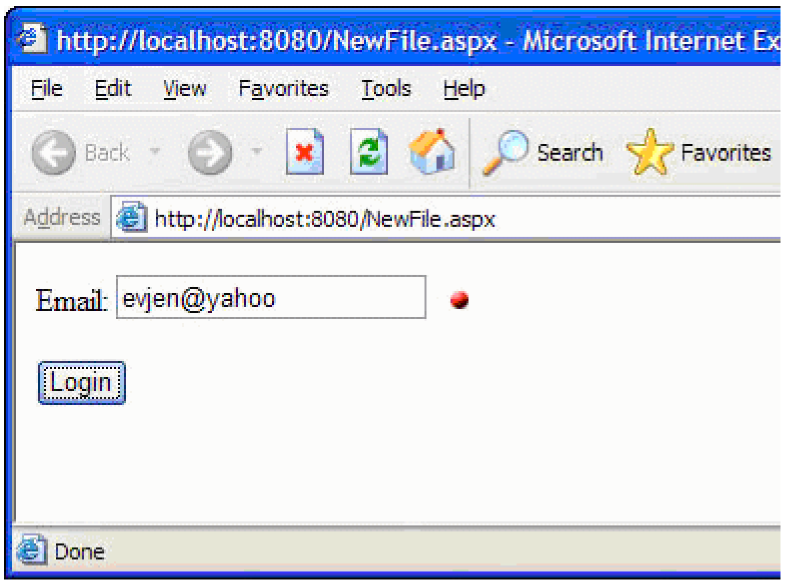

http://localhost:8080/NewFile.aspx - Microsoft Internet Ex
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/NewFile.aspx
Email: evjen@yahoo
Login
Done

Figure 7: An image is displayed when the incorrect e-mail address is entered.

Validation Expressions

Field	Expression	Format Samples	Description
Name	^[a-zA-Z''-'\s]{1,40}$	John Doe O'Dell	Validates a name. Allows up to 40 uppercase and lowercase characters and a

			few special characters that are common to some names. You can modify this list.
Phone Number	^[01]?[- .]?(\([2-9]\d{2}\)|[2-9]\d{2})[- .]?\d{3}[- .]?\d{4}$	(425) 555-0123 425-555-0123 425 555 0123 1-425-555-0123	Validates a U.S. phone number. It must consist of 3 numeric characters, optionally enclosed in parentheses, followed by a set of 3 numeric characters and then a set of 4 numeric characters.
E-mail	^(?("")("".+?""@)|(([0-9a-zA-Z]((\.(?!\.))|[-!#\$%&'*\+/=\?\^`\{\}\\|~\w])*)(?<=[0-9a-zA-Z])@))(?(\[)(\[(\d{1,3}\.){3}\d{1,3}\])|(([0-9a-zA-Z][-\w]*[0-9a-zA-Z]\.)+[a-zA-Z]{2,6}))$	someone@example.com	Validates an e-mail address.
<asp:RegularExpressionValidator ID="RegularExpressionValidator2" runat="server" ErrorMessage="Please Enter Valid Email ID" ValidationGroup="vgSubmit" ControlToValidate="txtEmail" CssClass="requiredFieldValidateStyle"ForeColor="Red" ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"> </asp:RegularExpressionValidator>			
URL	^(ht|f)tp(s?)\:\/\/[0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])*(:(0-9)*)*(\/?)([a-zA-Z0-9\-\.\?\,\'\/\\\+&amp;%\$#_]*)?$	http://www.microsoft.com	Validates a URL

Chapter 5

FILE UPLOAD CONTROL

Overview

- The upload process is quite simple. There are always two parts, the client and server sides, that communicate with each other through HTTP requests and responses. Let's consider the usual upload scenario:
  - A user visits a web page and chooses files to upload.
  - The client application packs these files to a POST request and sends it to the server.
  - The server parses the request, handles it (for example, saves files on a hard disk), and sends a response to the client side.

How to Use FileUpload Control

- FileUpload allows a user to choose a file to be uploaded via the **Browse** button. The control doesn't automatically save a selected file to the server, but it exposes the SaveAs method to perform this. Deploying FileUpload in your web application is very easy.

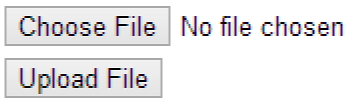
Example of File Upload

- To add a file upload functionality in your website just embed the FileUploadcontrol to the <form> tag in the place where you want users to display the upload interface. The code may look as follows:

```
<html>

  <head>

    <title>Upload Files</title>

  </head>
```

```
<body>

  <form id="form1" runat="server">

    <asp:FileUpload ID="FileUpload1" runat="server" />

    <br/>

    <asp:Button ID="UploadButton" runat="server"

          OnClick="UploadButton_Click"

          Text="Upload File" />

    <br/>

    <asp:Label ID="FileUploadedLabel" runat="server" />

  </form>

</body>

</html>
```

- After running this page, you will see the following interface:



- Let's highlight the difficult parts:
- The FileUpload control (like any other server control) needs to be included in the <form>tag.
- The form should have the runat="server" attribute, which indicates that the form is processed on the server and the FileUpload control can be accessed by server scripts.
- The form should also contain id, name, and method attributes. Typically they are automatically generated by ASP.NET.
- The onClick attribute of the **Upload File** button specifies the event handler that processes file upload.
- After a user clicks the **Upload File** button, the form data will be sent to the server. The code of the **Upload File** button click handler should look like this:

```
protected void UploadButton_Click(object sender, EventArgs e)

{

    if (FileUpload1.HasFile)

        try

        {

            FileUpload1.SaveAs(Server.MapPath("~/uploads/") +

                 FileUpload1.FileName);

            FileUploadedLabel.Text = "File name: " +

                 FileUpload1.PostedFile.FileName + "<br>" +

                 FileUpload1.PostedFile.ContentLength + " kb<br>" +

                 "Content type: " + FileUpload1.PostedFile.ContentType;

        }

        catch (Exception ex)

        {

            FileUploadedLabel.Text = "ERROR: " + ex.Message.ToString();

        }

    else

    {

        FileUploadedLabel.Text = "You have not specified a file.";

    }

}
```

- This event handler checks if any file has been specified, tries to save it to the **uploads** folder, and displays a message indicating whether the file has been saved successfully.

Note: - The FileUpload1 name is similar to the FileUpload id attribute in the client form discussed above.

Chapter 6

GRIDVIEW CONTROL

- GridView control is used to display whole table data on web page. In GridView control each column define a filed or title, while each rows represents a record or data.
- The GridView control display data with rows and columns wise, we can display whole table in GridView and we also display only required columns from table in GridView control in asp.net.
- In GridView control we can easily do sorting, paging and inline editing.
- Also we can perform editing and deleting operation on displayed data with GridView control.

GridView Control Syntax:

```
<asp:GridView ID="GridView1" runat="server">
</asp:GridView>
```

GridView Example:

Step 1 – Open the Visual Studio –> Create a new empty Web application.

Step 2 – Create a New web page.

Step 3 – Drag and drop GridView Control on web page from toolbox.

Step 4 – Create New Database in SQL Server

Step 5 – Make connection between Database and web application

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data.SqlClient;
using System.Data;

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

    }
    protected void btnview_Click(object sender, EventArgs e)
    {
        SqlConnection SQLConn = new SqlConnection("Data Source=.\\SQLEXPRESS;Initial
Catalog='Blog';Integrated Security=True");
        SqlDataAdapter SQLAdapter = new SqlDataAdapter("Select * from UserMst", SQLConn);
        DataTable DT = new DataTable();

        SQLAdapter.Fill(DT);

        GridView1.DataSource = DT;
        GridView1.DataBind();
    }
```

Chapter 7

REPEATER CONTROL

- A Repeater is a Data-bound control. Data-bound controls are container controls. It creates a link between the Data Source and the presentation UI to display the data. The repeater control is used to display a repeated list of items.
- A Repeater has five inline templates to format it:
    - <HeaderTemplate>
    - <AlternatingItemTemplate>
    - <Itemtemplate>
    - <SeperatorTemplate>
    - <FooterTemplate>

<HeaderTemplate>

- Displays Header text for a Data Source collection and applies a different style for the Header text.

<AlternatingItemTemplate>

- Changes the background color or style of alternating items in a Data Source collection.

<Itemtemplate>

- It defines how the each item is rendered from the Data Source collection.

<SeperatorTemplate>

- It will determine the separator element that separates each item in the item collection. It will be a <br> or <Hr> HTML element.

<FooterTemplate>

- Displays a footer element for the Data Source collection.
- Now, in this article I am describing the Repeater control in ASP.NET and how to create a comment page in ASP.NET .
- First I created a database "EmpDetail". Then I created a table in the database.

Table Query

```
CREATE TABLE [dbo].[Comment](
      [UserName] [nvarchar](50) NULL,
      [Subject] [nvarchar](max) NULL,
      [CommentOn] [nvarchar](max) NULL,
      [Post_Date] [datetime] NULL
) ON [PRIMARY]
```

Complete Program

- Repeter_Control_Example.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;

public partial class Repeter_Control_Example : System.Web.UI.Page
{

    SqlCommand cmd;
    SqlDataAdapter da;
    DataSet ds;

    SqlConnection con = new SqlConnection("Data Source=.;Initial Catalog=EmpDetail;Persist Security
Info=True;User ID=sa;Password=****");

    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            RepeterData();
        }
    }
    protected void btnSubmit_click(object sender, EventArgs e)
    {
        try
        {
            con.Open();
             cmd = new SqlCommand("insert into Comment (UserName,Subject,CommentOn,Post_Date)
values(@userName,@subject,@comment,@date)", con);
            cmd.Parameters.Add("@userName", SqlDbType.NVarChar).Value = txtName.Text.ToString();
            cmd.Parameters.Add("@subject", SqlDbType.NVarChar).Value = txtSubject.Text.ToString();
            cmd.Parameters.Add("@comment", SqlDbType.NVarChar).Value = txtComment.Text.ToString();
            cmd.Parameters.Add("@date", SqlDbType.DateTime).Value = DateTime.Now.Date;
            cmd.ExecuteNonQuery();
            con.Close();
            txtName.Text = string.Empty;
            txtSubject.Text = string.Empty;
            txtComment.Text = string.Empty;
            RepeterData();

        }
        catch(Exception ex)
          {
            txtComment.Text= ex.Message;
          }


    }
    public void RepeterData()
    {
        con.Open();
        cmd = new SqlCommand("Select * from Comment Order By Post_Date desc", con);
        DataSet ds = new DataSet();
        da = new SqlDataAdapter(cmd);
        da.Fill(ds);
        RepterDetails.DataSource = ds;
        RepterDetails.DataBind();


    }
}
```

Repeter_Control_Example.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Repeter_Control_Example.aspx.cs"
Inherits="Repeter_Control_Example" %>


<!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 id="Head1" runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
```

```
   <h3>Repeter Control in ASP.NET</h3>
   <div">
   <table>
   <tr>
   <td>Enter Name:</td>
   <td><asp:TextBox ID="txtName" runat="server"></asp:TextBox></td>
   </tr>
    <tr>
   <td>Enter Subject:</td>
   <td><asp:TextBox ID="txtSubject" runat="server"></asp:TextBox></td>
   </tr>
    <tr>
   <td valign="top">Enter Comments:</td>
   <td><asp:TextBox ID="txtComment" runat="server" Rows="5" Columns="20"
TextMode="MultiLine"></asp:TextBox></td>
   </tr>
    <tr>
   <td></td>
   <td><asp:Button ID="btnSubmit" runat="server" Text="Summit" OnClick="btnSubmit_click"
/></td>
   </tr>
   </table>
   </div>
   <div>
   <asp:Repeater ID="RepterDetails" runat="server">
   <HeaderTemplate>
   <table style="border:1px solid #0000FF; width:500px" cellpadding="0">
   <tr style="background-color:#FF6600; color:#000000; font-size: large; font-weight: bold;">
   <td colspan="2">
   <b>Comments</b>
   </td>
   </tr>
```

```
    </HeaderTemplate>
    <ItemTemplate>
    <tr style="background-color:#EBEFF0">
    <td>
    <table style="background-color:#EBEFF0;border-top:1px dotted #df5015; width:500px" >
    <tr>
    <td >
    Subject:
    <asp:Label ID="lblSubject" runat="server" Text='<%#Eval("Subject") %>' Font-Bold="true"/>
    </td>
    </tr>
    </table>
    </td>
    </tr>
    <tr>
    <td>
    <asp:Label ID="lblComment" runat="server" Text='<%#Eval("CommentOn") %>'/>
    </td>
    </tr>
    <tr>
    <td>
    <table style="background-color:#EBEFF0;border-top:1px dotted #df5015;border-bottom:1px solid
#df5015; width:500px" >
    <tr>
    <td >Post By: <asp:Label ID="lblUser" runat="server" Font-Bold="true" Text='<%#Eval("UserName")
%>'/></td>
    <td >Created Date:<asp:Label ID="lblDate" runat="server" Font-Bold="true"
Text='<%#Eval("Post_Date") %>'/></td>
    </tr>
    </table>
    </td>
    </tr>
    <tr>
    <td colspan="2">&nbsp;</td>
    </tr>
    </ItemTemplate>
    <FooterTemplate>
    </table>
    </FooterTemplate>
    </asp:Repeater>
    </div>
    </form>
</body>
</html>
```

Chapter 8

ASP.NET USER CONTROL

- User controls are reusable controls that can be defined once and used whenever we need them, in any of the .aspx pages of our application. We do have skins,themes and css to give a standard look to our website. User controls help to achieve that as well because since we define them once and use many times, thus making sure we do not have same controls on pages looking differently. When we make changes to a user control all these changes will be reflected to all instances of the control.
- We will have to create a simple user control step by step. Imagine you have many pages in your asp.net website. Let’s say that you are developing a fully functional e-commerce site. You will find that you need to collect user data, e.g address information (shipping address info, billing address info, registration customer address info). In this example we will use a user control to collect customer address data.

1) Launch Visual Studio and create a new asp.net website

2) Save this website in you local file system and give it a name. Choose C# as your development language for this website

3) Add new folder to your website and name it UserControl

4) Right-click on this folder and add a new item. From all the items available select a Web User control. Also choose C# as the devlopment language and tick the option Place code in a seperate file.Name the control Address.ascx

5) Switch to the Source view of the Address.ascx control.Have a look at the first line.

```
<%@ Control Language=”C#” AutoEventWireup=”true” CodeFile=”Address.ascx.cs”
Inherits=”UserControls_Items_Address” %>
```

Notice that even though this directive looks a lot like a page directive it starts with @ Control

6) Open the Address.ascx.cs file.Your newly create class does not inherit from theSystem.Web.UI.Page but from

```
public partial class UserControl_Address : System.Web.UI.UserControl
```

The classes System.Web.UI.Page and System.Web.UI.UserControl have lots in common since they both inherit from another class,TemplateControl class

For more information click here

7) Let’s work on our new user control. Insert a table in your “Address.ascx” page. This table should have 4 rows and 2 columns.

8) Add 4 label web server controls on the first 4 rows of the first column. Set their IDproperty as you like(AddressLabel1,AddressLabel2,AddressLabel3,PostCodeLabel). Set their text property like this

- Address1
- Address2
- Address3
- PostCode

9) Add 4 textbox web server controls on the 4 rows of the second column.Set their IDproperty like this

txtaddr1, txtaddr2, txtaddr3, txtpostcode

10) In order to add this newly created web user control to the Default.aspx page just drag and drop it from the Solution Explorer onto the .aspx page

11) Look in the source view of the Default.aspx page and notice this line

```
<%@ Register src=”UserControl/Address.ascx” tagname=”Address” tagprefix=”uc1” %>
```

This is how the user control is registered with the .aspx page. You will also see this

```
<uc1:Address  ID=”Address1” runat=”server” />
```

Change the ID property to Shipping_Address

12) We can add new server controls to our user control. If we wanted to have a label control as a header we must select the Address.ascx and just above the table to insert a new label.Name this label headingLabel.

One can set the Text property of this new header label control and make it apparent to the end use that we talk about Shipping Address. But as I mentioned before we need to use this user control in many places in our website. So we do not want to have a fixed Textproperty but one we can set its value accordingly.

Select the Default.aspx page and in the

```
<uc1:Address ID=”Shipping_Address”  runat=”server”/>
```

section you will see thatheadingLabel is not exposed as a property. It cannot be accessed from our page.

So we must add a new public property on this UserControl_Address class.This is the same with every other normal class.

So in the Address.ascx.cs file type

```
public string Header

{

set { headingLabel.Text = value; }

}
```

13) Now you can go back to the Default.aspx page (Source View) and add theHeader=Shipping Address

```
<uc1:Address ID="Shipping_Address"  runat="server" Header="Shipping Address">
```

14) It is very easy to handle events in a user control.Add a button to the user control. This means that you go to the Address.ascx file and drop a button under the table. Set the IDproperty to be txtNext and the Text property to be Next. Add another .aspx page to your website and call it Checkout.aspx. Double click on the button and you have the empty event handling routine.

Type the following

```
protected void Button1_Click(object sender, EventArgs e)

{

Response.Redirect("~/Checkout.aspx");

}
```

You see how easy it is to handle events in a user control.

14) Let's see now, how we can create an event in the user control. First we need to define an event.

When we define an event we must define the signature of the event handler method. We do that by choosing a delegate type.We also must give a name to our event. We will call itConfirmed event.

Select the Address.ascx.cs and type

public event EndEventHandler Confirmed

So we know now that our user control class will raise an event and we must write some code to actually cause this event to be raised.

So in our Button_Click event handler, we type

if (Confirmed!=null) Confirmed(this, new EventArgs());

When the user clicks the button, the Confirmed event will be raised.

Select the Default.aspx page and in the Source View locate the

```
<uc1:Address ID=”Shipping_Address”  runat=”server” Header=”Shipping Address”/>
```

and change it to

```
<uc1:Address ID=”Shipping_Address” onConfirmed=”ShippingAdress_Confirmed”  runat=”server”
Header=”Shipping Address”/>
```

We just added the event handler in the user control.We do that by entering the name of the event and the name of the event handling routine we will add shortly.

In order to add the ShippingAdress_Confirmed event handling routine we go back to theDefault.aspx.cs file and type

```
protected void ShippingAdress_Confirmed(object sender, EventArgs e)
{
}
```

Inside this routine add this line of code:

```
Response.Write(“The event has been handled”);
```

Run the application and you will see the event raised and handled (the text “The event has been handled” will be printed in the Default.aspx page)

15) Let’s assume that we need 2 instances of our user contol in the Default.aspx page (e.g one for shipping address and one for billing address).Select the Default.aspx and add a new user control. Go to the Source View and add a Header property e.g Billing Address. Also set the ID property to “Billing_Address“. You should have something like this

```
<uc1:Address ID=”Billing_Address” runat=”server” Header=”Billing Address” />.
```

Run the application and see the 2 user controls.

We need to copy all the fields entered in our Shipping address to the Billing address fields.

So we must access the text properties of the Shipping_Address user control and copy them to the Billing_Address user control.

You will see that it is impossible to access these textbox values directly fromShippingAdress_Confirmed event handler in the Default.aspx.cs class, because these text properties are protected and thus invisible to the event handling routine.

So we need to create some public properties in our user control class.

Select the Address.asx.cs file and type

```
public string Address1
{
get {return txtaddr1.Text; }
set {txtaddr1.Text=value;}
}
public string Address2
{
get { return txtaddr2.Text; }
set { txtaddr2.Text = value; }
}
public string Address3
{
get { return txtaddr3.Text; }
set { txtaddr3.Text = value; }
}
public string PostCode
{
get { return txtpostcode.Text; }
set { txtpostcode.Text = value; }
}
```

16) Select the Default.aspx.cs file and inside the ShippingAddress_Confirmed event handling routine, comment out the (Response.Write(“The event has been handled”);) and type

```
Billing_Address.Address1 = Shipping_Address.Address1;
Billing_Address.Address2 = Shipping_Address.Address2;
Billing_Address.Address3 = Shipping_Address.Address3;

Billing_Address.PostCode = Shipping_Address.PostCode;
```

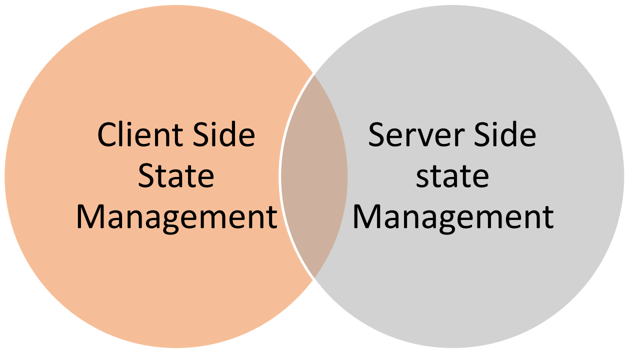
Run your application. Type some address data into the first user control and then click the button. You will see the entered data copied in the second user control.

Chapter 9

STATE MANAGEMENT

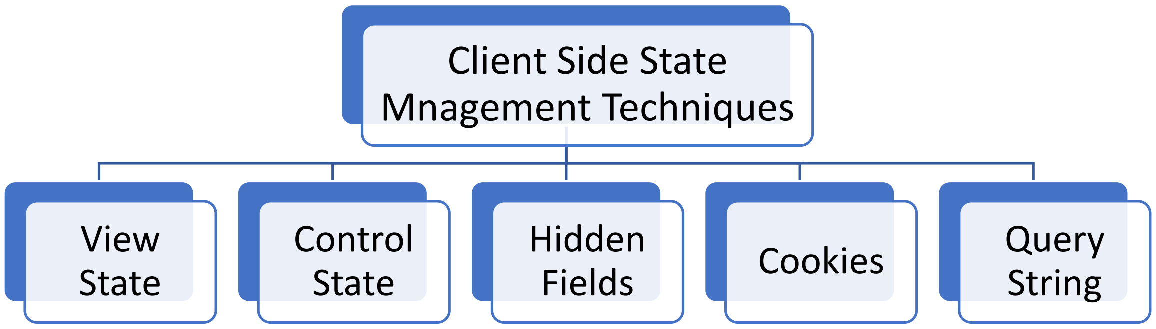
State Management

- HTTP is a stateless Protocol.
- A client open a connection and request some information. The server respond with a requested resources if available. After, closing the connection, the server does not remember any information about the client. So, server considers the next request from the same client as a fresh request, with no relation to the previous request.
- State management means to preserve state of a control, web page, object/data, and user in the application explicitly because all ASP.NET web applications are stateless, i.e., by default, for each page posted to the server, the state of controls is lost.
- Types of State Management:-



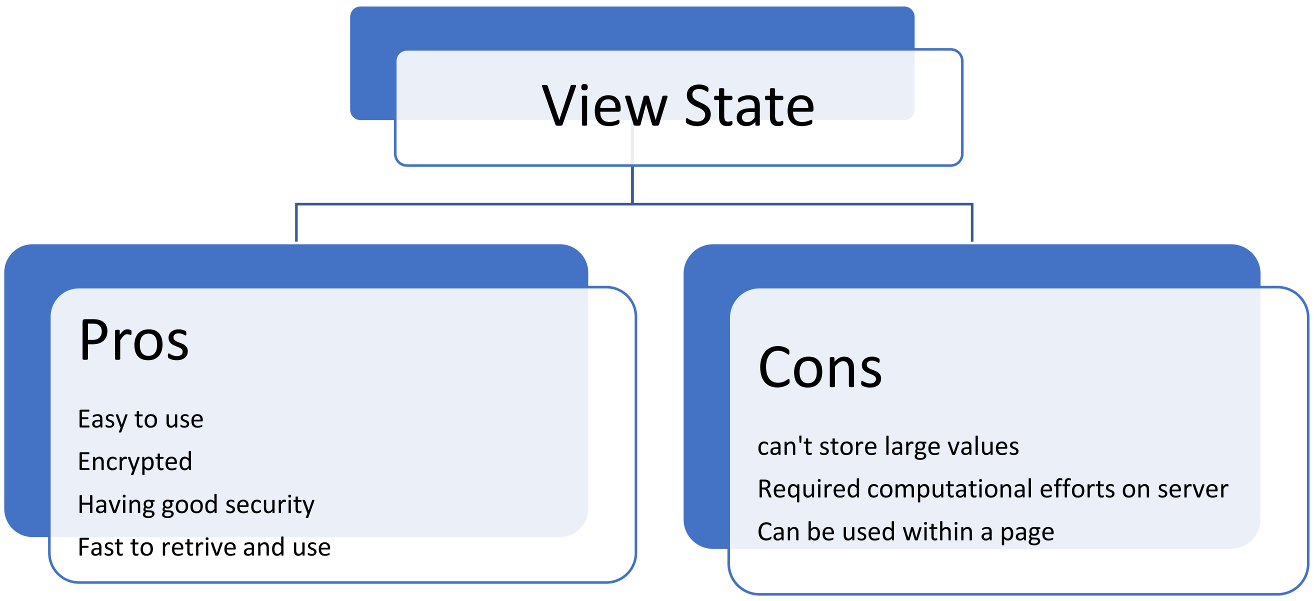
Client Side State Management

- In Client-Side State Management, the state related information will directly get stored on the client-side.



View State

- ViewState is an approach to saving data for the user.
- Because of the stateless nature of web pages, regular page member variables will not maintain their values across postbacks. ViewState is the mechanism that allows state values to be preserved across page postbacks and by default, EnableViewState property will be set to true.
- Stores values per control by key name, like a Hashtable.
- Tracks changes to a View State value's initial state.
- Serializes and Deserializes saved data into a hidden form field on the client.
- Automatically restores View State data on postbacks.



Sample:

```
//To Save Information in View State
ViewState.Add ("NickName", "Dolly");

//Retrieving View state
String strNickName = ViewState ["NickName"];
```

Here is a simple example of using the "ViewState" property to carry values between postbacks.

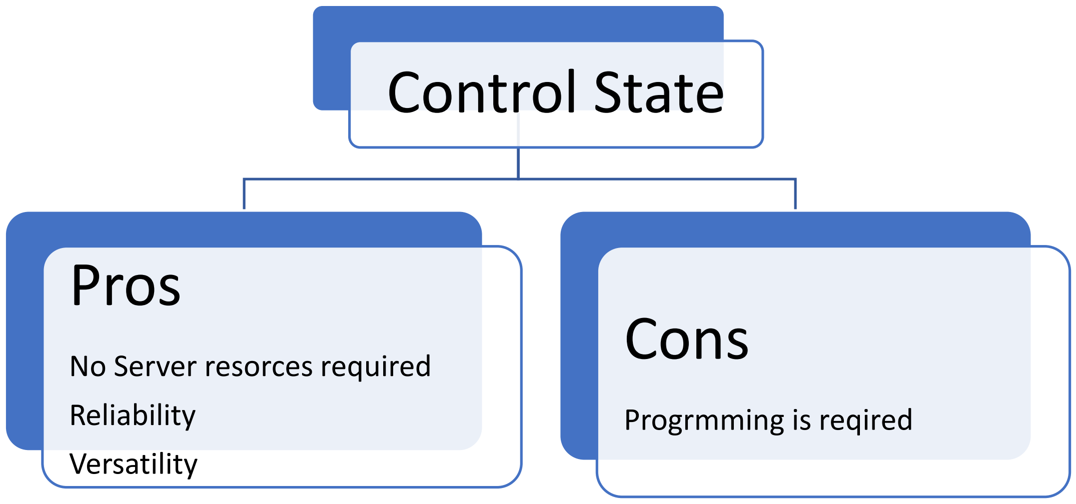
Code Example:

```
publicintSomeInteger {
  get {
    object o = ViewState["SomeInteger"];
    if (o != null) return (int)o;
    return 0;
    //a default
  }
  set { ViewState["SomeInteger"] = value;
```

Control State

- The purpose of the control state repository is to cache data necessary for a control to properly function. ControlState is essentially a private ViewState for your control only, and it is not affected when ViewState is turned off. ControlState is used to store small amounts of critical information. Heavy usage of ControlState can impact the performance of application because it involves serialization and deserialization for its functioning.

  There are two methods you have to implement in your custom control.
- Load Control State
- Save Control State

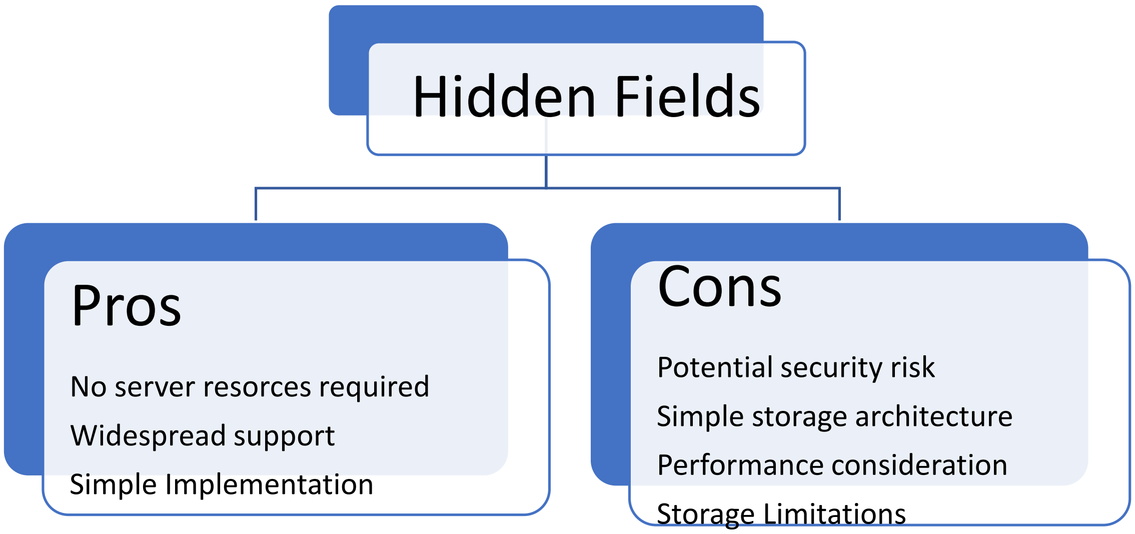


Code Example:

```
publicclassControlStateWebControl : Control
{
   #region Members

   privatestring _strStateToSave;

   #endregion
   #region Methods

   protectedoverridevoidOnInit(EventArgs e)
   {
      Page.RegisterRequiresControlState(this);
      base.OnInit(e);
   }

   protectedoverrideobjectSaveControlState()
   {
      return _strStateToSave;
   }

   protectedoverridevoidLoadControlState(object state)
   {
      if (state != null)
      {
         _strStateToSave = state.ToString();
      }
   }


   #endregion
}
```

Hidden Fields

- A Hidden control is the control which does not render anything on the web page at client browser but can be used to store some information on the web page which can be used on the page. Hidden fields are used to store data at the page level. These fields are not rendered by the browser, rather it's just like a standard control for which you can set its properties. If you use hidden fields, it is best to store only small amounts of frequently changed data on the client.



Sample:

```
//Declaring a hidden variable
protectedHtmlInputHiddenhidNickName;

//Populating hidden variable
hidNickName.Value = "Page No 1";

//Retrieving value stored in hidden field.
stringstr = hidNickName.Value;
```

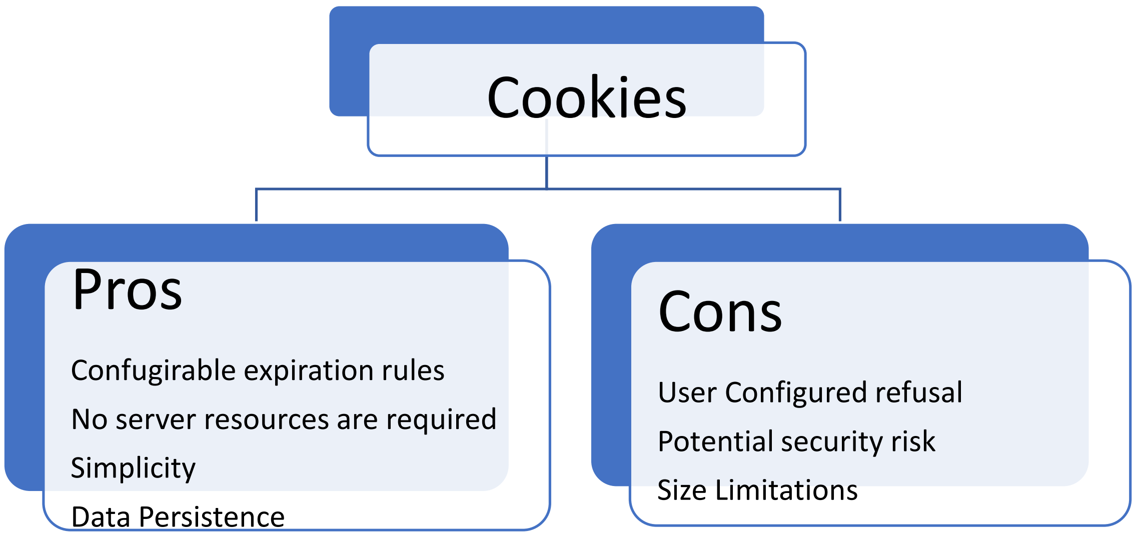
Code Example:

```
protectedvoidPage_Load(object sender, EventArgs e)
{
   if(!IsPostBack)|
      Label1.Text = string.Format("Clicked {0} times", HiddenField1.Value);
}
```

```
protectedvoid Button1_Click(object sender, EventArgs e)
{
    HiddenField1.Value = (Convert.ToInt32(HiddenField1.Value) + 1).ToString();

    Label1.Text = string.Format("Clicked {0} times", HiddenField1.Value);
}
```

Cookies

- A cookie is a small amount of data which is either stored at client side in text file or in memory of the client browser session.
- Cookies are always sent with the request to the web server and information can be retrieved from the cookies at the web server. Every time a user visits a website, cookies are retrieved from the user machine and help identify the user.
- Cookies are useful for storing small amounts of frequently changed information on the client. The information is sent with the request to the server.



Sample:

```
 // Creating a cookie
myCookie.Values.Add("muffin", "chocolate");
myCookie.Values.Add("babka", "cinnamon");
```

```
// Adding Cookie to Collection
Response.Cookies.Add(myCookie);
// Getting Values stored in a cookie
Response.Write(myCookie["babka"].ToString());
// Setting cookie path
myCookie.Path = "/forums";
// Setting domain for a cookie
myCookie.Domain = "forums.geekpedia.com";
// Deleting a cookie
myCookie.Expires = DateTime.Now.AddDays(-1);
```

Code Example:

```
//Storing value in cookie
HttpCookie cookie = new HttpCookie("NickName");
cookie.Value = "David";
Response.Cookies.Add(cookie);
Response.Redirect("Page2.aspx");
//Retrieving value in cookie
if (Request.Cookies.Count> 0 && Request.Cookies["NickName"] != null)
        lblNickName.Text = "Welcome" + Request.Cookies["NickName"].Value.ToString();
else
        lblNickName.Text = "Welcome Guest";
```

Using Cookies Creating/riting Cookies

- There are many ways to create cookies, I am going to outline some of them below:

Way 1 (by using HttpCookies class)

```
HttpCookie StudentCookies = new HttpCookie("StudentCookies");
StudentCookies.Value = TextBox1.Text;
StudentCookies.Expires = DateTime.Now.AddHours(1);
Response.Cookies.Add(StudentCookies);
```

Way 2 (by using Response directly)

```
Response.Cookies["StudentCookies"].Value = TextBox1.Text;
Response.Cookies["StudentCookies"].Expires = DateTime.Now.AddDays(1);
```

Way 3 (multiple values in same cookie)

```
Response.Cookies["StudentCookies"]["RollNumber"] = TextBox1.Text;
Response.Cookies["StudentCookies"]["FirstName"] = "Abhimanyu";
Response.Cookies["StudentCookies"]["MiddleName"] = "Kumar";
Response.Cookies["StudentCookies"]["LastName"] = "Vatsa";
Response.Cookies["StudentCookies"]["TotalMarks"] = "499";
Response.Cookies["StudentCookies"].Expires = DateTime.Now.AddDays(1);
```

Reading/Getting Cookies

- In the above code, I have used many ways to write or create cookies so I need to write here using all the above ways separately.

For Way 1

```
string roll = Request.Cookies["StudentCookies"].Value; //For First Way
```

For Way 2

```
string roll = Request.Cookies["StudentCookies"].Value;  //For Second Way
```

For Way 3

```
string roll;
roll = Request.Cookies["StudentCookies"]["RollNumber"];
roll = roll + " " + Request.Cookies["StudentCookies"]["FirstName"];
roll = roll + " " + Request.Cookies["StudentCookies"]["MiddleName"];
roll = roll + " " + Request.Cookies["StudentCookies"]["LastName"];
roll = roll + " " + Request.Cookies["StudentCookies"]["TotalMarks"];
Label1.Text = roll;
```

Deleting Cookies

- In the above code, I have used many ways to create or read cookies. Now look at the code given below which will delete cookies.

```
if (Request.Cookies["StudentCookies"] != null)
{
   Response.Cookies["StudentCookies"].Expires = DateTime.Now.AddDays(-1);
   Response.Redirect("Result.aspx");  //to refresh the page
}
```

Understanding HttpCookie Class It contains a collection of all cookie values.

- We do not need to use any extra namespaces for HttpCookiesclass (we already have used this in Way 1 above), because this class is derived from System.Webnamespaces. HttpCookiesclass lets us work with cookies without using Response and Request objects (we have already used this in Way 2 and Way 3 above).
- HttpCookie StudentCookies = new HttpCookie("StudentCookies");
- StudentCookies.Domain
- HttpCookieclass has a list of some properties, let us outline them.

```
Domain: It contains the domain of the cookie.
Expires: It contains the expiration time of the cookie.
HasKeys: It contains True if the cookie has subkeys.
Name: It contains the name of the cookie.
Path: It contains the virtual path to submit with the cookie.
Secure: It contains True if the cookie is to be passed in a secure connection only.
Value: It contains the value of the cookie.
```

Limitations of Cookies

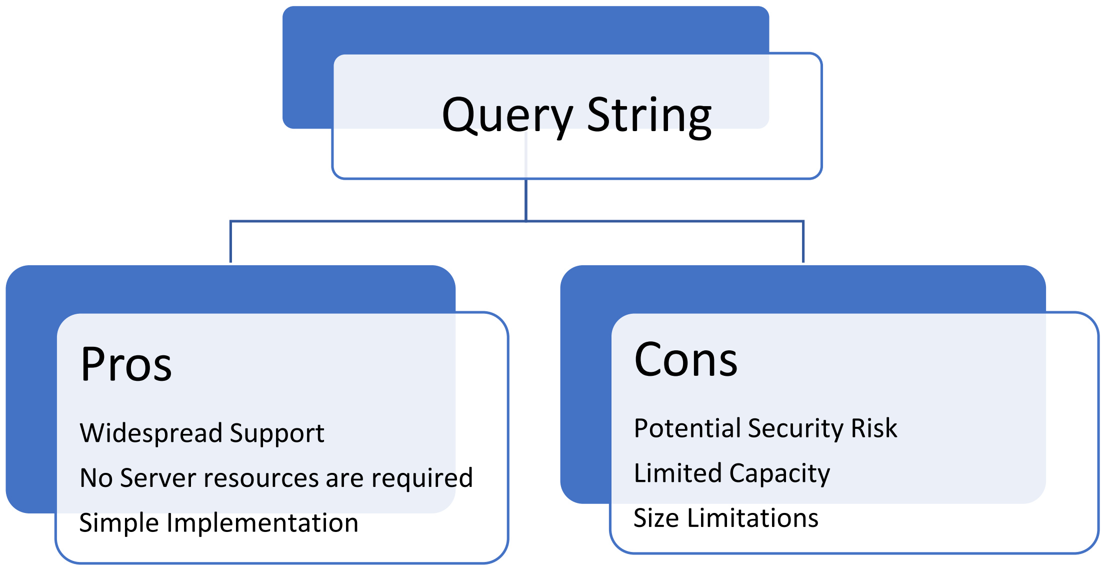
- There are following limitations for cookies:
    - Size of cookies is limited to 4096 bytes.
    - Total 20 cookies can be used on a single website; if you exceed this browser will delete older cookies.
    - End user can stop accepting cookies by browsers, so it is recommended to check the users' state and prompt the user to enable cookies.
- Sometimes, the end user disables the cookies on browser and sometimes browser has no such feature to accept cookies. In such cases, you need to check the users' browser at the home page of website and display the appropriate message or redirect on appropriate page having such message to enable it first. The following code will check whether the users' browser supports cookies or not. It will also detect if it is disabled too.

```
protected void Page_Load(object sender, EventArgs e)
{
   if (Request.Browser.Cookies)
   {
      //supports the cookies
   }
   Else
   {
      //not supports the cookies
      //redirect user on specific page
      //for this or show messages
   }
}
```

Note*: It is always recommended not to store sensitive information in cookies.*

Query String:

- A Query string is used to pass the values or information form one page to another page. They are passed along with URL in clear text. Query strings provide a simple but limited way of maintaining some state information. When surfing the internet you should have seen weird internet addresses such as:
- http://www.localhost.com/Webform2.aspx?name=ABC&lastName=XYZ
- This HTML address uses a QueryString property to pass values between pages.



Syntax:

```
Request.QueryString(variable)[(index)|.Count]
```

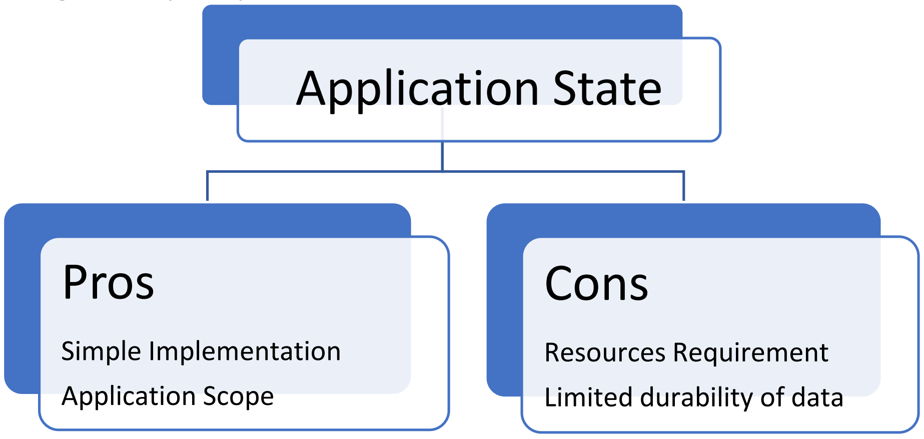
Code Example:

```
using System;
usingSystem.Web.UI;

publicpartialclass_Default : Page
{
    protectedvoidPage_Load(object sender, EventArgs e)
    {
        string v = Request.QueryString["param"];
        if (v != null)
        {
            Response.Write("param is ");
            Response.Write(v);
        }
        string x = Request.QueryString["id"];
        if (x != null)
        {
            Response.Write("   id detected");
        }
    }
}
```

SERVER SIDE STATE MANAGEMENT:

Application State:

- Application State is used to store information which is shared among users of the ASP.Net web application. Application state is stored in the memory of the windows process which is processing user requests on the web server. Application state is useful in storing a small amount of often-used data. If application state is used for such data instead of frequent trips to the database, then it increases the response time/performance of the web application.
- In classic ASP, an application object is used to store connection strings. It's a great place to store data that changes infrequently.



Code Example 1:

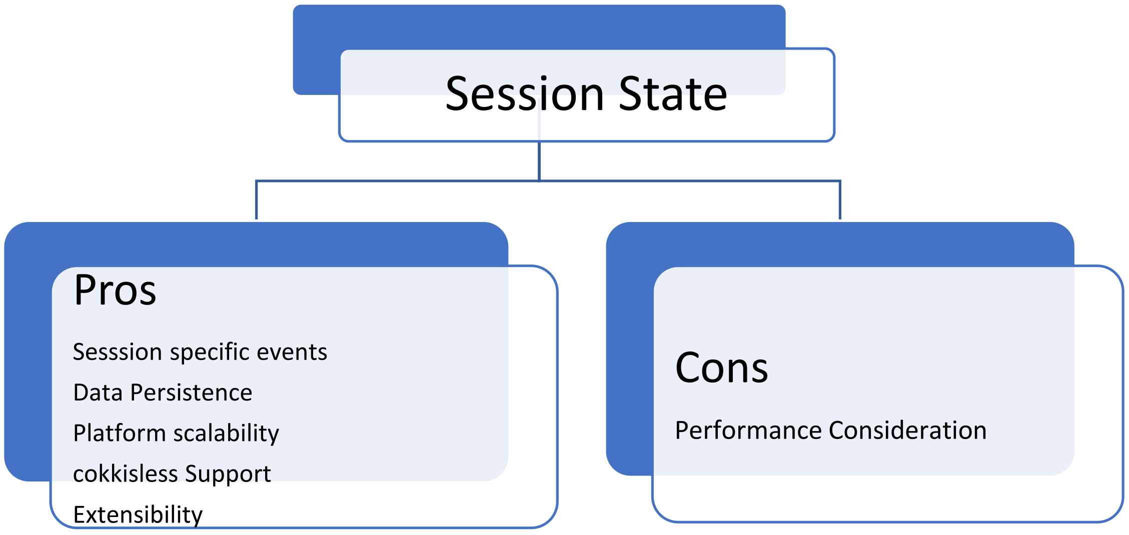
```
//Stroing information in application state

    Application["NickName"] = "Nipun";      stringstr = Application["NickName"].ToString();
```

Code Example 2:

```
 protectedvoidPage_Load(object sender, EventArgs e)
   {
// Code that runs on page load
      Application["LoginID"] = "Nipun";
      Application["DomainName"] = "www.nipun.com";
   }
```

Session State:

- ASP.NET Session state provides a place to store values that will persist across page requests. Values stored in Session are stored on the server and will remain in memory until they are explicitly removed or until the Session expires. It is defined as the period of time that a unique user interacts with a Web application. Session state is a collection of objects, tied to a session stored on a server.



Sample:

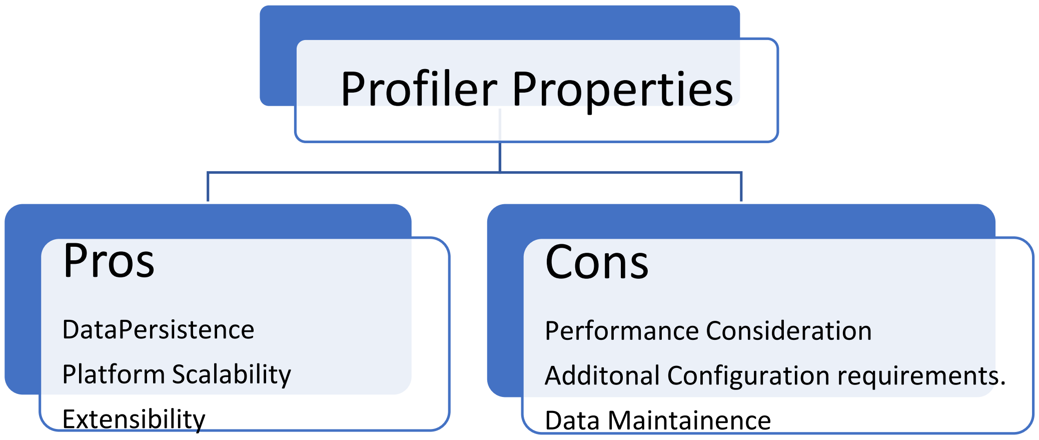
```
//Storing informaton in session state
Session["NickName"] = "ABC";
//Retrieving information from session state
stringstr = Session["NickName"];
```

Code Example:

```
 objectsessionObject = Session["someObject"];
if (sessionObject != null)
{
 myLabel.Text = sessionObject.ToString();
}
```

Profile Properties:

- ASP.NET provides a feature called profile properties, which allows you to store user-specific data. It is similar to session state, except that unlike session state, the profile data is not lost when a user's session expires. The profile properties feature uses an ASP.NET profile, which is stored in a persistent format and associated with an individual user. In this each user has its own profile object.



Sample:

```
<profile>
  <properties>
    <add item="item name" />
  </properties>
</profile>
```

Code Example:

```
<authentication mode="Windows" />
     <profile>
     <properties>
    <add name="FirstName"/>
    <add name="LastName"/>
    <add name="Age"/>
    <add name="City"/>
     </properties>
  </profile>
```

Application Level variable:

Value will persist till the down of web server.

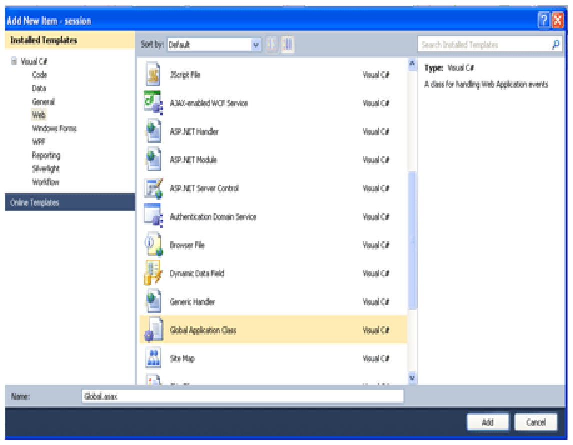
Global.asaxfile:

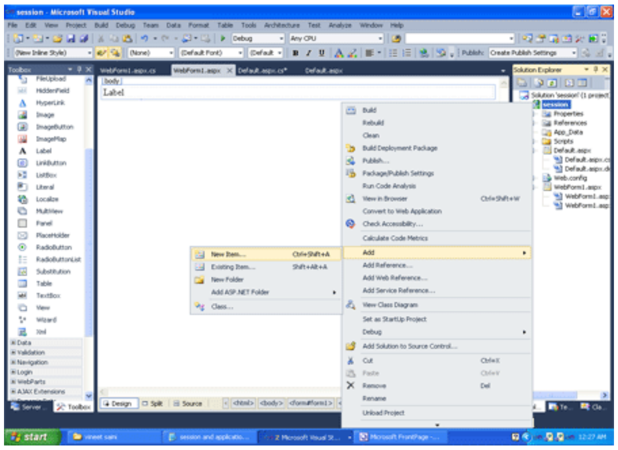
- Global.asax file is ASP.NET application file. Global.asax is extension of Global Application Class. Global.asax file resides in the IIS virtual root of an ASP.NET application. It is used to declare application level variable. There are many events in Global.asax file which are as follows:

Application_Init	•Fires when the application initializes for the first time.
Application_Start	•Fires the first time an application starts.
Session_Start	•Fires the first time when a user's session is started.
Application_BeginRequest	•Fires each time a new request comes in.
Application_EndRequest	•Fires when the request ends.
Application_AuthenticateRequest	•Indicates that a request is ready to be authenticated.
Application_Error	•Fires when an unhandled error occurs within the application.
Session_End	•Fires whenever a single user Session ends or times out.
Application_End	•Fires when the application ends or times out.

Example :

- HIT counter
- Solution Explorer->Right click->add->new item->Global application class (Global.asax)->add likes below image.





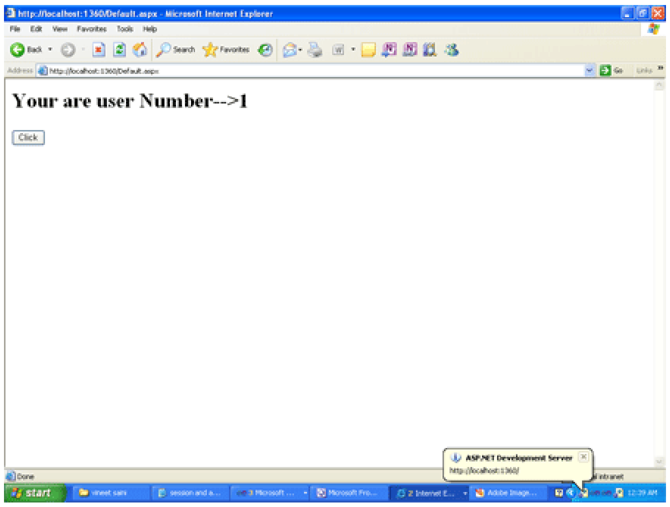
Then coding on Global.asax file.

```
using System;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Web;
usingSystem.Web.Security;
usingSystem.Web.SessionState;
namespace Application
{   publicclassGlobal : System.Web.HttpApplication
    {
        protectedvoidApplication_Start(object sender, EventArgs e)
        {
            Application["hits"] = 0;
        }
        protectedvoidSession_Start(object sender, EventArgs e)
        {
            Application["hits"] = Int32.Parse(Application["hits"].ToString()) + 1;
        }
        protectedvoidApplication_BeginRequest(object sender, EventArgs e)
        {
        }
        protectedvoidApplication_AuthenticateRequest(object sender, EventArgs e)
        {
        }
        protectedvoidApplication_Error(object sender, EventArgs e)
        {
        }
        protectedvoidSession_End(object sender, EventArgs e)
        {
        }
        protectedvoidApplication_End(object sender, EventArgs e)
        {
        }
    }
}
```

Now we will take a button on web form then coding on button click.

```
using System;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Web;
usingSystem.Web.UI;
usingSystem.Web.UI.WebControls;
namespace Application
{
    publicpartialclass_Default : System.Web.UI.Page
    {
        protectedvoidPage_Load(object sender, EventArgs e)
        {
        }
        protectedvoid Button1_Click(object sender, EventArgs e)
        {
            Response.Write("<h1>Your are user Number-->" + Application["hits"].ToString());
        }
    }
}
```

Now run application (Press F5). Then click the button.



```
Your are user Number-->1
```

http://localhost:1360/Default.aspx - Microsoft Internet Explorer
Click
ASP.NET Development Server
http://localhost:1360/
start

Point to remeber

State management is the process by which you maintain state and page information over multiple requests for the same or different pages.

Chapter 10

CACHING

Overview

- The majority [if not all] of the pages in a dynamic website are dynamic. That is, pages that are created on user request. As we all know, dynamic web pages help to provide dynamic content, customized for the user requesting the page [e.g.: the user's home page]. Dynamic pages also help provide dynamic content fetched from a changing data store without the need for the administrator to change the page content every time something changes in the data store [e.g.: Listing of books in a publisher's website]. The disadvantage is the overhead in creating the pages for every user request.
- To overcome this, some websites have page creation engines which create all pages in one go and save them as HTML pages which are then served to the users. But this will only help in scenarios where the page content is the same for all requests [user-independent] as in the second example above. The listing of books is the same irrespective of the user requesting the page. Even if there is provision for listing books category wise by providing different category ID values through the querystring, the page output for a particular category of books is the same for all users.
- ASP.NET provides support for "caching" which will help us solve this problem to a great extend. It can cache [store in memory] the output generated by a page and will serve this cached content for future requests. And this is useful only in the second scenario described earlier, where the page content is the same for all requests [user-independent]. The caching feature is customizable in various ways and we will see how we can do that as we go through this article.

Caching a page

- In order to cache a page's output, we need to specify an @OutputCache directive at the top of the page. The syntax is as shown below:

```
<%@ OutputCache Duration=5 VaryByParam="None" %>
```

- As you can see, there are two attributes to this directive. They are:
- Duration - The time in seconds of how long the output should be cached. After the specified duration has elapsed, the cached output will be removed and page content generated for the next request. That output will again be cached for 10 seconds and the process repeats.
- VaryByParam - This attribute is compulsory and specifies the querystring parameters to vary the cache.
- In the above snippet, we have specified the VaryByParam attribute as None which means the page content to be served is the same regardless of the parameters passed through the querystring [see Example 1 in the sample download].
- If there are two requests to the same page with varying querystring parameters, e.g.: .../PageCachingByParam.aspx?id=12 and .../PageCachingByParam.aspx?id=15] and separate page content is generated for each of them, the directive should be:
- <%@ OutputCache Duration=5 VaryByParam="id" %>
- The page content for the two requests will each be cached for the time specified by the Duration attribute [see Example 2 in the sample download].
- To specify multiple parameters, use semicolon to separate the parameter names. If we specify the VaryByParam attribute as *, the cached content is varied for all parameters passed through the querystring.
- Some pages generate different content for different browsers. In such cases, there is provision to vary the cached output for different browsers. The @OutputCache directive has to be modified to:
- <%@ OutputCache Duration=5 VaryByParam="id" VaryByCustom="browser" %>
- This will vary the cached output not only for the browser but also its major version. I.e., IE5, IE 6, Netscape 4, Netscape 6 will all get different cached versions of the output.

Caching page fragments

- Sometimes we might want to cache just portions of a page. For example, we might have a header for our page which will have the same content for all users. There might be some text/image in the header which might change every day. In that case, we will want to cache this header for duration of a day.
- The solution is to put the header contents into a user control and then specify that the user control content should be cached. This technique is called fragment caching.

- To specify that a user control should be cached, we use the @OutputCache directive just like we used it for the page.
- <%@ OutputCache Duration=10 VaryByParam="None" %>
- With the above directive, the user control content will be cached for the time specified by the Duration attribute [10 secs]. Regardless of the querystring parameters and browser type and/or version, the same cached output is served. [See Example 3 in the download for a demonstration].

Data Caching

- ASP.NET also supports caching of data as objects. We can store objects in memory and use them across various pages in our application. This feature is implemented using the Cache class. This cache has a lifetime equivalent to that of the application. Objects can be stored as name value pairs in the cache. A string value can be inserted into the cache as follows:

```
Cache["name"]="Smitha";
```

- The stored string value can be retrieved like this:

```
if (Cache["name"] != null)
   Label1.Text= Cache["name"].ToString();
```

- To insert objects into the cache, the Add method or different versions of the Insert method of the Cache class can be used. These methods allow us to use the more powerful features provided by the Cache class. One of the overloads of the Insert method is used as follows:

Cache.Insert("Name"	strName
new CacheDependency(Server.MapPath("name.txt")	
DateTime.Now.AddMinutes(2)	TimeSpan.Zero);

- The first two parameters are the key and the object to be inserted. The third parameter is of type CacheDependency and helps us set a dependency of this value to the file named name.txt.

So whenever this file changes, the value in the cache is removed. We can specify null to indicate no dependency. The fourth parameter specifies the time at which the value should be removed from cache. [See example 5 for an illustration.] The last parameter is the sliding expiration parameter which shows the time interval after which the item is to be removed from the cache after its last accessed time.

- The cache automatically removes the least used items from memory, when system memory becomes low. This process is called scavenging. We can specify priority values for items we add to the cache so that some items are given more priority than others:

Cache.Insert("Name"	strName
new CacheDependency(Server.MapPath("name.txt")	
DateTime.Now.AddMinutes(2)	TimeSpan.Zero
CacheItemPriority.High	null);

- The CacheItemPriority enumeration has members to set various priority values. The CacheItemPriority.High assigns a priority level to an item so that the item is least likely to be deleted from the cache.

Points of interest

- *If there are old ASP pages in your website which use the Response.Expires property to cache page output, they can be retained as such. ASP.NET supports this property as well.*
- *The Insert method of the Cache class will overwrite any existing item with the same key name.*
- *The CacheItemPriority.NotRemovable priority value can be used with Cache.Insert method to set the priority level of an item so that the item will not be removed from the cache during scavenging.*

Using the Code

- ASP.NET provides a couple of caching methods: Output Cache (including Page level cache and User Control level cache) and the Cache API. In this article, we will discuss Output Cache. Output

Cache has the advantage of being simple to implement, and is sufficient in most cases. It simply keeps a copy of the response that was sent to the client in memory and subsequent requests are then responded with the cached output until the cache expires, which incredibly improves the ASP.NET web application performance.

- For ASP.NET Output Cache, ASP.NET uses @ OutputCache to declare many attributes to control the output caching policies of the ASP.NET page or a user control contained in a page.

```
<%@ OutputCache Duration="#ofseconds"
  Location="Any | Client | Downstream | Server | None | ServerAndClient "
  Shared="True | False"
  VaryByControl="controlname"
  VaryByCustom="browser | customstring"
  VaryByHeader="headers"
  VaryByParam="parametername"
  VaryByContentEncoding="encodings"
  CacheProfile="cache profile name | ""
  NoStore="true | false"
  SqlDependency="database/table name pair | CommandNotification"
%>
```

- In this article, we will cover @OutputCache's Duration,
- VaryByCustom,
- VaryByParam, and
- VaryByControl attributes to cache the output of a page.
- The Duration attribute
- Add @OutputCache in the ASPX markup and specify the expiration time. In this case, we assign 10s for it. For example: OutputCache Duration="10" VaryByParam="none".
- Run the ASP.NET web application and launch this page, and we will see that the date-time on the page won't change 10s when the page is reloading.

```
<%@ OutputCache Duration="10" VaryByParam="none" %>
<!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>OutPutCacheWithDuration</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <asp:Label ID="lblResult" runat="server"></asp:Label>
      <br />
      <br />
      <asp:Button ID="btnPostBack" runat="server" Text="Post Back"  />
      <p>
         The page will be cached 10s, and then you can
              click Button to update datetime.
      </p>
   </div>
   </form>
</body>
</html>
```

- The VaryByControl attribute
  - Drag and drop a DropDownList in the page and add three items to it.
  - Add @OutputCache in the ASPX markup and specify the expiration time and VaryByControl attribute. For example: OutputCache Duration="1000" VaryByControl="ddlOption".
  - Run the ASP.NET web application and launch this page, we can see that the different items have their corresponding cache.

```
<%@ OutputCache Duration="1000" VaryByControl="ddlOption" %>
<!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>OutPutCacheWithVaryByControl</title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
    <asp:Label ID="lblResult" runat="server"></asp:Label>
    <br />
    <br />
    <asp:DropDownList ID="ddlOption" runat="server" AutoPostBack="True"
          OnSelectedIndexChanged="ddlOption_SelectedIndexChanged">
      <asp:ListItem Selected="True">Option One</asp:ListItem>
      <asp:ListItem>Option Two</asp:ListItem>
      <asp:ListItem>Option Three</asp:ListItem>
    </asp:DropDownList>
    <p>
      The page will be rendered from cache basing
      on the selected item of DropDownList.
      The different item has corresponding cache.
    </p>
  </div>
  </form>
</body>
</html>
```

- The VaryByCustom attribute
    - Add @OutputCache in the ASPX markup and specify the expiration time and VaryByControl attribute with the "browser" value. For example: OutputCache Duration="1000" VaryByCustom="browser"VaryByParam="none".
    - Run the ASP.NET web application and launch this page with IE and Firefox (or a browser with a different name, major version), and we will see that there is different cache versions for different browsers.

```
<%@ OutputCache Duration="1000" VaryByCustom="browser" VaryByParam="none" %>
<!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>OutPutCacheWithVaryByCustom</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="lblResult" runat="server"></asp:Label>
        <br />
        <br />
        <asp:Button ID="btnPostBack" runat="server" Text="Post Back" />
        <p>
            The page will be rendered from cache basing
            on the version of browser, such as IE and FireFox.
        </p>
    </div>
    </form>
</body>
</html>
```

- The VaryByParam attribute
    - Add @OutputCache in the ASPX markup and specify the expiration time and VaryByParam attribute with an "id" value. For example: OutputCache Duration="1000" VaryByParam="id".
    - Run the ASP.NET web application and launch this page, and we can request it using a QueryString "id" with a different value.

```
For example:
~/OutputCacheWithParam.aspx?id=1
~/OutputCacheWithParam.aspx?id=2 >
```

```
<%@ OutputCache Duration="1000" VaryByParam="id" %>
<!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>OutPutCacheWithVaryByParam</title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
    <asp:Label ID="lblResult" runat="server"></asp:Label>
    <p>
      The page will be rendered from cache until
      the value of QueryString named "id" is
      changed or Duration is expiration.
    </p>
  </div>
  </form>
</body>
</html>
```

Using Data Caching

- The .NET data caching API is comprised of the two classes in the System.Web.Caching namespace. The first class, Cache, is the class we'll be using to add and remove items from the data cache. The second class, CacheDependency, is used when assigning a cache dependency to an item in the data cache (we'll be discussing this in due time).
- To add an item to the cache you can simply do:

Note:- The C# version uses brackets instead of parenthesis when referencing the items of the cache in the above manner. In the remainder of the examples in this article I will be using VB.NET syntax, but will point out any major differences between the VB.NET and C# syntax

- The above code adds the item value to the data cache with the key key. The key is used to reference the item at some later point. That is, in another ASP.NET Web page we can extract the value inserted above by using:

```
value = Cache("key")

      - or -

value = Cache.Get("key")
```

- To explicitly remove an item from the data cache you can use the Remove method, specifying the key of the cache item you want removed:

```
Cache.Remove("key")
```

Chapter 11

CONFIGURATION AND DEPLOYMENT

Overview

- Like ASP, ASP.NET encapsulates its entities within a web application. A web application is an abstract term for all the resources available within the confines of an IIS virtual directory. For example, a web application may consist of one or more ASP.NET pages, assemblies, web services configuration files, graphics, and more. In this section we explore two fundamental components of a web application, namely global application files (Global.asax) and configuration files (Web.config).

Global.asax

- Global.asax is a file used to declare application-level events and objects. Global.asax is the ASP.NET extension of the ASP Global.asa file. Code to handle application events (such as the start and end of an application) reside in Global.asax. Such event code cannot reside in the ASP.NET page or web service code itself, since during the start or end of the application, its code has not yet been loaded (or unloaded). Global.asax is also used to declare data that is available across different application requests or across different browser sessions. This process is known as application and session state management.
- The Global.asax file must reside in the IIS virtual root. Remember that a virtual root can be thought of as the container of a web application. Events and state specified in the global file are then applied to all resources housed within the web application. If, for example, Global.asax defines a state application variable, all .aspx files within the virtual root will be able to access the variable.
- Like an ASP.NET page, the Global.asax file is compiled upon the arrival of the first request for any resource in the application. The similarity continues when changes are made to the Global.asax file; ASP.NET automatically notices the changes, recompiles the file, and directs all new requests to the newest compilation. A Global.asax file is automatically created when you create a new web application project in the VS.NET IDE.

Global.asax file in ASP.NET

- The Global.asax, also known as the ASP.NET application file, is located in the root directory of an ASP.NET application. This file contains code that is executed in response to application-level and session-level events raised by ASP.NET or by HTTP modules.
- You can also define 'objects' with application-wide or session-wide scope in the Global.asax file. These events and objects declared in the Global.asax are applied to all resources in that web application.
- Note 1: The Global.asax is an optional file. Use it only when there is a need for it.
- Note 2: If a user requests the Global.asax file, the request is rejected. External users cannot view the file.
- The Global.asax file is parsed and dynamically compiled by ASP.NET. You can deploy this file as an assembly in the \bin directory of an ASP.NET application.

How to create Global.asax

- Adding a Global.asax to your web project is quiet simple.
- Open Visual Studio 2005 or 2008 > Create a new website > Go to the Solution Explorer > Add New Item > Global Application Class > Add.

Examining the methods related to the events in Global.asax

- There are 2 'set' of methods that fire corresponding to the events. The first set which gets invoked on each request and the second set which does not get invoked on each request. Let us explore these methods.

Methods corresponding to events that fire on each request

Application_BeginRequest()	• fired when a request for the web application comes in.
Application_AuthenticateRequest()	• fired just before the user credentials are authenticated. You can specify your own authentication logic over here.
Application_AuthorizeRequest()	• fired on successful authentication of user's credentials. You can use this method to give authorization rights to user.
Application_ResolveRequestCache()	• fired on successful completion of an authorization request.
Application_AcquireRequestState()	• fired just before the session state is retrieved for the current request.
Application_PreRequestHandlerExecute()	• fired before the page framework begins before executing an event handler to handle the request.
Application_PostRequestHandlerExecute()	• fired after HTTP handler has executed the request.
Application_ReleaseRequestState()	• fired before current state data kept in the session collection is serialized.
Application_UpdateRequestCache()	• fired before information is added to output cache of the page.
Application_UpdateRequestCache()	• fired at the end of each request

Methods corresponding to events that do not fire on each request

Application_Start()	• fired when the first resource is requested from the web server and the web application starts.
Session_Start()	• fired when session starts on each new user requesting a page.
Application_Error()	• fired when an error occurs.
Session_End()	• fired when the session of a user ends.
Session_End()	• fired when the web application ends.
Session_End()	• fired when the web application is destroyed.

Show me an example!!

- Let us see an example of how to use the Global.asax to catch unhandled errors that occur at the application level.
- To catch unhandled errors, do the following. Add a Global.asax file (Right click project > Add New Item > Global.asax). In the Application_Error() method, add the following code:

```
void Application_Error(object sender, EventArgs e)
  {
     // Code that runs when an unhandled error occurs
     Exception objErr = Server.GetLastError().GetBaseException();
     string err = "Error in: " + Request.Url.ToString() +
                    ". Error Message:" + objErr.Message.ToString();


  }
```

- Here we make use of the Application_Error() method to capture the error using the Server.GetLastError().

Note: - Global.asax is a file used to declare application-level events and objects. The file is responsible for handling higher-level application events such as Application_Start, Application_End, Session_Start, Session_End, and so on.

Application Directives

- Application directives are placed at the top of the Global.asax file and provide information used to compile the global file. Three application directives are defined, namely Application, Assembly, and Import. Each directive is applied with the following syntax:
- <%@ appDirective appAttribute=Value ...%>

Web.config

- In ASP, configuration settings for an application (such as session state) are stored in the IIS metabase. There are two major disadvantages with this scheme. First, settings are not stored in a human-readable manner but in a proprietary, binary format. Second, the settings are not easily ported from one host machine to another.(It is difficult to transfer information from an IIS�s metabase or Windows Registry to another machine, even if it has the same version of Windows.)
- Web.config solves both of the aforementioned issues by storing configuration information as XML. Unlike Registry or metabase entries, XML documents are human-readable and can be modified with any text editor. Second, XML files are far more portable, involving a simple file transfer to switch machines.
- Unlike Global.asax, Web.config can reside in any directory, which may or may not be a virtual root. The Web.config settings are then applied to all resources accessed within that directory, as well as its subdirectories. One consequence is that an IIS instance may have many web.config files. Attributes are applied in a hierarchical fashion. In other words, the web.config file at the lowest level directory is used.

- Since Web.config is based on XML, it is extensible and flexible for a wide variety of applications. It is important, however, to note that the Web.config file is optional. A default Web.config file, used by all ASP.NET application resources, can be found on the local machine

[Web.config file]

```
<appSettings>
<add key="ConnectionString" value="server=localhost;database=TestDB;uid=sa;password=secret;" />

</appSettings>
using System.Configuration;
public class TestGetConnection
{
  public TestGetConnection()
  {
    try
    {
      // Get connection string from Web.Config
      string strConnection = ConfigurationSettings.AppSettings("ConnectionString");
      }
   }
 }
```

Introduction to web.config

- You will more often use the Web.config file not only for securing your application but also for wide range of other purposes which it is intended for. ASP.NET Web.config file provides you a flexible way to handle all your requirements at the application level. Despite the simplicity provided by the .NET Framework to work with web.config, working with configuration files would definitely be a task until you understand it clearly.
- The contents of the articles are summarized below:
- Web.config sections/settings
- Reading Web.config
- Writing or manipulating Web.config
- Encrypting the Web.config and
- Creating your own Custom Configuration Sections

Points to be Remembered

ASP.NET Web.config allows you to define or revise the configuration settings at the time of developing the application or at the time of deployment or even after deployment. The following are brief points that can be understood about the Web.config file:

- *Web.config files are stored in XML format which makes us easier to work with.*
- *You can have any number of Web.config files for an application. Each Web.config applies settings to its own directory and all the child directories below it.*
- *All the Web.config files inherit the root Web.config file available at the following location systemroot\Microsoft.NET\Framework\versionNumber\CONFIG\Web.config location*
- *IIS is configured in such a way that it prevents the Web.config file access from the browser.*
- *The changes in Web.config don't require the reboot of the web server.*

Web.config Settings

◆ Before we start working with configuration settings of ASP.NET, we see the hierarchy of the Web.config file.

```
<configuration>

        <configSections>
            <sectionGroup>
            </sectionGroup>
        </configSections>

        <system.web>
        </system.web>

        <connectionStrings>
        </connectionStrings>

        <appSettings>
        </appSettings>
        ..........................................................................
        ..........................................................................
        ..........................................................................
        ..........................................................................
        ..........................................................................
```

```
</configuration>
```

- So from the above tree structure, we can understand that the configuration tag is the root element of the Web.config file under which it has all the remaining sub elements. Each element can have any number of attributes and child elements which specify the values or settings for the given particular section. To start with, we'll see the working of some of the most general configuration settings in the Web.config file.

system.web

- In the configuration hierarchy, the most common thing we will work with is the system.web section. Now we look at some of the child sections of the system.web section of Web.config file.

Compilation Settings

- If you are using Visual Studio 2010, probably the only available section of Web.config file by default is Compilation section. If you want to specify the target framework or if you need to add an assembly from the Global Assembly Cache (GAC) or if you want to enable the debugging mode of the application, you can take Compilation settings as granted for these tasks. The following code is used to achieve the discussed settings:

```
<system.web
        <compilation
                  debug="true" strict="true" explicit="true" batch="true"
                  optimizeCompilations="true" batchTimeout="900"
                  maxBatchSize="1000" maxBatchGeneratedFileSize="1000"
                  numRecompilesBeforeAppRestart="15" defaultLanguage="c#"
                  targetFramework="4.0" assemblyPostProcessorType="">
        <assemblies>
                <add assembly="System, Version=1.0.5000.0, Culture=neutral,
                   PublicKeyToken=b77a5c561934e089"/>
        </assemblies>

</compilation>
</system.web>
```

- Under the assemblies element, you are supposed to mention the type, version, culture and public key token of the assembly. In order to get the public key token of an assembly, you need to follow the below mentioned steps:
- Go to Visual Studio tools in the start menu and open the Visual Studio command prompt.
- In the Visual Studio command prompt, change the directory to the location where the assembly or .dll file exists.

- Use the following command, sn –T itextsharp.dll.
- It generates the public key token of the assembly. You should keep one thing in mind that only public key token is generated only for the assemblies which are strongly signed.

Example

```
C:\WINNT\Microsoft.NET\Framework\v3.5> sn -T itextsharp.dll
Microsoft (R) .NET Framework Strong Name Utility Version 3.5.21022.8
Copyright (c) Microsoft Corporation.  All rights reserved.

Public key token is badfaf3274934e0
```

- Explicit and sample attributes are applicable only to VB.NET and C# compiler however ignores these settings.

Page Settings

- Ok, by this time, we have got familiar with the Web.config file and we have seen the settings of Compilation Sections, now we will see the settings of a page. As an ASP.NET application consists of several number of pages, we can set the general settings of a page like sessionstate, viewstate, buffer, etc., as shown below:

```
<pages buffer ="true" styleSheetTheme="" theme ="Acqua"
              masterPageFile ="MasterPage.master"
              enableEventValidation="true">
```

- By using the MasterPageFile and theme attributes, we can specify the master page and theme for the pages in web application.
    - 400 Bad Request
    - 401 Unauthorized
    - 404 Not Found
    - 408 Request Timeout

Location Settings

- If you are working with a major project, probably you might have numerous numbers of folders and sub-folders, at this kind of particular situation, you can have two options to work with. First thing is to have a Web.config file for each and every folder(s) and Sub-folder(s) and the second one is to have a single Web.config for your entire application. If you use the first approach, then you might be in a smoother way, but what if you have a single Web.config and you need to configure the sub-folder or other folder of your application, the right solution is to use the "Location" tag of "system.web" section of Web.config file. However you can use this tag in either of the discussed methods.

- The following code shows you to work with Location settings:

```
<location path="Login.aspx">
        <system.web>
                <authorization>
                 <allow users="*"/>
                </authorization>
        </system.web>
</location>

<location path ="Uploads">
    <system.web>
        <compilation debug = "false">
    </system.web>
</location>
```

Authentication, Authorization, Membership Provider, Role Provider and Profile Provider Settings

- These settings are directly available in the web.config file if you have created the ASP.NET application by using the Visual Studio 2010. I'm not going to elaborate them as there are lot of articles in CodeProject describing the functionality and use of these settings and for further information you can refer to them.

Authentication Settings

```
<authentication mode="Forms">
     <forms cookieless="UseCookies" defaultUrl="HomePage.aspx"
                    loginUrl="UnAuthorized.aspx" protection="All"
timeout="30">
      </forms>
</authentication>
```

Authorization Settings

```
<authorization
        <allow roles ="Admin"/>
        <deny users ="*"/>
</authorization>
```

Membership Provider Settings

```
<membership defaultProvider="Demo_MemberShipProvider">
        <providers>
           <add name="Demo_MemberShipProvider"
                type="System.Web.Security.SqlMembershipProvider"
                connectionStringName="cnn"
                enablePasswordRetrieval="false"
                enablePasswordReset="true"
                requiresQuestionAndAnswer="true"
                applicationName="/"
                requiresUniqueEmail="false"
                passwordFormat="Hashed"
                maxInvalidPasswordAttempts="5"
                minRequiredPasswordLength="5"
                minRequiredNonalphanumericCharacters="0"
               passwordAttemptWindow="10"
passwordStrengthRegularExpression="">
        </providers>
</membership>
```

Role Provider Settings

```
 <roleManager enabled="true" cacheRolesInCookie="true"
 cookieName="TBHROLES" defaultProvider="Demo_RoleProvider">
               <providers>
                   <add connectionStringName="dld_connectionstring"
                   applicationName="/" name="Demo_RoleProvider"
                   type="System.Web.Security.SqlRoleProvider, System.Web,
                   Version=2.0.0.0, Culture=neutral,
 PublicKeyToken=b03f5f7f11d50a3a"/>
              </providers>
</roleManager>
```

Profile Provider Settings

```
<profile defaultProvider="Demo_ProfileProvider">
    <providers>
        <add name="Demo_ProfileProvider" connectionStringName="cnn"
        applicationName="/" type="System.Web.Profile.SqlProfileProvider,
        System.Web, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a"/>
    </providers>
    <properties>
        <add name="Name" type="String"/>
        <add name="DateofBirth" type="DateTime"/>
        <add name="Place" type="string"/>
    </properties>
</profile>
```

AppSettings

- In the above section, we have seen the settings available in system.web tag, now we will see the available settings in appSettings section.

appSettings element helps us to store the application settings information like connection strings, file paths, URLs, port numbers, custom key value pairs, etc.

The following code snippet shows the example of appSettings Section:

```
<appSettings>
        <add key="AppKey" value="APLJI12345AFAFAF89999BDFG"/>
</appSettings>
```

connectionStrings

- The most common section of web.config file the connectionStrings sections allows you to store multiple connection strings that are used in the application. The connectionStrings tag consists of child element with attributes name and connectionstring which is used to identify the connectionstring and the other is used to connect to the database server respectively.
- The general connectionstring settings are shown below:

```
<connectionStrings>
    <add name ="cnn" connectionString ="Initial Catalog = master;
               Data Source =localhost; Integrated Security = true"/>
</connectionStrings>
```

ConfigSections

- ConfigSections helps you to create your own custom configuration section that can be used with the web.config file. We look at this in the later section of the article, for the time being, we can have look at the configsection settings. ConfigSections should be declared just below the configuration (parent element) otherwise it is going through you an error.

```
<configSections>
    <sectionGroup name="pageAppearanceGroup">
      <section
        name="pageAppearance"
        type="PageAppearanceSection"
        allowLocation="true"
        allowDefinition="Everywhere"
      />
 </sectionGroup>
 </configSections>
```

Programmatically Accessing the Web.config File

- We can use the C# classes to read and write the values to the Web.config file.

Reading appSettings values

- The following code is used to read the appSettings values from Web.config file. You can use either of the methods shown below:

```
//Method 1:
        string key = ConfigurationManager.AppSettings["AppKey"];
        Response.Write(key);

//Method 2:
        Configuration config =
WebConfigurationManager.OpenWebConfiguration("~/");
        KeyValueConfigurationElement Appsetting =
config.AppSettings.Settings["AppKey"];
        Response.Write(Appsetting.Key + " <br/>" + "Value:" +
Appsetting.Value);
```

Reading connectionstring values

- The following code is used to read the connectionstring values from Web.config file. You can use either of the methods shown below:

```
//Method 1:
        string cnn =
ConfigurationManager.ConnectionStrings["conn"].ConnectionString;

//Methods 2:
        Configuration config =
WebConfigurationManager.OpenWebConfiguration("~/");
        ConnectionStringSettings cnnstring;

        if (config.ConnectionStrings.ConnectionStrings.Count > 0)
        {
            cnnstring = config.ConnectionStrings.ConnectionStrings["conn"];
            if (cnnstring != null)
                Response.Write("ConnectionString:" +
cnnstring.ConnectionString);
            Else
                Response.Write(" No connection string");
        }
```

Reading configuration section values

- The following code is used to read the configuration section values from Web.config file. The comments in the code will help you to understand the code:

```
// Intialize System.Configuration object.
Configuration config = WebConfigurationManager.OpenWebConfiguration("~/");
//Get the required section of the web.config file by using configuration
object.
```

```
CompilationSection compilation =
        (CompilationSection)config.GetSection("system.web/compilation");
//Access the properties of the web.config
Response.Write("Debug:"+compilation.Debug+"<br/>""+
               "Language:"+compilation.DefaultLanguage);
```

Update the configuration section values

- The following code is used to read the configuration section values from Web.config file:

```
Configuration config = WebConfigurationManager.OpenWebConfiguration("~/");
//Get the required section of the web.config file by using configuration
object.
CompilationSection compilation =
        (CompilationSection)config.GetSection("system.web/compilation");
//Update the new values.
compilation.Debug = true;
//save the changes by using Save() method of configuration object.
if (!compilation.SectionInformation.IsLocked)
{
    config.Save();
    Response.Write("New Compilation Debug"+compilation.Debug);
}
Else
{
    Response.Write("Could not save configuration.");
}
```

Encrypt Configuration Sections of Web.config File

- As we have already discussed that IIS is configured in such a way that it does not serve the Web.Config to browser, but even in some such situation to provide more security, you can encrypt some of the sections of web.config file. The following code shows you the way to encrypt the sections of web.config file:

```
Configuration config = WebConfigurationManager.OpenWebConfiguration
                       (Request.ApplicationPath);
ConfigurationSection appSettings = config.GetSection("appSettings");
if (appSettings.SectionInformation.IsProtected)
{
    appSettings.SectionInformation.UnprotectSection();
}
else
{

appSettings.SectionInformation.ProtectSection("DataProtectionConfigurationPr
ovider");
}
config.Save();
```

Custom Configuration Section in Web.config

Create Custom Configuration Section

- The ConfigurationSection class helps us to extend the Web.config file in order to fulfill our requirements. In order to have a custom configuration section, we need to follow the below steps:
- Before we actually start working with it, we will have a look at the section settings. We need to have a ProductSection element with child elements girdSettings and color. For this purpose, we will create two classes with the child elements which inherits ConfigurationElement as shown below:

```
public class GridElement : ConfigurationElement
{
    [ConfigurationProperty("title", DefaultValue = "Arial", IsRequired =
true)]
    [StringValidator(InvalidCharacters = "~!@#$%^&*()[]{}/;'\"|\\",
                          MinLength = 1, MaxLength = 60)]
    public String Title
    {
        get
        {
            return (String)this["title"];
        }
        set
        {
            this["title"] = value;
        }
    }

    [ConfigurationProperty("count", DefaultValue = "10", IsRequired =
false)]
    [IntegerValidator(ExcludeRange = false, MaxValue = 30, MinValue = 5)]
    public int Count
    {
        get
        { return (int)this["count"]; }
        set
        { this["size"] = value; }
    }
}

public class ColorElement : ConfigurationElement
{
    [ConfigurationProperty("background", DefaultValue = "FFFFFF", IsRequired
= true)]
    [StringValidator(InvalidCharacters = "~!@#$%^&*()[]{}/;
                  '\"|\\GHIJKLMNOPQRSTUVWXYZ", MinLength = 6, MaxLength = 6)]
    public String Background
    {
        get
        {
            return (String)this["background"];
        }
        set
        {
            this["background"] = value;
        }
    }

    [ConfigurationProperty("foreground", DefaultValue = "000000", IsRequired
= true)]
    [StringValidator(InvalidCharacters = "~!@#$%^&*()[]{}/;
                '\"|\\GHIJKLMNOPQRSTUVWXYZ", MinLength = 6, MaxLength = 6)]
    public String Foreground
    {
        get
        {
            return (String)this["foreground"];
        }
        set
        {
            this["foreground"] = value;
        }
    }

}
```

- Then we will create a class called ProductSection, for the root element which includes the above child elements.

```
public class ProductSection : ConfigurationSection
{
    [ConfigurationProperty("gridSettings")]
    public GridElement gridSettings
    {
        get
        {
            return (GridElement)this["gridSettings"];
        }
        set
        { this["gridSettings"] = value; }
    }

    // Create a "color element."
    [ConfigurationProperty("color")]
    public ColorElement Color
    {
        get
        {
            return (ColorElement)this["color"];
        }
        set
        { this["color"] = value; }
    }
}
```

- Then finally, we will configure these elements in Web.config file as shown below:

```
<configSections>
      <section name ="ProductSection" type ="<ProductSection"/>
  </configSections>

  <ProductSection>
    <gridSettings title ="Latest Products" count ="20"></gridSettings>
    <color background="FFFFCC" foreground="FFFFFF"></color>
  </ProductSection>
```

Access Custom Configuration Section

- The following code is used to access the custom configuration section:

```
ProductSection config =
(ProductSection)ConfigurationManager.GetSection("ProductSection");
string color =config.Color.Background;
string title =config.gridSettings.Title;
int count = config.gridSettings.Count;
```

Chapter 12

AJAX

- AJAX stands for Asynchronous JavaScript and XML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS and Java Script.
- Ajax uses XHTML for content and CSS for presentation, as well as the Document Object Model and JavaScript for dynamic content display.
- Conventional web application transmit information to and from the sever using synchronous requests. This means you fill out a form, hit submit, and get directed to a new page with new information from the server.
- With AJAX when submit is pressed, JavaScript will make a request to the server, interpret the results and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.
- XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.
- AJAX is a web browser technology independent of web server software.
- A user can continue to use the application while the client program requests information from the server in the background
- Intuitive and natural user interaction. No clicking required only Mouse movement is a sufficient event trigger.
- Data-driven as opposed to page-driven

What is AJAX?

- AJAX is just an acronym referring to Asynchronous JavaScript and XML. Now, if we take a look those words, most of us know what JavaScript and XML are, but the term Asynchronous can be confusing, so let’s focus on that.

What does asynchronous refer to?

- Asynchronous refers to events that are happening on the background independently of the main application flow. These events do not disturb the flow of the application, allowing the continuation of its normal process. A fairly good example of this happening is in your Facebook home page, when all of a sudden, without refreshing your browser window, you notice that there are new status feed updates from your friends. (Although I did notice, and think many have as well, how they do implement this. If you haven't then, try leaving your facebook homepage on the browser without moving your mouse for about 1 minute. After that minute, just move your mouse from one point to another, and all of a sudden, you will see the feeds get updated.) I'm assuming they are using some jQuery mousemove or something similar for this to happen.

- So what happens there is, facebook sends your profile information (or your user id) to their servers. Their servers then look for your friends list, grab their newly added status, return the result to the browser and then add them to your wall so that you can see. All of that, without pressing that refresh button.
- So you see, AJAX allows you to update a web page asynchronously on the background by exchanging simple, and small amounts of data. Some more examples of pages using AJAX is: Youtube, Gmail, Google Maps, StackOverflow and many more on the web.
- So you are still asking yourself, What is AJAX? Well, I'm going to ask for forgiveness, as I have no art skills, but the following image should help you just a little bit.

Advantages

- AJAX, you see, is based on internet standards. It uses a combination of the following to accomplish it's goal:
- XMLHttpRequest Object(Modern Broswers and IE7+)
- ActiveXObject (IE6 and below)
- JavaScript/DOM (Used to interact browser and server)
- XML (Returned results)
- JSON (Returned results)
- HTML (Returned results)
- These standards are browser based, making them platform independent. It doesn't matter where you program this in, as long as you have a browser, then this 'should' work. All you would need is a server with the application files, and the browser should do the rest.
- Another advantage using AJAX would be a better user interactivity. This could be named the most obvious benefit of using AJAX, and why web developers and webmasters are using AJAX more and more every day. AJAX simplifies the flow of an application, thus making it have quicker

interaction between user and website since pages are not reloaded for content to be displayed. This activity can be simplified, since the loading times can be reduced from websites.

- The advantage above opens up for this next advantage, which you can call it, smoother navigation on a website. While using AJAX, users will not have the need to use the refresh nor the back button, thus, allowing quicker response from the server.

Disadvantages

- Even though the back and refresh button are not needed while navigating a website with AJAX, these two buttons can become useless. This is due to the fact that, AJAX 'navigating' does not change you URL, so if you were in the middle of a process, and have no direct URL to where you were, then this might be bad. In some cases, the use of Hijaxing is used, which is the use of hashing url (#) at the end.
- Another disadvantage would be that it is dependent on JavaScript. While it is ok to depend on it, since most modern (if not all) already use it, but in some cases, there are users who prefer disabling JavaScript. This makes AJAX worthless. Gladly, there is a workaround this problem, as web developers must have in mind, every time they create an AJAX Website, they need to always think of this and make an optional non-AJAX version of the AJAXified website they just created.
- The last disadvantage I want to point out would be the SEO factor. Since there are no SEO Friendly URL's, then search engine tend to bypass your application, and it would appear as if that part of your site does not exist.

Implement AJAX using Update Panel

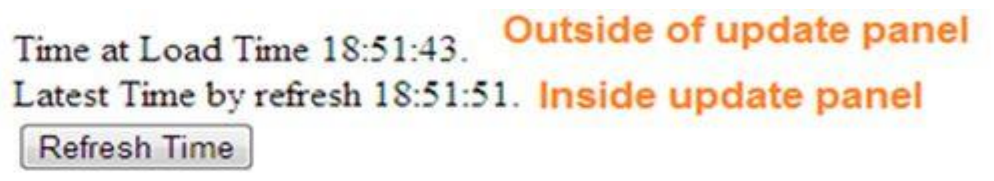
- If you are an ASP.NET developer then you probably are aware of the AJAX Extension tab in Visual Studio (2010 or higher generally) Toolbox and the tab has a few very useful tools to implement AJAX in a web application. In the first example we will implement AJAX using UpdatePanel and ScriptManager . Both tools are available in the same tab of the Toolbox.

UpdatePanel

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
```

```
    <form id="form1" runat="server">
    <div>
    //Script manager tag is here
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>

    // Code outside of Update panel
    Time at Load Time <%= DateTime.Now.ToLongTimeString() %>.

    // code inside Update panel
    <asp:UpdatePanel ID="UpdatePanel1"
         runat="server" UpdateMode="Conditional">
    <ContentTemplate>
    Latest Time by refresh <%= DateTime.Now.ToLongTimeString() %>.
    <br /><asp:Button Text="Refresh Time" ID="Button1"
        runat="server"/>
    </ContentTemplate>
    </asp:UpdatePanel>

    </div>

    </form>
</body>
</html>
```

Here is the output window:



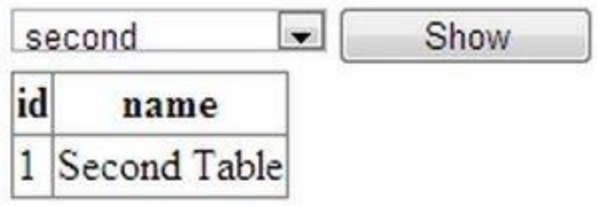
Fetch Data from Database using AJAX

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>

   <asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
            <asp:DropDownList ID="DDLAjax" runat="server" Height="16px"
Width="117px">
            </asp:DropDownList>
            <asp:Button ID="Button1" runat="server" onclick="Button1_Click"
Text="Show"
                Width="95px" />
            <br />
            <asp:GridView ID="GridView1" runat="server">
            </asp:GridView>
        </ContentTemplate>
    </asp:UpdatePanel>
    </form>
</body>
</html>
```

- If you observe the content of my aspx page then you will discover that I have kept one drop down list, one button and one GridView within the UpdatePanel. And our hero (read ScriptManager) is just in the previous line of update panel. Now it's time to observe C# code for the same aspx page.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data.SqlClient;
namespace YahooProject
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                this.DDLAjax.Items.Add("first");
                this.DDLAjax.Items.Add("second");
            }
        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            SqlConnection con = new SqlConnection();
     con.ConnectionString = "Data Source=SERVERNAME;Initial
atalog=DATABALENAME;Integrated Security=True";
            con.Open();
            SqlCommand cmd = null;

            if (this.DDLAjax.SelectedItem.Text == "first")
            {
                cmd = new SqlCommand("select * from first",con);
            }
            if (this.DDLAjax.SelectedItem.Text == "second")
            {
                cmd = new SqlCommand("select * from second", con);
            }
            this.GridView1.DataSource = cmd.ExecuteReader();
            this.GridView1.DataBind();
        }
    }
}
```

- The user will choose the table name by selecting it from the drop down list and after clicking the button the data from the relevant table will be fetched and bound with the Grid View.



id	name
1	Second Table

Implement AJAX using jQuery Method

```
<!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
src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.1/jquery.min.js">
    </script>

    <script type = "text/javascript">
        function DisplayMessageCall() {
            var pageUrl = '<%=ResolveUrl("~/AjaxServeice.asmx")%>'
            $.ajax({
                type: "POST",
                url: pageUrl + "/HelloWorld",
                contentType: "application/json; charset=utf-8",
                dataType: "json",
                success: OnSuccessCall,
                error: OnErrorCall
            });
        }
        function OnSuccessCall(response) {
            $('#<%=lblOutput.ClientID%>').html(response.d);
        }
        function OnErrorCall(response) {
            alert(response.status + " " + response.statusText);
        }
</script>

</head>
<body>
    <form id="form1" runat="server">
    <div>
    <asp:Button ID="btnGetMsg" runat="server" Text="Click Me"
OnClientClick="DisplayMessageCall();return false;" /><br />
```

```
    <asp:Label ID="lblOutput" runat="server" Text=""></asp:Label>
    </div>
    </form>
</body>
</html>
```

- In the head section of the HTML element you can see one function that I have implemented to an AJAX call in JavaScript.

```
$.ajax({
                type: "POST",
    url: pageUrl + "/HelloWorld",
    contentType: "application/json; charset=utf-8",
    dataType: "json",
    success: OnSuccessCall,
    error: OnErrorCall
});
```

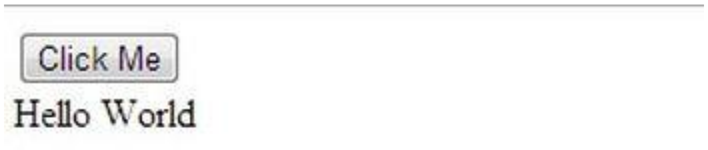
- You can see a few parameters are needed for AJAX to call using the jQuery ajax() method. My URL parameter is in the second and it's the defined location of the actual function from where the data will come. Here "HelloWorld" is the function name. And this function is located within the AjaxServeice.asmx service application page.

```
var pageUrl = '<%=ResolveUrl("~/AjaxServeice.asmx")%>'
```

- OK, let's concentrate on the service function from where the data will come in the ajax() method and in the following it is.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;

namespace YahooProject
{

    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    [System.Web.Script.Services.ScriptService] // Don't forget to uncomment
it.
    public class AjaxServeice : System.Web.Services.WebService
    {
        [WebMethod]
        public string HelloWorld()
        {
            return "Hello World";
        }
    }
}
```

Here is the output of the above example:



Example on Update Panels

- Update panels are the oldest trick in the book for ASP.NET. It is the simplest way to have ajax functionality in your web application. We will not see them in depth, just mention a few basic points about them. Below is a simple example:

```
<asp:UpdatePanel ID="SimpleUpdatePanel" runat="server">
    <ContentTemplate>
        <asp:Button ID="AjaxButton" runat="server" Text="Button" />
        <asp:Literal ID="MessageLiteral" runat="server"></asp:Literal>
    </ContentTemplate>
</asp:UpdatePanel>
```

```
Protected Sub AjaxButton_Click(sender As Object, e As EventArgs) Handles
AjaxButton.Click
    MessageLiteral.Text = "Ajax update occured"
End Sub
```

- In code behind, you have access to the entire form. That is possible because when an update panel refreshes, the whole form is been posted to the web server. But only the markup of the update panel can change. So, if we try to change something that it is not inside our update panel the change will not happen.
- It is that simple to have ajax functionality in your web application. We can also assign some event handlers so when an update panel refreshes we can execute some JavaScript code.

```
var prm = Sys.WebForms.PageRequestManager.getInstance();

prm.add_beginRequest(BeginRequestHandler);
prm.add_endRequest(EndRequestHandler);
```

```
function BeginRequestHandler(sender, args) {
    //code that runs when an UpdatePanel starts refreshing

}

function EndRequestHandler(sender, args) {
    //code that runs when an UpdatePanel has refreshed.
}
```

- With that level of abstraction, it is pretty clear than it is not a very optimal approach. If an update panel encloses a big part of the web form, it will be much slower. Even having a small update panel, the whole form is posted to the server, including the ViewState.
- Generally it is not a good practice to have nested update panels in a big depth (up to 2 is generally ok). Finally, different update panels cannot refresh simultaneously. You can read about it (and a workaround for it) here.

Page Methods

- With update panels we can update html sections of our web page. There is no direct way to get a value (e.g. a string or an integer) via ajax. For that, we can use page methods.
  Page methods are static methods declared in code behind and are exposed via JavaScript int the client. Page methods are much more lightweight since they don't post nothing from the form, except what necessary parameters each method has.
- In the following example you see a sample page method. You have to enable page methods in the ScriptManager. When calling the method in JavaScript you can set two function as attributes that will behave as success and error handlers of the ajax call.

ASP.NET part

```
<form id="form1" runat="server">
<asp:ScriptManager ID="ScriptManager1" runat="server"
EnablePageMethods="true">
</asp:ScriptManager>
<div>
    <asp:Button ID="PageMethodButton" runat="server" Text="Call Page Method"
OnClientClick="javascript:call_page_method(); return false;" />
</div>
</form>
```

JavaScript part

```
function success(data) {
    alert(data);
}

function error() {
    ...
}

function call_page_method() {
    PageMethods.HelloWorld(false, success, error);
}
```

- Page methods are lightweight. One disadvantage is that we can only use them in .aspx pages. We cannot declare a page method in a user control. Also, the JavaScript declaration of the method is embedded directly into the HTML. So, having many page methods will lead to bigger HTML files. Also, there is no way to declare them in a single place so that our entire web application can make use of them.

Web Methods

- Page methods can be very useful and practical. In the scenarios where we would like to have such methods declared globally we can use Web methods. We can declare our methods in a .asmx file. When we create the new .asmx file we need to uncomment one line of code, as the comments on the top of the page suggest.

```
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.ComponentModel

' To allow this Web Service to be called from script, using ASP.NET AJAX,
uncomment the following line.
<System.Web.Script.Services.ScriptService()> _
<System.Web.Services.WebService(Namespace:="http://tempuri.org/")> _
<System.Web.Services.WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_
1)> _
<ToolboxItem(False)> _
Public Class myMethods
    Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Shared Function HelloWorld(ByVal LowerCase As Boolean) As String
        If (LowerCase) Then Return ("Hello World").ToLower
        Return "Hello World"
    End Function
End Class
```

- In order to use the declared web methods, we need to register the .asmx service to our script manager. This can be in the asp.net markup or in the code behind.

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
    <Services>
        <asp:ServiceReference Path="~/myMethods.asmx" />
    </Services>
</asp:ScriptManager>
```

- Finally, to use the web method in our JavaScript code we need to write the full namespace. For example:

```
function call_web_method() {
    DemoWebApplication.myMethods.HelloWorld(true, success, error);
}
```