2.5.5 TypeLibrary Declarations

A TypeLibrary is a container for the contents of a DLL file corresponding to a COM object. By including a call to the TypeLibrary in the Global.asa file, the constants of the COM object can be accessed, and errors can be better reported by the ASP code. If your Web application relies on COM objects that have declared data types in type libraries, you can declare the type libraries in Global.asa.

Syntax

```
<!--METADATA TYPE="TypeLib"
file="filename"
uuid="typelibraryuuid"
version="versionnumber"
lcid="localeid"
-->
```

Parameter	Description
File	Specifies an absolute path to a type library. Either the file parameter or the uuid parameter is required
uuid	Specifies a unique identifier for the type library. Either the file parameter or the uuid parameter is required
version	Optional. Used for selecting version. If the requested version is not found, then the most recent version is used
localeid	Optional. The locale identifier to be used for the type library

Error Values

The server can return one of the following error messages:

Error Code	Description
ASP 0222	Invalid type library specification
ASP 0223	Type library not found
ASP 0224	Type library cannot be loaded
ASP 0225	Type library cannot be wrapped

Note: METADATA tags can appear anywhere in the Global.asa file (both inside and outside <script> tags). However, it is recommended that METADATA tags appear near the top of the Global.asa file.

2.5.6 Restrictions

Restrictions on what you can include in the Global.asa file:

- You can not display text that is written in the Global.asa file. This file can't display information
- You can only use Server and Application objects in the Application_OnStart and Application_OnEnd subroutines. In the Session_OnEnd subroutine, you can use Server, Application, and Session objects. In the Session_OnStart subroutine you can use any built-in object

2.5.7 How to use the Subroutines

Global.asa is often used to initialize variables.

The example below shows how to detect the exact time a visitor first arrives on a Web site. The time is stored in a Session variable named "started", and the value of the "started" variable can be accessed from any ASP page in the application:

```
<script language="vbscript" runat="server">
sub Session_OnStart
Session("started")=now()
end sub
</script>
```

Global.asa can also be used to control page access.

The example below shows how to redirect every new visitor to another page, in this case to a page called "newpage.asp":

```
<script language="vbscript" runat="server">
sub Session_OnStart
Response.Redirect("newpage.asp")
end sub
</script>
```

And you can include functions in the Global.asa file.

In the example below the Application_OnStart subroutine occurs when the Web server starts. Then the Application_OnStart subroutine calls another subroutine named "getcustomers". The "getcustomers" subroutine opens a database and retrieves a record set from the "customers" table. The record set is assigned to an array, where it can be accessed from any ASP page without querying the database:

```
<script language="vbscript" runat="server">
sub Application_OnStart
getcustomers
end sub
sub getcustomers
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
set rs=conn.execute("select name from customers")
Application("customers")=rs.GetRows
rs.Close
conn.Close
end sub
</script>
```

Global.asa Example

In this example we will create a Global.asa file that counts the number of current visitors.

- The Application_OnStart sets the Application variable "visitors" to 0 when the server starts
- The Session_OnStart subroutine adds one to the variable "visitors" every time a new visitor arrives
- The Session_OnEnd subroutine subtracts one from "visitors" each time this subroutine is triggered

The Global.asa file:

```
<script language="vbscript" runat="server">
Sub Application_OnStart
Application("visitors")=0
End Sub
Sub Session_OnStart
Application.Lock
Application("visitors")=Application("visitors")+1
Application.UnLock
End Sub
Sub Session_OnEnd
Application.Lock
Application("visitors")=Application("visitors")-1
Application.UnLock
End Sub
</script>
```

To display the number of current visitors in an ASP file:

```
<html>
<head>
</head>
<body>
<p>
There are <%response.write(Application("visitors"))%>
online now!
</p>
</body>
</html>
```

Check Your Progress

1. Define hang up and Send Card Number methods with their respective parameter and return values.
2. What is Global.asa file? Define TypeLibrary Declarations.
3. Define Lock and Unlock methods in an application object.
4. Define HHTP-COOKIE and HTTP_USER_AGENT.

2.6 LET US SUM UP

They're the things we look at, pick up, and use every day - things like our chairs, our telephones, and our computers. Each property of the object describes a particular aspect of the object. The property is actually described as a name/value pair. A method is defined as an action that an object can take. The code in a method is executed when the method is called. What is needed is a way for the object to tell the user that something has happened. The mechanism for this is called an event. An object generates an event whenever something of interest happens. One of the advantages of working with objects and events is that it awakens us to the concept of asynchronous programming. The terms synchronous and asynchronous refer to how two separate actions are related to each other. An application on the Web may be a group of ASP files. The ASP files work together to perform some purpose. The Application object in ASP is used to tie these files together. When an application is locked, the users cannot change the Application variables. Unlock method removes the lock from the Application variable. In a client-server model, when client is a web browser communicating with the server, requesting a web page. The ASP Response object is used to send output to the user from the server. The Session object is used to store information about, or change settings for a user session. Variables stored in the Session object hold information about one single user, and are available to all pages in one application. The ASP Server object is used to access properties and methods on the server. The ASP Error object is used to display detailed information of any error that occurs in scripts in an ASP page. The Global.asa file is an optional file that can contain declarations of objects, variables, and methods that can be accessed by every page in an ASP application. In Global.asa you can tell the application and session objects what to do when the application/session starts and what to do when the application/session ends.

2.7 KEYWORDS

Objects: They're the things we look at, pick up, and use every day - things like our chairs.

Property: It is actually described as a name/value pair.

Method: A method is defined as an action that an object can take.

Lock: When an application is locked, the users cannot change the Application variables.

Unlock: Unlock method removes the lock from the Application variable.

Client-server Model: In a client-server model, when client is a web browser communicating with the server, requesting a web page.

The Global.asa File: The Global.asa file is an optional file that can contain declarations of objects, variables, and methods that can be accessed by every page in an ASP application.

2.8 QUESTIONS FOR DISCUSSION

1. What is a collection?
2. What three tings make up an object?
3. What built-in object would you use to read a client's cookies?
4. Suppose that you want an object to represent a desk lamp. What properties and methods would you need? Assume that the lamp has three settings: off, dim, and full power. Design the object. (No coding required)

5. Suppose that you want an object to represent a microwave. What properties, methods, and events would you need? It can have variable power and time settings and should beep when the food is done. Design the microwave. (No coding required).

Check Your Progress: Modal Answers

1. Hang up method means to hang up the current call. It includes No parameters. Its return value is true if the telephone was hung up successfully and false if not.

   Send Card Number means to Enter or send our calling card number. Its parameter include Calling Card Number, PIN. Its return value is true if the card was accepted and false if the card was not accepted.

2. The Global.asa file is an optional file that can contain declarations of objects, variables, and methods that can be accessed by every page in an ASP application. All valid browser scripts (JavaScript, VBScript, JScript, PerlScript, etc.) can be used within Global.asa.

   A TypeLibrary is a container for the contents of a DLL file corresponding to a COM object. By including a call to the TypeLibrary in the Global.asa file, the constants of the COM object can be accessed, and errors can be better reported by the ASP code. If your Web application relies on COM objects that have declared data types in type libraries, you can declare the type libraries in Global.asa.

   Syntax

   ```
   <!--METADATA TYPE="TypeLib"
   file="filename"
   uuid="typelibraryuuid"
   version="versionnumber"
   lcid="localeid"
   -->
   ```

3. Lock method is used to prevent other users from modifying the variables in the Application object.

   Unlock method is used to enable other users to modify the variables in the Application object (after it has been locked using the Lock method)

4. HTTP_COOKIE refers to the cookies sent from the browser.

   HHTP_USER_AGENT refers to the browser type and version and information system of the client.

2.9 SUGGESTION READINGS

Jeffrey C. Jackson, *Web Technologies*, Prentice Hall, 2007

Godbole, *Web Technologies*, Tata McGraw-Hill, 2003

Ramesh Bangia, *Internet and Web Design*, firewall media

Gopalan, Gopalan/akilandeswari, *Web Technology: A Developer S Perspective*, PHI Learning Pvt. Ltd.

Ramesh Bangia, *Web Technologies*, firewall media

UNIT II

LESSON 3

UNDERSTANDING COMPONENTS

CONTENTS

3.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Understand ad-rotator component
- Discuss the browser capabilities component
- Discuss the content liking component
- Discuss the counters component
- Discuss the content rotator component
- Discuss counter component
- Discuss the permission checker component
- Discuss the text stream component and its properties

3.1 INTRODUCTION

Active Server Pages includes some useful in-built components. ASP applications use these components to handle common web development tasks. We will discuss these built in ASP components in this lesson.

3.2 ASP COMPONENTS

3.2.1 Ad-Rotator Component

The Ad-Rotator component is ideal for showing a series of graphic images each time a user visits your web page.

This component is used to display banner advertisements on the web pages of the website. This component specifies the frequency of different advertisements.

First, a rotator schedule is developed for the images. A Rotator Schedule file is defined as a text file which includes information about the images to be rotated. It involves two sections separated by asterisk (*) where the first section contains a list of keyword value pairs which may be omitted.

The keywords used in the first section of the rotator schedule file are:

- ***Redirect:*** URL of file that implements redirection.
- ***Width:*** It represents width of advertisements in pixel.
- ***Height:*** It shows the height of advertisements in pixel.
- ***Border:*** It represents size of border.

The second section contains information about each image to be rotated. Four lines of information is needed by each image:

- The first line includes the URL of the advertisement image.
- The second line contains the URL of the web page to which the graphic is linked. If the URL is not included in the advertisement graphic, then include a hyphen (-).
- The third line involves the text description of the graphic. This description is displayed by the web browsers that do not display the graphic.
- The last line involves a number (0 to 4,294,967,295) that shows the frequency of the advertisement image to be displayed. For example: if one graphic has a frequency of 1 and other has a frequency of 8, then the graphic with frequency 1 will be represented 10% of the time, and the graphic with a frequency of 8 will be displayed 80% of the time.

Example: The Asp application rotating three images.

```
<%@ LANGUAGE="VBSCRIPT" %>
<HTML>
   <HEAD>
        <TITLE>Rotating Images</TITLE>
   </HEAD>
```

```
<BODY Bgcolor=#FFFFFF>
    <%
        ' Instantiation of Advertisement Rotator Object'
  Set AdRotator = Server.CreateObject(MSWC.AdRotator")
  %>
<H2>
     The image is rotated using Advertisement Rotator object
</H2>
<%=AdRotator.GetAdvertisement ("../text/Advertisements.txt")%>
</BODY></HTML>
```

AdRotator object can be created by using the ASP AdRotator component that shows a different image every time a user enters or refreshes a page. A text file contains the information about the images.

Syntax

```
<%
set adrotator=server.createobject("MSWC.AdRotator")
adrotator.GetAdvertisement("textfile.txt")
%>
```

Example: Assume we have a file called "Banners.asp". It appears like this:

```
<html>
<body>
<%
set adrotator=Server.CreateObject("MSWC.AdRotator")
response.write(adrotator.GetAdvertisement("advertisement.txt"))
%>
</body>
</html>
```

The file "advertisement.txt" appears a below:

```
*
learnASP.jpg
http://www. tutorials.com/
Visit tutorials
80
veena.gif
http://www.veena.co.in/
Visit Veena
20
```

The lines below the asterisk in the file "advertisement.txt" mention the images to be displayed, the hyperlink addresses, the alternate text (for the images), and the display rates in percent of the hits. We see that the tutorials image will be displayed for 80 % of the hits and the veena image will be displayed for 20 % of the hits in the text file above.

Note: When a user clicks on the links, we will have to update the file "advertisement.txt" a bit to make the links to work.

```
REDIRECT Banners.asp
*
learnASP.gif
http://www. tutorials.com/
Visit tutorials
80
veena.gif
http://www.veena.co.in/
Visit Veena
20
```

To redirect, the redirection page will now receive a querystring with a variable named URL containing the URL.

Note: The following lines can be inserted under REDIRECT to specify the height, width, and border of the image:

```
REDIRECT banners.asp
WIDTH 468
HEIGHT 60
BORDER 0
*
learnASP.gif
...
...
The last thing to do is to add some lines of code to the "banners.asp" file:
<%
url=Request.QueryString("url")
If url<>"" then Response.Redirect(url)
%>
<html>
<body>
<%
set adrotator=Server.CreateObject("MSWC.AdRotator")
```

```
response.write(adrotator.GetAdvertisement("textfile.txt"))
%>
</body>
</html>
```

Properties

Property	Description	Example
Border	Specifies the size of the borders around the advertisement	<% set adrot=Server.CreateObject("MSWC.AdRotator") adrot.Border="2" Response.Write(adrot.GetAdvertisement("ads.txt")) %>
Clickable	Specifies whether the advertisement is a hyperlink	<% set adrot=Server.CreateObject("MSWC.AdRotator") adrot.Clickable=false Response.Write(adrot.GetAdvertisement("ads.txt")) %>
TargetFrame	Name of the frame to display the advertisement	<% set adrot=Server.CreateObject("MSWC.AdRotator") adrot.TargetFrame="target='_blank'" Response.Write(adrot.GetAdvertisement("ads.txt")) %>

Methods

Method	Description	Example
Get Advertisement	Returns HTML that displays the advertisement in the page	<% Response.Write(adrot.GetAdvertisement("ads.txt")) %>

3.2.2 Browser Capabilities Component

This component is used to represent html content depending on the capabilities of different browsers.

Browser Capabilities is a class located in the System.Web namespace. There is no need to create new instance of this class, you can just use the one givenby Page.Request.Browser property in the ASP.NET page.

A BrowserType object is created by the ASP Browser Capabilities component that determines the type, capabilities and version number of a visitor's browser. ASP Browser Capabilities Component has the syntax as shown below:

Code:

```
Set BrowserType = Server.CreateObject("MSWC.BrowserType")
```

The parameters are as follows:

BrowserType

This parameter shows the name of the object that is created by the call to Server.CreateObject.

3.2.3 Content Liking Component

This component is used to link several HTML pages so as to let them navigate in an easier manner. For example this component can be used to represent the online book pages.

The Content Linking component returns a Nextlink object which is used to hold a list of Web pages to be transferred.

Syntax

```
<%
Set nl=Server.CreateObject( "MSWC.NextLink" )
%>
```

Here, firstly a text file is created named "links.txt. The pages that are to be navigated are included in this file. The pages must be listed in the same order you want them to be displayed, and it also must contain a description for each file.

Note: The text file is to be modified. If you want to add a page to the list or change the order of the pages in the list, the navigation will get correct automatically.

"links.txt"

asp_intro.asp ASP Intro

asp_syntax.asp ASP Syntax

asp_variables.asp ASP Variables

asp_procedures.asp ASP Procedures

On every page listed above, insert one line of code: <!- #include file="nlcode.inc"-->. This line will contain the code below on every page listed in "links.txt" and the navigation will work.

"nlcode.inc"

```
<%
'Use the Content Linking Component
'to tranfer the pages listed
'in links.txt
dim nl
Set nl=Server.CreateObject("MSWC.NextLink")
if (nl.GetListIndex("links.txt")>1) then
Response.Write("<a href='" & nl.GetPreviousURL("links.txt"))
Response.Write("'>Previous Page</a>")
end if
```

```
Response.Write("<a href='" & nl.GetNextURL("links.txt"))
Response.Write("'>Next Page</a>")
%>
```

The ASP Content Linking Component's methods are described below:

Methods

Property	Description	Example
GetListCount	It returns the number of items that are listed in the Content Linking List file	<% dim nl,c Set nl=Server.CreateObject("MSWC.NextLink") c=nl.GetListCount("links.txt") Response.Write("There are ") Response.Write(c) Response.Write(" items in the list") %> Output: There are 4 items in the list
GetListIndex	It returns the index number of the current item in the Content Linking List file. The index number of the first item is 1. 0 is returned if the current page is not in the Content Linking List file	<% dim nl,c Set nl=Server.CreateObject("MSWC.NextLink") c=nl.GetListIndex("links.txt") Response.Write("Item number ") Response.Write(c) % > Output: Item number 3
GetNextDescription	It returns the text description of the next item listed in the Content Linking List file. If the current page is not found in the list file it returns the text description of the last page on the list	<% dim nl,c Set nl=Server.CreateObject("MSWC.NextLink") c=nl.GetNextDescription("links.txt") Response.Write("Next ") Response.Write("description is: ") Response.Write(c) %> Next description is: ASP Variables
GetNextURL	It returns the URL of the next item listed in the Content Linking List file. If the current page is not found in the list file it returns the URL of the last page on the list	<% dim nl,c Set nl=Server.CreateObject("MSWC.NextLink") c=nl.GetNextURL("links.txt") Response.Write("Next ") Response.Write("URL is: ") Response.Write(c) %> Next URL is: asp_variables.asp

Contd...

GetNthDescription	It returns the description of the Nth page listed in the Content Linking List file	<% dim nl,c Set nl=Server.CreateObject("MSWC.NextLink") c=nl.GetNthDescription("links.txt",3) Response.Write("Third ") Response.Write("description is: ") Response.Write(c) %> < p>Third description is: ASP Variables
GetNthURL	It returns the URL of the Nth page listed in the Content Linking List file	<% dim nl,c Set nl=Server.CreateObject("MSWC.NextLink") c=nl.GetNthURL("links.txt",3) Response.Write("Third ") Response.Write("URL is: ") Response.Write(c) %> Third URL is: asp_variables.asp
GetPreviousDescription	It returns the text description of the previous item listed in the Content Linking List file. If the current page is not found in the list file it returns the text description of the first page on the list	<% dim nl,c Set nl=Server.CreateObject("MSWC.NextLink") c=nl.GetPreviousDescription("links.txt") Response.Write("Previous ") Response.Write("description is: ") Response.Write(c) %> Previous description is: ASP Variables
GetPreviousURL	It returns the URL of the previous item listed in the Content Linking List file. If the current page is not found in the list file it returns the URL of the first page on the list	<% dim nl,c Set nl=Server.CreateObject("MSWC.NextLink") c=nl.GetPreviousURL("links.txt") Response.Write("Previous ") Response.Write("URL is: ") Response.Write(c) %> Previous URL is: asp_variables.asp

3.2.4 Counters Component

This component keeps track of the number of visitors to the website.

The Counters component contains an object known as Counters object which can create, store, increment, and retrieve any number of individual counters.

A counter contains an integer. A counter can be manipulated with the Get, Increment, Set, and Remove methods of the Counters object. Once the counter is created, it persists until it is removed.

Like a page hit, counters do not increment on an event automatically. Counters must be set manually or incremented using the Set and Increment methods.

Counters are not limited in scope. On creating a counter, its value can be retrieved or manipulated by any page. For example, if a counter(let say, hits) is incremented and displayed in a page called Page1.asp, and you increment hits in another page called Page2.asp, the same counter is incremented by both pages. If you hit Page1.asp and increment hits to 25, hitting Page2.asp will increment hits to 26. The next time you hit Page1.asp, hits will increment to 27.

All counters are located in a single text file, Counter.txt.

Only one Counters object is created in thete. This single Counters object can create any number of individual counters.

3.2.5 Content Rotator Component

This component is used to rotate the content on the page through HTML. For example, it can be used to display some relevant information on the main page of the website.

As we know HTML can represent text, images, colors, or hyperlinks, all these type of contents can be displayed and this content is stored in the content strings.

Syntax

```
<%
Set cr=Server.CreateObject( "MSWC.ContentRotator" )
%>
```

We show an example which shows a different content every time a user views the Web page. Create a text file named "text1.txt" in your default Web Site folder, where a subfolder called text.

"text1.txt":

```
%% #1
Have a nice day!!
%% #2
<h1>Hello</h1>
%% #3
<img src="star.gif">
%% #4
```

Here's a link.

the #number shown at the beginning of each content string is an optional parameter that shows the relative weight of the HTML content string. In this example, the Content Rotator will display the first content string one-tenth of the time, the second string two-tenths of the time, the third string three-tenths of the time, and the fourth string four-tenths of the time.

Then, create an ASP file, and insert the following code:

```
<html>
<body>
```

```
<%
set cr=server.createobject("MSWC.ContentRotator")
response.write(cr.ChooseContent("text/text1.txt"))
%>
</body>
</html>
```

3.2.6 Page Counter Component

It is used to track number of visitors and add it to a particular webpage.

A Page Counter object is created by the Page Counter component that counts and displays the number of times a Web page has been opened. The object writes the number of hits to a text file at regular intervals, so that when the server gets shutdown, the data is not lost. The Page Counter component uses an internal Central Management object to record how many number of times each page in the application has been opened.

3.2.7 Permission Checker Component

This component allows only authorized users to display the links to web pages. Only administers of a web site have permission to see them.

A Permission Checker object is created that uses the password authentication protocols given in Microsoft® Internet Information Services (IIS) to examine whether a Web user has been provided permissions to read a file.

The Permission Checker component is used to customize an ASP-based page for different types of users. For example, if a Web page contains hyperlinks, you can use the Permission Checker component to test whether the user has permissions for the required Web pages. If the user does not have the authority, you can then alter the hyperlinks to those pages the user may not access.

Syntax

Set *oVar* = Server.CreateObject("MSWC.PermissionChecker")

Parameters

oVar

It shows the name of the Permission Checker object created by the call to Server.CreateObject.

Methods

HasAccess	Determines whether the user has permissions to access a specified file.

Examples: The following example uses the above mentioned method to test whether the current user has access to a specified file. You can specify either a physical or virtual path.

```
<% Set pmck = Server.CreateObject("MSWC.PermissionChecker") %>
Physical Path Access = <%= pmck.HasAccess("c:\pages\abc\default.htm") %>
Virtual Path  Access = <%= pmck.HasAccess("/abc/default.htm") %>
```

Remarks

The following three types of password authentication are supported by IIS:

- Anonymous
- Basic
- Integrated Windows authentication

All anonymous users share the same account, so the Permissions Checker component cannot authenticate individual users when anonymous access is allowed. That is when Anonymous authentication is enabled, all users are initially comes under the IIS anonymous user account.

It is recommended to disable anonymous authentication for applications where all users have their separate accounts, such as intranet-only Web sites, so that the Permissions Checker component can authenticate individual users.

You should enable anonymous authentication and at least one other password authentication method, either integrated Windows authentication or Basic, for applications where some pages must be available to anonymous users and other pages need to be secure, such as mixed Internet and intranet Web sites. Then if you deny anonymous access to a specific page, the server will attempt to confirm the user by using either integrated Windows authentication or Basic password authentication.

The following two methods can be used to deny anonymous access to a particular page.

- Set the Access Control List for the ASP-based file to exclude the anonymous user account.
- Check for the anonymous user account and set Response.Status to the 302 unauthorized error message. This will cause IIS to attempt to identify the user by using NTLM or Basic authentication.

This is showed in the following example.

```
<%
If Request("LOGON_USER") = "" Then
  Response.Status = "302 Unauthorized"
End if
%>
```

The Permission Checker component will not able to distinguish individual user accounts, if all the files in your application must be available to anonymous users. It can still be used to ensure that the particular Web page exists and test whether the anonymous user account has access permissions for that page.

3.2.8 Text Stream Component

The Text Stream Component allows the ASP applications to access the server file system. It uses a file system object to read and write and text streams to files. It was basically designed to manage the text streams in ASP applications in an easy manner. There are two methods of the FileSystem object which can be used to manipulate text streams.

1. *CreateTextFile:* It is used to create a text file and returns a TextStream object which can be used to read and write text.
2. *OpenTextFile:* It is used to read from a text file or append text to it.

The object in the ASP statements can be instantiated in the following manner:

```
Set FileStreamObject = CreateObject("Scripting.FileSystemObject")
```

After the instantiation use either the CreateTextFile or OpenTextFile method to manipulate text streams.

Properties of the Textstream Object

Properties of the Text Stream Object are used to monitor the current position in a file, while reading a text stream.

- *At End of Line:* It is used to examine whether the end of the line has been reached when reading characters from the text stream. When only one character has to be read from a file at a time, using the read method, this property proves to be beneficial.
- *At End of Stream:* It is used to examine whether the end of the textstream has been reached. When the whole line has to be read at a time using the Read Line method, then this property proves to be useful.
- *Column:* It is used to examine the number of characters that have been currently read from the beginning of the line.
- *Line:* This property is used to examine the number of lines that have been read so far.

Net to Change

The TextStream object is used to access the contents of a text file. The code shown below creates a text file (D:\test.txt) and then writes some text to the file:

```
<%
dim fs, f
set fs=Server.CreateObject("Scripting.FileSystemObject")
set f=fs.CreateTextFile("D:\test.txt",true)
f.WriteLine("Hello!")
f.Close
set f=nothing
set fs=nothing
%>
```

The CreateTextFile or OpenTextFile methods of the FileSystemObject object can be used or you can use the OpenAsTextStream method of the File object to create an instance of the TextStream object.

The TextStream object's properties and methods are described below.

Properties

Property	Description
AtEndOfLine	Returns true if the file pointer is positioned immediately before the end-of-line marker in a TextStream file, and false if not
AtEndOfStream	Returns true if the file pointer is at the end of a TextStream file, and false if not
Column	Returns the column number of the current character position in an input stream
Line	Returns the current line number in a TextStream file

Methods

Method	Description
Close	Close Closes an open TextStream file
Read	Reads a specified number of characters from a TextStream file and returns the result
ReadAll	Reads an entire TextStream file and returns the result
ReadLine	Reads one line from a TextStream file and returns the result
Skip	Skips a specified number of characters when reading a TextStream file
SkipLine	Skips the next line when reading a TextStream file
Write	Writes a specified text to a TextStream file
WriteLine	Writes a specified text and a new-line character to a TextStream file
WriteBlankLines	Writes a specified number of new-line character to a TextStream file

Read Text File

```
<html>
<body>
<p>This is the text in the text file:</p>
<%
Set fs=Server.CreateObject("Scripting.FileSystemObject")
Set f=fs.OpenTextFile(Server.MapPath("testread.txt"), 1)
Response.Write(f.ReadAll)
f.Close
Set f=Nothing
Set fs=Nothing
%>
</body>
</html>
```

O/P:

This is the text in the text file:

Hello! How are you today

Check Your Progress

1. Define the keywords used in rotator schedule file.
2. What is content rotator component?
3. Define two methods of the FileSystem object which can be used to manipulate text streams.

3.3 LET US SUM UP

The Ad-Rotator component is used to display banner advertisements on the web pages of the website. This component specifies the frequency of different advertisements. The Browser Capabilities component is used to represent html content depending on the capabilities of different browsers. The Content liking Component is used to link several HTML pages so as to let them navigate in an easier manner. The Counters Component keeps track of the number of visitors to the website. The Content Rotator component is used to rotate the content on the page through HTML. The Page Counter component is used to track number of visitors and add it to a particular webpage. The permission Checker component allows only authorized users to display the links to web pages. The Text Stream Component allows the ASP applications to access the server file system. It uses a file system object to read and write and text streams to files.

3.4 KEYWORDS

Redirect: URL of file that implements redirection.

Width: It represents width of advertisements in pixel.

Height: It shows the height of advertisements in pixel.

Border: It represents size of border.

CreateTextFile: It is used to create a text file and returns a TextStream object which can be used to read and write text.

OpenTextFile: It is used to read from a text file or append text to it.

Column: It is used to examine the number of characters that have been currently read from the beginning of the line.

Line: This property is used to examine the number of lines that have been read so far.

AtEndOfLine: It is used to examine whether the end of the line has been reached when reading characters from the text stream. When only one character has to be read from a file at a time, using the read method, this property proves to be beneficial.

AtEndOfStream: It is used to examine whether the end of the text stream has been reached. When the whole line has to be read at a time using the ReadLine method, then this property proves to be useful.

3.5 QUESTIONS FOR DISCUSSION

1. What is Ad-Rotator component? Discuss how it is used to display advertisements. Give example.
2. What is File System object? What are the methods used for manipulation?

3. Discuss the properties of the TextStream Object which is used to monitor the current position in a file.

Check Your Progress: Modal Answers

1. The keywords used in rotator schedule file are:
   (a) ***Redirect:*** URL of file that implements redirection.
   (b) ***Width:*** It represents width of advertisements in pixel.
   (c) ***Height:*** It shows the height of advertisements in pixel.
   (d) ***Border:*** It represents size of border.
2. Content Rotator Component is used to rotate the content on the page through HTML. For example, it can be used to display some relevant information on the main page of the website.
3. There are two methods of the FileSystem object which can be used to manipulate text streams.
   (a) ***CreateTextFile:*** It is used to create a text file and returns a TextStream object which can be used to read and write text.
   (b) ***OpenTextFile:*** It is used to read from a text file or append text to it.

3.6 SUGGESTED READINGS

Jeffrey C. Jackson, *Web Technologies*, Prentice Hall, 2007

Godbole, *Web Technologies*, Tata McGraw-Hill, 2003

Ramesh Bangia, *Internet and Web Design*, firewall media

Gopalan, Gopalan/akilandeswari, *Web Technology: A Developer S Perspective*, PHI Learning Pvt. Ltd.

Ramesh Bangia, *Web Technologies*, firewall media

LESSON

4

WORKING WITH USERS

CONTENTS

4.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Discuss receiving information from the user
- Explain using form fields
- Define forms and CGI
- Discuss user input
- Understand how web forms are processed
- State what validation is
- Discuss password attributes

4.1 INTRODUCTION

In this lesson we will discuss the concept of working with users in detail. Here you will learn about forms, creating forms, designing forms, retrieving form data, submitting forms, etc. We will discuss how web forms are processed. Also we will discuss how to use different form fields such as text box, text area, etc.

4.2 RECEIVING INFORMATION FROM THE USER

A form has two duties:

1. To collect information from the user and to send that information to a separate Web page for processing.
2. Through the use of form, an ASP page can acquire the user's input, and make programmatic decisions based on that input.

Creating form is straightforward and simple. It requires as little as two lines of HTML.

It is created using the <FORM> Tag

```
1. <FORM METHOD= POST ACTION= "somepage.asp">
2. </FORM>
```

The above form is useless and serves no function. It has no text boxes for users to enter information into. It has no list boxes, radio buttons or check boxes either. On a web site, this form would be useless; however it does demonstrate how a <FORM> tag is used to collect information from the user.

Each text box, list box, check box or radio button in a form is a field. We need a way to create form fields within our form.

4.2.1 Using Form Fields

To create text boxes, check boxes, and radio buttons, we should use <INPUT> tag. The <INPUT> tag has a number of properties, but we will only concentrate on the following three:

- *Name:* The name tag uniquely identifies each element in the form.
- *Type:* The TYPE tag determines what type of form field is displayed. To display a text box, we should set TYPE equal to TEXT. To create a check box we should assign TYPE equal to CHECKBOX.
- *Value:* The value tag determines the default value for the form field. This property is important when processing the information submitted by list boxes, check boxes, and radio buttons.

4.2.2 Designing Forms

A few things are important worth considering before designing form:

1. To make sure that the form has a submitting button.
2. The form should be easy for the user to complete. We have four form fields at your disposal: the text box list box check box and radio button. Make sure that the type of form field used best fits the information needing to be collected .For example if we are going to ask for a user's mailing address it makes more sense to have a list box that contains the 50 states than to have the user type in the name of his or her state of residency.

For example the first thing to do is to ask," What information do I need from the user?" In this case the information needed would be user's name, street address, city, state and zip code. We may also want to obtain some background information on the user as well, such as how often he purchase widgets.

Based on these understandings we need following form fields:

- Text boxes would work well for entering the first and last names of the customer, as well as for the street address and zip code.
- A list box should be employed so that the users can select their states, instead of typing state names or postal abbreviations list box, containing 50 states, may seem unwieldy, but resist the temptation to use a text box. When using a text box, we are allowing users to enter anything, where a list box requires them to choose one of the viable options. If users are permitted to type in the names of their states, several different values can be received for one particular state.
- Radio buttons would work well for the background information on the user's widget buying habits.
- A simple check box would suffice for the online newsletter subscription.

4.2.3 Send Information by using Forms

A common use of intranet and Internet server applications is to process a form submitted by a browser. With ASP, we can embed scripts written in VBScript directly into an HTML file to process the form. ASP processes the script commands and returns the results to the browser.

Using a standard web browser, a user can surf to a web page with a form on it and enter information when the user does this, the information he/she id typing in has not yet been sent to the Web server. This information is not available for the web sever to process until the user submits the form by checking the form's submit button.

4.2.4 Forms and CGI

The communication for static **HTML** works only one way. There is no way to send information back to a Web server. To fix this problem, forms and **CGI** were created. Forms are **HTML** tags that allow Web page creators to include controls like check boxes, and radio buttons in their Web pages. That way, the user can enter information. It also provides a Submit button that sends the information off to the server.

But now the server has to be smarter, too. It can't just get requests for pages and send out pages. The server has to know what to do with this form information when it gets it. That's where **CGI** comes in.

CGI stands for **Common Gateway Interface**. CGI makes it possible for the Web server to talk to another application that can handle the form information when it's sent back. Often these **CGI** applications are written in a language called **Perl**. When the **CGI** application receives the form information, it can save it to a text file or store it in a database.

This system works great for simple guest books, but if we want to make our Web pages really interactive, then we will soon start running into big trouble.

The problem with **CGI** is that if five people are submitting form information at the same time, five different copies of the **CGI** application have to be running on the server to handle them. If a hundred people are submitting form information at once – a hundred copies of the application run at the same time. This is a great way to make a popular Web server fall to its knees, start crawling slowly and then fall over.

4.3 RETRIEVING FORM DATA: INPUT FUNCTION

The RequestQuery.String and Request.Form commands may be used to retrieve information from forms, like user input.

4.3.1 User Input

The Request object may be used to retrieve user information from forms:

```
<form method="get" action="simpleform.asp">
First Name: <input type="text" name="fname">
<br />
Last Name: <input type="text" name="lname">
<br /><br />
<input type="submit" value="Submit">
</form>
```

User input can be retrieved in two ways: With Request.QueryString or Request.Form.

Request.QueryString

The Request.QueryString command is used to collect values in a form with method="get". Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send.

If a user typed "Asheesh" and "Kumar" in the form example above, the URL sent to the server would look like this:

```
http://www.ibsg.org/simpleform.asp?fname=Ashessh&lname=Kumar
```

Assume that the ASP file "simpleform.asp" contains the following script:

```
<body>
Welcome
<%
response.write(request.querystring("fname"))
response.write(" " & request.querystring("lname"))
%>
</body>
```

The browser will display the following in the body of the document:

```
Welcome Asheesh Kumar
```

Example of a form with "get"

```
<html>
<body>
<form action="demo_reqquery.asp" method="get">
Your name: <input type="text" name="fname" size="20">
<input type="submit" value="Submit">
</form>
<%
dim fname
fname=Request.QueryString("fname")
If fname<>"" Then
              Response.Write("Hello " & fname & "!<br />"
              Response.Write("How are you today?")
     End If
%>
</body>
</html>
```

Form with method "post" : Request Form

```
<html>
<body>
<form action="demo_simpleform.asp" method="post">
Your name: <input type="text" name="fname" size="20">
<input type="submit" value="Submit">
</form>
<%
```

```
dim fname
fname=Request.Form("fname")
If fname<>"" Then
                Response.Write("Hello " & fname & "!<br />")
                Response.Write("How are you today?")
    End If
    %>
</body>
</html>
```

Form with radio buttons

```
<html>
<%
dim cars
cars=Request.Form("cars")
%>
<body>
<form action="demo_radiob.asp" method="post">
<p>Please select your favorite car:</p>
<input type="radio" name="cars"
<%if cars="Volvo" then Response.Write("checked")%>
value="Volvo">Volvo</input>
<br />
<input type="radio" name="cars"
<%if cars="Saab" then Response.Write("checked")%>
value="Saab">Saab</input>
<br />
<input type="radio" name="cars"
<%if cars="BMW" then Response.Write("checked")%>
value="BMW">BMW</input>
<br /><br />
<input type="submit" value="Submit" />
</form>
<%
if cars<>"" then
            Response.Write("<p>Your favorite car is: " & cars & "</p>")
end if
%>
</body>
</html>
```

4.4 UNDERSTANDING HOW WEB FORMS ARE PROCESSED

A Web form is another name for an ASP.NET page. A Web form can be made up of a single file with an .aspx extension. An .aspx file and a code behind file can be combined to make a Web form. As we'll show in the following few days, Web forms can also contain your own custom controls, defined similarly to .aspx pages, called *user controls*.

Because a Web form can include HTML, ASP.NET Web controls, custom (user) controls, and code behind, creating them can get complicated pretty quickly. However, we need to remember only a few key ideas about ASP.NET page processing so that we can develop and debug effectively.

The first key idea is that much of ASP.NET infrastructure is set up so that a single ASP.NET page can post form data back to itself repeatedly. This technique is most useful when we divide our Web site's functionality into a few main pages.

The second key idea to remember is that all ASP.NET pages are eventually compiled into executable files by the ASP.NET infrastructure. This means that every time a page is pro-cessed and rendered, a small program corresponding to each Web form is executed by the ASP.NET infrastructure.

Let's explore in more detail how Web forms are processed. Listing 1 shows a sample Web form that performs an English unit to metric unit conversion, and this is shown in Figure 4.1.

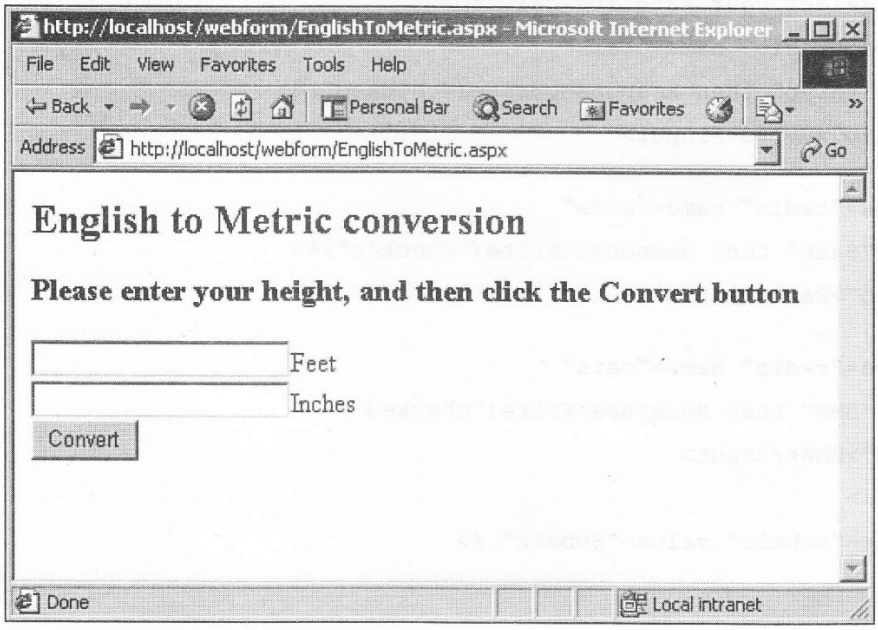

Figure 4.1: The English Unit to Metric Unit Conversion Page

Listing 1: EnglishToMetric.aspx: Converting Feet and Inches to Meters

```
1: <%@ Page Language="C#" %>
2: <html>
3: <body>
4:  <form runat="server">
```

```
 5:   <h2>English to Metric conversion</h2>
 6:   <h3>please enter your height, and then click the Convert button</h3>
 7:   <asp:Textbox id="Feet" runat="server"/>Feet
 8:   <br>
 9:   <asp:Textbox id="Inches" runat="server"/>Inches
10:   <br>
11:   <asp:Button OnClick="OnConvert" Text="Convert" runat="server"/>
12:   <br>
13:   <br>
14:   <asp:Label id="lblMeters" runat="server"/>
15:  </form>
16: </body>
17: </html>
18:
19: <script runat="server">
20: void Page_Load(Object Sender, EventArgs e)
21: {
22:   if(IsPostBack) {
23:     if(Feet.Text == "") {
24:       Feet.Text = "0";
25:     }
26:     if(Inches.Text == "") {
27:       Inches.Text = "0";
28:     }
29:   }
30: }
31:
32: void OnConvert(Object Sender, EventArgs e)
33: {
34:   Single fFeet = Single.Parse(Feet.Text);
35:   Single fInches = Single.Parse(Inches.Text);
36:   Double fMeters = 0.305*fFeet + 0.0254*fInches;
37:  lblMeters.Text = "<b>You are" + fMeters.ToString() + "meters tall</b>";
38: }
39: </script>
```

Let's examine how Web forms work using Listing 1 as our reference. Web forms like Listing 1 are processed in two different ways:

- A Web form is rendered when the user initially browses to the .aspx file. In this case, the Web form doesn't process any events because there has been no user interaction.
- A Web form may be processed after the user interacts with one of the page controls. For instance, the user might click a button on the page or select an item from a drop-down list. When the user does so, the same Web form gets hit by the user's browser, this time with information about what the user has done.

Through both cases, a Web form goes through five distinct stages when it's rendered, as shown in Figure 4.2.

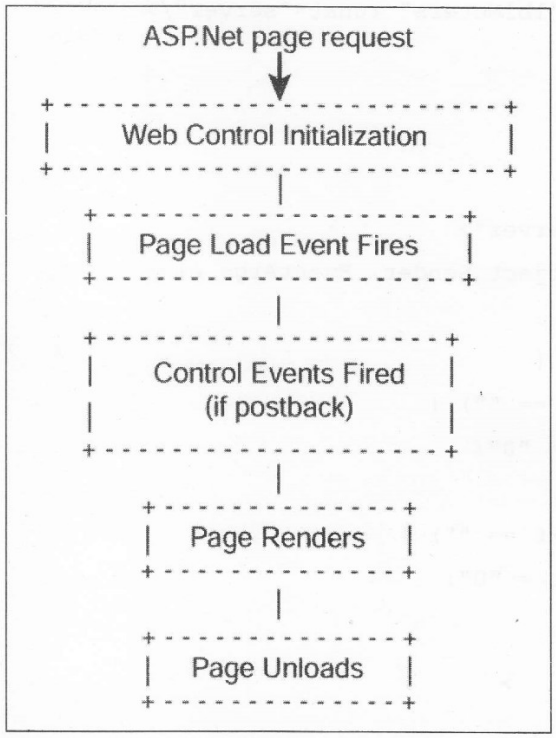

Figure 4.2: Web forms Processing Stages

In Figure 4.2, the first stage deals with Web control initialization. We don't need to be too concerned about this stage in our own programming tasks.

Our code is frequently involved in the second stage of Web forms processing, when the ASP.NET infrastructure fires the Load event in the Page class. As Listing 1 shows, we can use the Page_Load event (Lines 20-30) to change the contents of controls or to perform any other kinds of processing before a page is rendered. If the page posts back to itself (that is, if the user clicks the Convert button), the Page_Load event will fill in zeros if the user leaves any field blank.

The third stage of Web forms processing is for control event handling, which happens only if the user manipulates one of the Web controls on the page. This stage won't happen if this is the first time the user is browsing this particular page. In Listing 1, the On Convert method is called during this stage

(Lines 32-38). Note that event handling happens after the Page_Load event. In the fourth stage, the HTML for the page is rendered. This process involves calling the Render method for the page, which will call the Render method for every control, among other things.

Note:

Listing 1 doesn't contain an explicit Render method because the page class and Web control classes have default implementations for it already. We will need to override the Render method only if we want to customize the exact HTML that each class produces.

In the last stage, the Page_Unload event for every gets called. We can use the Unload event to clean up database connections and other objects that have to be closed explicitly, for example. (Listing 1 also doesn't define the Page_Unload event because it doesn't need to clean up any objects.)

4.5 USING ADVANCED FORM TECHNIQUES

4.5.1 Revisiting the Action Property, Client-Side form Validation

We have covered some of the basics of forms; it's time to discuss some of the more advanced form techniques. We've focused most of our attention so far on the various type of form fields. Each form field is responsible for gathering a discrete chunk of information from the user, while the form is responsible for gathering this information and sending it to an ASP page for processing.

To accomplish this, the <FORM> tag has two properties: Method and Action. The METHOD property determines *how* the information is passed to the form processing script. As we know that Method property can be set to either GET or POST. The ACTION property determines to what form processing script to send the users' information.

4.5.2 Revisiting the Action Property

Till now we have specified the action as a simple filename, such as AGE.asp or GetMethodExample.asp. Although there is nothing wrong with this, it restricts us to placing our form creation Web pages and form processing scripts in the same directory. What if we want our form to have Web page and form processing scripts in different directories? What if we want our form processing script on another page altogether? The ACTION property can be used to allow for either of these two scenarios.

Assume that we have a directory called scripts, which we create in the root directory. In the scripts directory, we place all our form processing scripts.

Now if we say that we plan on creating FormCreationWebPage.asp and FormProcessingScript.asp to handle form creation and form processing, respectively. If we place FormCreationScript.asp in the scripts directory, then in the form in FormCreationWebPage.asp, we would need to set the action property as follows:

```
ACTION= "/scripts/FormProcessingScript.asp"
```

The leading forward slash (/), then the directory name (scripts), and finally the form processing script filename (FormProcessingScript.asp). This is the syntax used to specify the directory and the filename of the form processing script.

4.5.3 What is Validation?

Validation is basically the act of comparing something to a given set of rules and determining if it satisfies the criteria those rules represent. In this case, the something that we are trying to validate is the input that a visitor to our site has entered into a web form.

There are a number of reasons why we'd want to do this. Some basic examples are:

- No data or incomplete data was entered
- The value of the data entered is not within the appropriate range
- The format of the entered data is not as expected

There is any number of explanations why one of the above might occur. Perhaps a spider or web-bot has come across our form and tried to submit it without entering any data, maybe the user entered a item that doesn't really exist (i.e. Feb. 29, 2002), or maybe they simply made a typo (i.e. forgot to enter one of the digits in their phone number). Whatever the reason, the best course of action is usually easiest to determine if we know there is a problem while the user is still available to fix it.

4.5.4 Form Validation to the Rescue

That's why the concept of form validation became so popular so fast. If the type of data we're expecting to receive is relatively specific, there's no reason we can't set up a set of rules by which we can validate the data we receive right as the user is giving it to us. Assuming we can't make the correction automatically, still having the user available allows us to ask them to review their submission. After all, computers are very precise, but they're not all that smart. It's often easier for the user to fix whatever's causing the error then it is for the computer to do so.

4.5.5 ASP.NET Validation Controls

Okay so now that we have an idea of what form validation is, it's time to see how ASP.NET makes it easier then it's ever been before. The key to validation in the .NET world is a set of controls called validation controls

There are 5 types of individual validation controls. They are:

- Required Field Validator
- Compare Validator
- Range Validator
- Regular Expression Validator
- Custom Validator

The Required Field Validator makes sure the user enters something. It doesn't have to be anything in particular, but they can't leave the field blank. The Compare Validator and the Range Validator both compare things using equality comparisons (the is x>=y type of thing). They only differ in that the Compare Validator is one-sided while the Range Validator allows you to specify both a lower and an upper bound. The Regular Expression Validator validates input against a regular expression. And if nothing else works for us, we can write our own criteria and encapsulate it in a Custom Validator.

In addition to these individual validation controls, there's one more kind: the Validation Summary control. This control is a great little touch by the folks from Redmond and allows us to easily check if every validator on a page is satisfied or not instead of having to check them all individually.

Example:

Here's a simple sample which utilizes a Required Field Validator to make sure the user enters something for a name:

required.aspx

```
<%@ Page Language="VB" %>
<script runat="server">
    Sub btnSubmit_Click(Sender As Object, E As EventArgs)
        ' Do Something
    End Sub
</script>

<html>
<head>
<title>ASP.NET Form Validation Sample - Required Field Validator</title>
</head>
<body bgcolor="#FFFFFF">

<p>
Note that this form doesn't actually do anything
except illustrate the Required Field Validator.
</p>

<form id="frmValidator" action="required.aspx"
    method="post" runat="server">

    Enter Your Name:
    <asp:TextBox id="txtName" runat="server" />
    <asp:RequiredFieldValidator id="valTxtName"
        ControlToValidate="txtName"
        ErrorMessage="Please enter your name!"
        runat="server" />

    <br />
```

```
    <asp:button id="btnSubmit" text="Submit"
        onClick="btnSubmit_Click" runat="server" />

</form>

<p>
Hint: Try submitting it before you enter something.
</p>

</body>
</html>
```

Since that's pretty boring, here's another sample... this time of the Range Validator:

range.aspx

```
<%@ Page Language="VB" %>
<script runat="server">
    Sub btnSubmit_Click(Sender As Object, E As EventArgs)
        ' Do Something
    End Sub
</script>

<html>
<head>
<title>ASP.NET Form Validation Sample - Range Validator</title>
</head>
<body bgcolor="#FFFFFF">

<p>
Note that this form doesn't actually do anything
except illustrate the Range Validator.
</p>

<form id="frmValidator" action="range.aspx"
    method="post" runat="server">
```

```
    Enter Your Age:
    <asp:TextBox id="txtAge" runat="server" />
    <asp:RequiredFieldValidator id="valTxtAgeReq"
        ControlToValidate="txtAge"
        ErrorMessage="Please enter your age!"
        Display="Dynamic"
        runat="server" />
    <asp:RangeValidator id="valTxtAgeRange"
        ControlToValidate="txtAge"
        Type="Integer"
        MinimumValue="21"
        MaximumValue="100"
        ErrorMessage="You need to be over 21 and under 100!"
        Display="Dynamic"
        runat="server" />

    <br />

    <asp:button id="btnSubmit" text="Submit"
        onClick="btnSubmit_Click" runat="server" />

</form>

<p>
Hint: Try entering an age under 21.
</p>

</body>
</html>
```

And here's one script that includes both at the same time and illustrates the basic use of the Validation Summary control:

summary.aspx

```
<%@ Page Language="VB" %>
<script runat="server">
```

```
    Sub btnSubmit_Click(Sender As Object, E As EventArgs)
        ' Checks to see if all the
        ' controls on the page are valid!
        If Page.IsValid Then
            ' Do Something
        End If
    End Sub
</script>

<html>
<head>
<title>ASP.NET Form Validation Sample - Validation Summary</title>
</head>
<body bgcolor="#FFFFFF">

<p>
Note that this form doesn't actually do anything
except illustrate the Validators involved.
</p>

<form id="frmValidator" action="summary.aspx"
    method="post" runat="server">

    Enter Your Name:
    <asp:TextBox id="txtName" runat="server" />
    <asp:RequiredFieldValidator id="valTxtName"
        ControlToValidate="txtName"
        ErrorMessage="Please enter your name!<br />"
        runat="server">
        ***
    </asp:RequiredFieldValidator>

    <br />
```

```
    Enter Your Age:
    <asp:TextBox id="txtAge" runat="server" />
    <asp:RequiredFieldValidator id="valTxtAgeReq"
        ControlToValidate="txtAge"
        ErrorMessage="Please enter your age!<br />"
        Display="Dynamic"
        runat="server">
        ***
    </asp:RequiredFieldValidator>
    <asp:RangeValidator id="valTxtAgeRange"
        ControlToValidate="txtAge"
        Type="Integer"
        MinimumValue="21"
        MaximumValue="100"
        ErrorMessage="You need to be over 21 and under 100!<br />"
        Display="Dynamic"
        runat="server">
        ***
    </asp:RangeValidator>

    <br />

    <asp:button id="btnSubmit" text="Submit"
        onClick="btnSubmit_Click" runat="server" />

    <asp:ValidationSummary ID="valSummary"
        HeaderText="There was an error submitting your form.
            Please check the following:"
        DisplayMode="BulletList"
        runat="server" />
</form>

</body>
</html>
```

Client-side Validation

Before we wrap on this, we should check out a cool feature of the validation controls. They automatically perform client-side validation via JavaScript on higher-level browsers. The validation still takes place on the server no matter what, but we automatically get the benefit of immediate feedback on clients that support it and we don't even have to write any client-side code.

4.6 USING THE DIFFERENT FORM FIELDS

Here we will discuss Text Boxes, List Boxes, Check Boxes, Radio Buttons, Choosing your Check Boxes and Radio Buttons.

Text Boxes

```
<INPUT TYPE="text">
```

Enables users to input text such as an email address.

```
<FORM METHOD=post ACTION="/cgi-bin/example.cgi">
<INPUT type="TEXT" size="10" maxlength="30">
<INPUT type="Submit" VALUE="Submit">
</FORM>
```

Text Box Attributes

Type: Text

Size: The size of the text box specified in characters.

Name: Name of the variable to be processed by the form processing script.

Value: Will display a default value within the text box.

Maxlength: Maximum number of characters that may be placed within the text box.

Hidden

```
<INPUT TYPE="hidden">
```

Used to send information to the form processing script that you don't want your visitors to see. Nothing will show through the browser.

```
<INPUT type="hidden" name="redirect" value="http://www.yourdomain.com/">
```

Hidden Attributes

Type: Hidden

Name: Name of the variable to be processed by the form processing script.

Value: The value of the hidden name expected by the form processing script.

Password

```
<INPUT TYPE="password">
```

Used to enable users to enter a password. When a password is typed in, asterisks will appear instead of text.

```
<FORM METHOD=post ACTION="/cgi-bin/example.cgi">
<INPUT type="password" size="10" maxlength="30">
<INPUT type="Submit" VALUE="Submit">
</FORM>
```

Password Attributes

Type: Password

Name: Name of the variable to be processed by the form processing script.

Value: Usually blank.

Size: The size of the text box specified in characters.

Maxlength: Maximum number of characters that may be placed within the text box.

Checkbox

```
<INPUT TYPE="checkbox">
```

Enables the user to select multiple options.

```
<FORM METHOD=post ACTION="/cgi-bin/example.cgi">
<INPUT type="CHECKBOX" name="selection1"> Selection 1
<INPUT type="CHECKBOX" name="selection2"> Selection 2
<INPUT type="CHECKBOX" name="selection3"> Selection 3
<INPUT type="Submit" value="Submit">
</FORM>
```

Check Box Attributes

Type: Checkbox

Checked: Specifies a default selection.

Name: Name of the variable to be processed by the form processing script.

Value: The value of the selected check box.

In the next part of this series, we will finish the form element properties and move on to some more advanced form options. Make sure you don't miss this powerful series.

Drop Down List

About Drop down List: Until now, we've been fairly closely emulating existing (CLR) Server Controls in our Demonstrations. However, for this one, we're going to depart from that path, and build one from scratch. There are several reasons for this:

1. *Length:* This article would be entirely too long if I included *everything* that exists in **System.Web.UI.WebControls.DropDownList**. I want to communicate only the points this