article is trying to make here, and all we need to demonstrate is how Events are handled in ASP.Net.

2. System.Web.UI.WebControls.DropDownList is a Data-Bound Control. Data Binding is a subject for another article altogether. Instead, we will use a slightly different (albeit not as extensible) method to "bind" the data to the Control.

3. I want to walk you through some of the kind of thinking you may practice when developing Server Controls. That is, in developing a Server Control, one needs to take a look at the client-side attributes and properties of the HTML objects that these Controls emulate, in order to determine what kind and how many server-side properties to add in order to make it work well. And I plan to do that in this article.

Radio Buttons

```
<INPUT type="radio">
```

Enables the user to select multiple options.

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

Radio Button Attributes

Type: Radio

Checked: Specifies a default selection.

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

Value: The value of the selected radio button.

Submit

```
<INPUT type="submit">
```

Enables users to submit the form information to the form processing script.

```
<INPUT type="SUBMIT" value="Submit">
```

Submit Attributes

Type:- Submit

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

Value: Specifies the text to be displayed on the submit button.

Image Submit Button

```
<INPUT type="image" SRC="url">
```

Enables users to submit the form information to the form processing script. Instead of the regular submit button, an image submit button will be displayed.

```
<INPUT type="image" name="submit" SRC="image.gif">
```

Image Submit Attributes

Type: Image

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

SRC: Image URL.

Reset

```
<INPUT type="reset">
```

Enables users to clear a form if necessary.

```
<INPUT type="RESET" value="Reset">
```

Reset Submit Attributes

Type: Reset

Value: Specifies the text to be displayed on the reset button.

Select

```
<select></select>
```

Surrounds the code for a selection drop down menu.

```
<FORM METHOD=post ACTION="/cgi-bin/example.cgi">
<SELECT SIZE="5">
<OPTION>option 1
<OPTION>option 2
<OPTION>option 3
</SELECT>
<INPUT type="Submit" value="Submit">
</FORM>
```

Select Attributes

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

Size: Specifies the number of visible selections.

Multiple: Enables users to select multiple selections.

Option

```
<option>
```

Used with the SELECT element to display the options.

```
<FORM METHOD=post ACTION="/cgi-bin/example.cgi">
```

```
<SELECT>
<OPTION>option 1
<OPTION>option 2
<OPTION>option 3
</SELECT>
<INPUT type="Submit" VALUE="Submit">
</FORM>
```

Option Attributes

Selected: Specifies a default selection.

Value: Specifies the value of the variable in the select element.

Textarea

```
<textarea></textarea>
```

Specifies an open text area.

```
<FORM METHOD=post ACTION="/cgi-bin/example.cgi">
Enter Your Comments:<BR>
<TEXTAREA wrap="virtual" name="Comments" rows=3
cols=20 maxlength=100>
</TEXTAREA><BR>
<INPUT type="Submit" VALUE="Submit">
<INPUT type="Reset" VALUE="Clear">
</FORM>
```

Textarea Attributes

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

Cols: The number of columns within the text area.

Rows: The number of rows within the text area.

Wrap: Specifies the text wrap. The default setting is off.

The WRAP can be set to "VIRTUAL" or "PHYSICAL" and will wrap the text as the user types.

Tip: In order to properly format your form, you may want to place it within a table.

Here is a basic email form set up within a table:

```
<FORM action="mailto:you@yourdomain.com">
<TABLE BORDER="0" CELLPADDING="2">
<TR>
<TD><FONT face="Verdana" size=2>Name:</FONT></TD>
<TD><INPUT name="Name" value="" size="10"></TD>
```

```
</TR>
<TR>
<TD><FONT face="Verdana" size=2>Email:</FONT></TD>
<TD><INPUT name="Email" value="" size="10"></TD>
</TR>
<TR>
<TD></TD>
<TD><INPUT type="submit" value="Submit"></TD>
</TR>
</TABLE>
</FORM>
```

4.7 ADVANCED FORMS

If we have a good form processing script, we will have the option to create highly technical forms with additional options:

4.7.1 Multi-page Forms

Provides us with the ability to create a form that spans more than one page. The data we specify will be collected on the first form page and will be transferred to the second page. We can have as many pages as we need and the data will continue to be passed through each page until the final submission. Placeholders are used within each form page to collect and pass the data.

4.7.2 Customized Confirmation Page

Enables us to create a customized confirmation page that may contain our visitor's name and any other information we've collected. In addition, we can even include the date, time and your visitor's IP address (Internet Provider).

4.7.3 Printable Confirmation Page

Enables us to provide our customers with a printable confirmation page for data such as order receipts.

4.7.4 Templates

Provides us with the ability to completely customize the information our form processes. We can use a template to specify how our data will be displayed when it is sent to our email address, and even use a template to set up a database in a specific format.

4.7.5 Choosing Checkboxes and Radio Buttons

We may create three types of buttons:

- *Submit buttons:* When activated, a submit button submits a form. A form may contain more than one submit button.
- *Reset buttons:* When activated, a reset button resets all controls to their initial values.

- ***Push buttons:*** Push buttons have no default behavior. Each push button may have client-side scripts associated with the element's event attributes. When an event occurs (e.g., the user presses the button, releases it, etc.), the associated script is triggered.

Visitors hit the SUBMIT button, appropriately enough, to submit their form. This sends the form data to the URL specified in the <FORM> tag's ACTION attribute (PonyExpress).

A SUBMIT button is yet another INPUT tag, of TYPE="submit". The optional VALUE field is the text that will appear on the button itself:

```
<input type="submit" value="Send Form">
```

A RESET button clears all fields on the form for the user to start over. This tag is not required, but it often used for the convenience of the Web visitor, who would otherwise be required to delete all form fields to start over or correct input errors.

```
<input type="reset" value="Clear Form">
```

Check Your Progress

1. What are forms?
2. Define CGI.
3. Discuss how to use textbox and text area form fields. Also define their attributes.

4.8 LET US SUM UP

Creating form is straightforward and simple. It requires as little as two lines of HTML. To create text boxes, check boxes, and radio buttons, we should use <INPUT> tag. 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. 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. 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. 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. Validation is basically the act of comparing something to a given set of rules and determining if it satisfies the criteria those rules represent. If we have a good form processing script, we will have the option to create highly technical forms with additional options. Multi-page Forms provides us with the ability to create a form that spans more than one page. Customized Confirmation Page enables us to create a customized confirmation page that may contain our visitor's name and any other information we've collected. Templates provides us with the ability to completely customize the information our form processes.

4.9 KEYWORDS

Form: Forms are HTML tags that allow Web page creators to include controls like check boxes, and radio buttons in their Web pages.

CGI (Common Gateway Interface): CGI makes it possible for the Web server to talk to another application that can handle the form information.

Validation: It is basically the act of comparing something to a given set of rules and determining if it satisfies the criteria those rules represent.

4.10 QUESTIONS FOR DISCUSSION

1. What are the two properties for the <FORM> tag?
2. What are the roles of the form creation Web page and the form processing script?
3. How can you create a text box that is 50 characters wide?
4. How do you create a check box that is checked by default?
5. What property of the <OPTION> tag is used to uniquely identify what list box option was selected by the user?
6. How do you create a text box that has a value entered by default?

Check Your Progress: Modal Answers

1. 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.
2. 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.
3. **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.

Contd...

TEXTAREA

```
<textarea></textarea>
Specifies an open text area.
<FORM METHOD=post ACTION="/cgi-bin/example.cgi">
Enter Your Comments:<BR>
<TEXTAREA wrap="virtual" name="Comments" rows=3
cols=20 maxlength=100>
</TEXTAREA><BR>
<INPUT type="Submit" VALUE="Submit">
<INPUT type="Reset" VALUE="Clear">
</FORM>Textarea Attributes
```

- *Name:* Name of the variable to be processed by the form processing script.
- *Cols:* The number of columns within the text area.
- *Rows:* The number of rows within the text area.
- *Wrap:* Specifies the text wrap. The default setting is off.

4.11 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

UNIT III

LESSON

5

COOKIES

CONTENTS

5.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Understand the concept of cookies
- Discuss application of cookies
- Discuss drawbacks of using cookies
- Discuss web browser compatibility issues
- Discuss using cookies in ASP applications
- Discuss ASP application that uses cookies

5.1 INTRODUCTION

Cookies are a general mechanism which server side connections store and retrieve information on the client side of the connection. HTTP requests made by the client which fall in that range will include a transmittal of the current value of the state object from the client back to the server. The state object is called a cookie. In this lesson we will discuss the concept of working with cookies in detail. Here we will discuss how to read Cookies, using the Request Object, how to write Cookies using the Response Object, Advantages and disadvantages of using cookies.

5.2 WORKING WITH COOKIES

Cookies are a way for you to store nuggets of information on the visitor's computer. You can then use your code to retrieve the values stored on the visitor's system at a later time.

For example, you could store the most recent searches a visitor has made on your website in a cookie. Then, when visitors search again, we could present them with their last five search criteria, or you could store visitor's location in a cookie and when they return to your site, you could display local information for your visitors.

But since the data are on the visitor's system, you cannot use cookies as your only way to identify visitors or in situations where the cookie must be present. You should always try to provide an alternative to cookies.

You place the cookies on the visitor's machine by using the cookies collection of the Response object. To write a single simple cookie to the visitor's system you would code this:

```
Response.Cookies ("NameOfCookies") = "value"
```

The *name of cookie* is the name you want to store the cookie as on the visitor's system. The value represents the text to store in the cookie. So if you coded this:

```
Response.Cookies ("Search criteria 1") = "ASP"
```

This could add the cookie with the name Search Criteria 1to the visitor's system. The cookie would contain the value ASP. If the cookie already exists, the old value will be overwritten.

A cookie is defined as a small text file that is located on client machine. Every time the same computer requests a page with a browser, it will send the cookie too.

Cookie values can be created as well as retrieved in ASP. ASP Cookies are used to store information specific to a visitor of your website This cookie is located to the user's computer for an extended amount of time. The expiration date of the cookie can be set for some day in the future and it will remain their until that day unless manually erased by the user.

Response Object Collections

Object	Purpose
Cookies	Allows you to write cookies to the visitor's browser.

In our search criteria example, we could store the five most recent searches in a complex cookie. The recent searches field is blank. But after the visitor has made a few searches, the select element would be populated with those searches.

5.2.1 Using Cookies

Cookies are a general mechanism which server side connections (such as CGI scripts) can use to both store and retrieve information on the client side of the connection. The addition of a simple, persistent, client-side state significantly extends the capabilities of Web-based client/server applications.

An Overview

A server, when returning an HTTP object to a client, may also send a piece of state information which the client will store. Included in that state object is a description of the range of URLs for which that state is valid. Any future HTTP requests made by the client which fall in that range will include a transmittal of the current value of the state object from the client back to the server. The state object is called a **cookie**, for no compelling reason.

This simple mechanism provides a powerful new tool which enables a host of new types of applications to be written for web-based environments. Shopping applications can now store information about the currently selected items, for fee services can send back registration information and free the client from retyping a user-id on next connection, sites can store per-user preferences on the client, and have the client supply those preferences every time that site is connected to.

Specification

A cookie is introduced to the client by including a **Set-Cookie** header as part of an HTTP response, typically this will be generated by a CGI script.

Syntax of the Set-Cookie HTTP Response Header

This is the format a CGI script would use to add to the HTTP headers a new piece of data which is to be stored by the client for later retrieval.

```
Set-Cookie: NAME=VALUE; expires=DATE;
path=PATH; domain=DOMAIN_NAME; secure
NAME=VALUE
```

This string is a sequence of characters excluding semi-colon, comma and white space. If there is a need to place such data in the name or value, some encoding method such as URL style %XX encoding is recommended, though no encoding is defined or required.

This is the only required attribute on the **Set-Cookie** header.

expires=*DATE*

The **expires** attribute specifies a date string that defines the valid life time of that cookie. Once the expiration date has been reached, the cookie will no longer be stored or given out.

The date string is formatted as:

Wdy, DD-Mon-YYYY HH:MM:SS GMT

This is based on variations that the only legal time zone is **GMT** and the separators between the elements of the date must be dashes.

Expires is an optional attribute. If not specified, the cookie will expire when the user's session ends.

Note: There is a bug in Netscape Navigator version 1.1 and earlier. Only cookies whose **path** attribute is set explicitly to "/" will be properly saved between sessions if they have an **expires** attribute.

domain=DOMAIN_NAME

When searching the cookie list for valid cookies, a comparison of the **domain** attributes of the cookie is made with the Internet domain name of the host from which the URL will be fetched. If there is a tail match, then the cookie will go through **path** matching to see if it should be sent. "Tail matching" means that **domain** attribute is matched against the tail of the fully qualified domain name of the host. A **domain** attribute of "acme.com" would match host names "anvil.acme.com" as well as "shipping.crate.acme.com".

Only hosts within the specified domain can set a cookie for a domain and domains must have at least two (2) or three (3) periods in them to prevent domains of the form: ".com", ".edu", and "va.us". Any domain that fails within one of the seven special top level domains listed below only require two periods. Any other domain requires at least three. The seven special top level domains are: "COM", "EDU", "NET", "ORG", "GOV", "MIL", and "INT".

The default value of **domain** is the host name of the server which generated the cookie response.

path=*PATH*

The **path** attribute is used to specify the subset of URLs in a domain for which the cookie is valid. If a cookie has already passed **domain** matching, then the pathname component of the URL is compared with the path attribute, and if there is a match, the cookie is considered valid and is sent along with the URL request. The path "/foo" would match "/foobar" and "/foo/bar.html". The path "/" is the most general path.

If the **path** is not specified, it as assumed to be the same path as the document being described by the header which contains the cookie.

Secure

If a cookie is marked **secure**, it will only be transmitted if the communications channel with the host is a secure one. Currently this means that secure cookies will only be sent to HTTPS (HTTP over SSL) servers.

If **secure** is not specified, a cookie is considered safe to be sent in the clear over unsecured channels.

Syntax of the Cookie HTTP Request Header

When requesting a URL from an HTTP server, the browser will match the URL against all cookies and if any of them match, a line containing the name/value pairs of all matching cookies will be included in the HTTP request. Here is the format of that line:

Cookie: NAME1=OPAQUE_STRING1; NAME2=OPAQUE_STRING2 ...

Additional Notes

- Multiple **Set-Cookie** headers can be issued in a single server response.
- Instances of the same path and name will overwrite each other, with the latest instance taking precedence. Instances of the same path but different names will add additional mappings.
- Setting the path to a higher-level value does not override other more specific path mappings. If there are multiple matches for a given cookie name, but with separate paths, all the matching cookies will be sent.
- The expires header lets the client know when it is safe to purge the mapping but the client is not required to do so. A client may also delete a cookie before its expiration date arrives if the number of cookies exceeds its internal limits.
- When sending cookies to a server, all cookies with a more specific path mapping should be sent before cookies with less specific path mappings. For example, a cookie "name1=foo" with a path mapping of "/" should be sent after a cookie "name1=foo2" with a path mapping of "/bar" if they are both to be sent.
- There are limitations on the number of cookies that a client can store at any one time. This is a specification of the minimum number of cookies that a client should be prepared to receive and store.
- 300 total cookies
- 4 kilobytes per cookie, where the name and the OPAQUE_STRING combine to form the 4 kilobyte limit.
- 20 cookies per server or domain. (note that completely specified hosts and domains are treated as separate entities and have a 20 cookie limitation for each, not combined)
- Servers should not expect clients to be able to exceed these limits. When the 300 cookie limit or the 20 cookie per server limit is exceeded, clients should delete the least recently used cookie. When a cookie larger than 4 kilobytes is encountered the cookie should be trimmed to fit, but the name should remain intact as long as it is less than 4 kilobytes.
- If a CGI script wishes to delete a cookie, it can do so by returning a cookie with the same name, and an **expires** time which is in the past. The path and name must match exactly in order for the expiring cookie to replace the valid cookie. This requirement makes it difficult for anyone but the originator of a cookie to delete a cookie.
- When caching HTTP, as a proxy server might do, the Set-cookie response header should never be cached.

- If a proxy server receives a response which contains a **Set-cookie** header, it should propagate the **Set**-cookie header to the client, regardless of whether the response was 304 (Not Modified) or 200 (OK).
- Similarly, if a client request contains a Cookie: header, it should be forwarded through a proxy, even if the conditional If-modified-since request is being made.

Examples

Here are some sample exchanges which are designed to illustrate the use of cookies.

First Example – Transaction Sequence

Client requests a document, and receives in the response:

Set-cookie: CUSTOMER=WILE_E_COYOTE; path=/; expires=Wednesday, 09-Nov-99 23:12:40 GMT

When client requests a URL in path "/" on this server, it sends:

Cookie: CUSTOMER=WILE_E_COYOTE

Client requests a document, and receives in the response:

Set-Cookie: PART_NUMBER=ROCKET_LAUNCHER_0001; path=/

When client requests a URL in path "/" on this server, it sends:

Cookie: CUSTOMER=WILE_E_COYOTE; PART_NUMBER=ROCKET_LAUNCHER_0001

Client receives:

Set-Cookie: SHIPPING=FEDEX; path=/foo

When client requests a URL in path "/" on this server, it sends:

Cookie: CUSTOMER=WILE_E_COYOTE; PART_NUMBER=ROCKET_LAUNCHER_0001

When client requests a URL in path "/foo" on this server, it sends:

Cookie: CUSTOMER=WILE_E_COYOTE; PART_NUMBER=ROCKET_LAUNCHER_0001; SHIPPING=FEDEX

Second Example Transaction Sequence

Assume all mappings from above have been cleared.

Client receives:

Set-Cookie: PART_NUMBER=ROCKET_LAUNCHER_0001; path=/

When client requests a URL in path "/" on this server, it sends:

Cookie: PART_NUMBER=ROCKET_LAUNCHER_0001

Client receives:

Set-Cookie: PART_NUMBER=RIDING_ROCKET_0023; path=/ammo

When client requests a URL in path "/ammo" on this server, it sends:

Cookie: PART_NUMBER=RIDING_ROCKET_0023;

PART_NUMBER=ROCKET_LAUNCHER_0001

Note: There are two name/value pairs named "PART_NUMBER" due to the inheritance of the "/" mapping in addition to the "/ammo" mapping.

5.2.2 Session Variable Cookies

Kind of Session variables are bits of information that can be stored on a user-by-user basis. These bits of information are stored on the Web server. To tie these bits of memory with a particular user, for session variables to persist across Web pages, the user must have cookies enabled. When a user first visits a page on a site that uses Session variables, two things happen:

A block of memory is allocated for that particular user's session variables on the Web server. This block of memory is identified by a unique SessionID - A cookie is written to the user's computer, storing the value of the SessionID.

When the user visits another page on the site that uses session variables, the users SessionID cookie is referenced to determine if the user already has a session variable memory block setup. If he does, the values that are stored there are referenced. Through this mechanism, one can create seemingly "global" variables that persist from one page to another.

But what if the user doesn't have cookies enabled? Session variables can still be used, but there will be no way for the user to store the SessionID as he jumps from page to page. Hence, the session variables will not be saved for the user who has cookies disabled. However, session variables that are created and referenced on the same page will work fine. For example, the following script will output "Hello, World" regardless of whether or not the user has cookies enabled:

```
<%
  Session("Message") = "Hello, World!"
  Response.Write Session("Message")
%>
```

Of course, if, on another page, the value of Session ("Message") is referenced, only those users who have cookies enabled will have a value of "Hello, World!" there.

5.3 PURPOSE OF COOKIES

There is no procedure built into the world wide web that gives the identifying mark of a visitor which is sending a request, or that provides the power to recognize that a series of requests are all coming from the same visitor (PC). Cookies were basically built to overcome the limitation of recognizing visitors to the site for the purpose of managing some continuity between visits/requests.

WWW was originally created to provide universal access to pages of information. There was no need for the system to mark a string of requests as being from the same user. Execution of that type of continuity would slow down the server's presentation.

However, cookies were created to hold data about a user which involves an identifying number. The site can recognize that user and provide approximate pages.

5.4 APPLICATION OF COOKIES

There are number of applications that need the use of cookies. Information examples that can be holded between HTTP sessions with the support of cookies include the following:

- *Personal Information:* It includes user's name, time zone, geographic location, and account number.
- *Personal Preferences:* It includes preferred background color/bitmap, background music, preferred typeface for web pages text such as point size, color, Font name.
- *Information regarding online transaction:* It includes items in shopping carts, time spent shopping, shopping cart expiration time/date.

5.5 COOKIES CREATED BY ASP PAGE

It has been wondered by some people that how a cookie gets fixed in the first place. It is not automatic. The server or the browser doe not provide any kind of automatic services to fix cookies. The cookie intimation process begins with several lines of code that are written in a page on the site. These lines of code get implemented and order the browser to built the cookie on opening the page by visitor. So, the script of the web page informs the browser to complete the cookie creation and setting.

Cookies are situated on the visitor's system. Cookies are kept on the hard-drive of the browser's system. Each cookie involves several pieces of data:

- Which domain(website) set the cookie
- One or more pieces of data
- A date of expiration for this cookie
- Optionally, keys that organize multiple data in folder-like groups

The process of storing and arranging cookies on the user's pc is browser specific.

Programmers are required to know that all browsers have the power to receive and execute generic instructions on cookies like writing cookie content to a hard disk. All browsers have the same interface regarding cookies.

Programmers write an ASP page that instructs the browser to create a cookie. A browser actually produces the cookies and put it on a client's hard disk. So the programmer is required to do nothing further than writing code that sets the cookie.

The details of actually executing the job of writing cookie contents to the hard disk may vary within browsers, but is of little or no concern to programmers. Cookies do not persist unless informed.

A programmer must fix the expiration date of the cookie for some day in the future.

5.5.1 Cookies have Size/Number Limits

Practically, the limits on keys are not a problem. However, a very active web surfer could quickly exceed the number of cookies supported by the surfer's browser. The www standards for browsers specify that at least 300 cookies should be supported. That is a total from all websites, so the server that sets 301st cookie wipe out the first cookie. Also each cookie size cannot exceed 4KB. There is also a limit of about 250 keys per cookie. Now consider it from the web server side, this means that all cookies specifically set by a web server are available to the web server. This is available in each and every page request sent from the browser to the web server.

Before sending a URL request to the ISP, the browser checks if there are any cookies that have been fixed by that domain. If there are, then the browser copies the cookie's information into the header and sends it with the URL.

Cookie data can then be extracted and used right from the first response sent back by the browser. So, when reading cookie data there is no need to make a specific call to the browser, all the cookie information set by a specific domain is sitting right in the ASP Request object, having been sent to the sever by the browser with the first request.

5.5.2 Looking at Your Cookies

It is not of direct interest to a user to see cookie content, even then you can view the cookie content easily.

With Microsoft internet explorer on Windows 95/98, look in:

C:\Windows\Cookies

where there are small text files each carrying a cookie. The file name is the username and the domain name.

With Netscape Communicator see in:

C:\program files\netscape\users\<profilename>\cookies.txt.

5.6 DRAWBACKS OF USING COOKIES

As we know that client-side cookies are powerful tools for producing Web applications, still there are a number of drawbacks regarding using cookies.

Before the deployment in a web application, it is important to consider the drawbacks which are discussed below:

1. ***Cookies can be lost:*** It is to be consider that the valuable and irreplaceable information should always be stored in a server-side database. Do not depend on cookies to store such type of information. User's hard drive can be used to store other cookie information.

   On the installation of new version of a web browser, cookies files can be corrupted, deleted, or overwritten. When using cookies, be ready for a user to lose the cookie.

2. ***Cookies can be copied:*** Do not Cookies use cookies to store information that is unique to the user's computer. Cookies are not universally unique even if a universally unique cookie is created for each HTTP transaction. It is possible for two computers to have exactly the same information.

3. ***Cookies can be changed by Users:*** Do not assume cookie information is authentic. Technically inclined users may be able to figure out how to change the value of a cookie. It is okay to depend on the value of a cookie to examine the last time a user visited a web page. However, it is not a good idea to depend on the value of a cookie to determine a user's account balance.

Non-confidential information can be stored in cookies. If confidential information is required to be stored for future reference, store it in a server side database, and assign an identification code and password to the user.

Store the identification code in a client-side cookie and ask the user for a password before allowing a transaction to be performed or using information stored in a server-side database.

4. *Cookies can be stolen:* Cookie data can be copied with or without the permission or knowledge of the owner of the cookie file.

5.7 WEB BROWSER COMPATIBILITY ISSUES

Both Client-side persistent cookies are endorsed by both Internet Explorer and Netscape Navigator. Although Internet Explorer and Netscape Navigator goes for over 95% of all Web browsers which are used to navigate the internet, some users may still be using other Web browsers that do not support cookies.

Be sure to take this into consideration, when designing Web pages. It is necessary provide a URL that can be used by a Web browser that does not support cookies.

When designing web pages, be sure to take this into consideration. Always provide a URL that can be used by a technologically challenged Web browser to access information.

Consider a case where Browser does not support cookies.

In this case, the other methods are used to pass information from one page to another in your application.

There are two methods as below:

1. Add parameters to a URL
2. Use a form

Add Parameters to a URL

You can add parameters to a URL:

```
<a href="hello.asp?fname=Sid&lname=jain">
Go to hello Page</a>
And retrieve the values in the "hello.asp" file like this:
<%
fname=Request.querystring("fname")
lname=Request.querystring("lname")
response.write("<p>Hi " & fname & " " & lname & "!</p>")
response.write("<p>Hello to everyone!</p>")
%>
```

Use a Form

When the user clicks on the Submit button, the form passes the user input to "hello.asp":

```
<form method="post" action="hello.asp">
```

```
First Name: <input type="text" name="fname" value="">
Last Name: <input type="text" name="lname" value="">
<input type="submit" value="Submit">
</form>
And retrieve the values in the "hello.asp" file like this:
<%
fname=Request.form("fname")
lname=Request.form("lname")
response.write("<p>Hello " & fname & " " & lname & "!</p>")
response.write("<p>Hello to everyone!</p>")
%>
```

5.8 USING COOKIES IN ASP APPLICATIONS

Cookies can be used very easily in ASP applications. Cookies are manipulated by these applications by using the Response and Request objects of Active Server Pages.

5.8.1 Creating Cookies

The Response object is used to create a cookie and fix its attributes. The following syntax is used for creating cookies:

Response.Cookies (NameOfCookie) = ValueOfCookie

Here, NameOfCookie is the name of the cookie which is to be created. If NameOfCookie already exists, then its value will be replaced with a new value. ValueOfCookie is the value of the cookie which is to be created. The following ASP statement creates a cookie named" TimeZone" with the value "Eastern".

Response.Cookies("TimeZone") = "Eastern"

The "Response.Cookies" command is used to create cookies. The Response.Cookies command must appear before the <html> tag.

We create a welcome cookie in the example below.

We will create a cookie named "Visitors" in the example below and assign the value visits to it.

```
<%
dim visits
response.cookies("Visitors").Expires=date+365
visits=request.cookies("Visitors")
if visits="" then
response.cookies("Visitors")=1
response.write("Welcome! This is the first time you are visiting this Web
page.")
else
response.cookies("Visitors")=visits+1
response.write("You have visited this ")
response.write("Web page " & visits)
if visits=1 then
response.write " time before!"
else
response.write " times before!"
end if
end if
%> <html> <body> </body> </html>
```

O/P: You have visited this page first time.

5.8.2 Setting Attributes of Cookies

Cookies include various attributes such as when the cookie expires and to which domain the cookie is sent. The following syntax is used to change the attributes of a cookie:

Response.Cookies (NameOfCookie).CookieAttribute = ValueOfCookieAttribute

The name of the cookie is NameOfCookie whose attribute is about to be changed. The cookie that is to be changed is named as CookieAttribute, and can be replaced with one of the values described in the table below. ValueOfCookieAttribute is the value of the cookie attribute.

Table 5.1: Cookie Attributes

Name	Description of the cookie
Expires	Used to specify when the cookie expires. This is a write-only attribute.
Domain	Used to specify to which domain the cookie is sent. This is a write-only attribute.
Path	Used to specify that cookie data must be sent in reply to request originating from a specific path. The default path is a path of the ASP application that set the cookie.. Tjis is a write only attribute.
Secure	Used to specify whether the cookie is secure. This is a write only attribute
HasKeys	Used to specift whetherthe cookies has keys. This is a read-oly attribute.

The following ASP statement creates a cookie named "TimeZone", provides it a value "Eastern", and sets it to expire on March 27, 2010.

Response.Cookie ('TimeZone") = "Eastern"

Response.Cookie ("TimeZone").Expires = "march 27, 2010

5.8.3 Storing Multiple Values in a Cookie

'Keys' are used to store multiple in a cookie. This is shown as below:

Response.Cookies("NameOfCookie")("NameOfKey") = ("ValueOfKEY")

Different names must be given to the keys, which then store values

Response.Cookies('MyCookie")("CustId") = ("CID01")

Response.Cookies('MyCookie")("Name") = ("Ivan")

Response.Cookies('MyCookie")("PinCd") = ("400057")

By using this method, a cookie can be built on a client's computer, which can carry multiple values. This type of cookie, when set, is called a Cookie Dictionary. In the sample code above MyCookie is a cookie dictionary.

5.8.4 Retrieving a Single Value from a Cookie

The Request object is used to retrieve the value of cookie. The following ASP statement retrieves the value of the cookie named NameOfCookie and stores this in a variable.

myvar = Request.Cookies(NameOfCookie)

Request.Cookies" command is used to retrieve a cookie value.

We retrieve the value of the cookie named "name" and display it on a page as shown in the following example:

```
<%
gname=Request.Cookies(name")
response.write("name=" & gname)
%>
```

O/P: name=Alex

5.8.5 Retrieving Multiple Values from Cookie Keys

The following syntax is used to retrieve the values held in a cookie's sub keys:

MyVar = Request.Cookie("NameOfCookie")("NameOfSubkeys")

5.8.6 Testing if a Cookie has Key Values or Not

It is easy to examine whether a cookie has sub keys by using the following ASP statement:

MyVar = Request.Cookies(NameOfCookie).HasKeys

The cookie has sub keys, if the result is TRUE . If the result is FALSE, the cookie does not have sub keys. If the cookie has keys, its key values can be retrieved.

If a cookie has keys, a simple FOR....NMEXT loop can be written to retrieve key values and write them to memory variables for use later.

5.9 AN ASP APPLICATION THAT USES COOKIES

The ASP application in the example below shows how an ASP application deletes cookies created by the ASP application.

The cookies are deleted through the use of the Expire attribute of the response object to specify an expiry date that has already passed.

This ASP application deletes cookies created by the ASP application.

```
<% @ LANGUAGE="VBSCRIPT" %>
<% RESPONSE.COOKIES("NUMVISITS").EXPIRES="JULY 15. 1999" %>
<% Response.Cookies ("LastDate").Expires=" june10, 2001" %>
<% Response.Cookies ("LastTime").Expires="june 10, 2010" %>
<HTML>
     <HEAD>
           <TITLE>Cookies are Deleted</TITLE>
     </HEAD>
    <BODY BGCOLOR="FFFFFF">
          <H1>Cookies are Deleted></H1>
<A Href=cookies.asp>Click here to start</A>
    </BODY>
</HTML>
```

Check Your Progress

1. What are session variable cookies?
2. Discuss how to create cookies.
3. How is single value retrieved from cookie?

5.10 LET US SUM UP

Cookies are a way for you to store nuggets of information on the visitor's computer. You can then use your code to retrieve the values stored on the visitor's system at a later time. Cookies are a general mechanism which server side connections (such as CGI scripts) can use to both store and retrieve information on the client side of the connection. A server, when returning an HTTP object to a client, may also send a piece of state information which the client will store. HTTP requests made by the client which fall in that range will include a transmittal of the current value of the state object from the client back to the server.

Kind of session variables are bits of information that can be stored on a user-by-user basis. These bits of information are stored on the Web server. To tie these bits of memory with a particular user, for session variables to persist across Web pages, the user must have cookies enabled.

Cookies were basically built to overcome the limitation of recognizing visitors to the site for the purpose of managing some continuity between visits/requests. However, cookies were created to hold data about a user which involves an identifying number. The site can recognize that user and provide approximate pages.

The cookie indication process starts with several lines of code that are written in a page on the site. When the visitor opens that page, these lines of code get implemented and order the browser to built the cookie.

The details of actually executing the job of writing cookie contents to the hard disk may vary within browsers, but is of little or no concern to programmers. Cookies do not persist unless instructed.

The www standards for browsers specify that at least 300 cookies should be supported. That is a total from all websites, so the server that sets 301st cookie wipe out the first cookie.

Cookies can be used very easily in ASP applications. Cookies are manipulated by these applications by using the Response and Request objects of Active Server Pages.

5.11 KEYWORDS

Cookies: Cookies are a way for you to store nuggets of information on the visitor's computer.

Path: The path attribute is used to specify the subset of URLs in a domain for which the cookie is valid.

5.12 QUESTIONS FOR DISCUSSION

What are cookies? Discuss the applications of cookies.

Discuss the purpose of cookies and explain the creation of cookies by ASP page.

Discuss the concept of using cookies in ASP application.

What are the limitations of using cookies?

Check Your Progress: Modal Answers

1. Kind of. Session variables are bits of information that can be stored on a user-by-user basis. These bits of information are stored on the Web server. To tie these bits of memory with a particular user, for session variables to persist across Web pages, the user must have cookies enabled. When a user first visits a page on a site that uses Session variables, two things happen: A block of memory is allocated for that particular user's session variables on the Web server. This block of memory is identified by a unique SessionID -- A cookie is written to the user's computer, storing the value of the SessionID

2. The Response object is used to create a cookie and fix its attributes. The following syntax is used for creating cookies:

   Response.Cookies (NameOfCookie) = ValueOfCookie

Contd...

Here, NameOfCookie is the name of the cookie which is to be created. If NameOfCookie already exists, then its value will be replaced with a new value. ValueOfCookie is the value of the cookie which is to be created. The following ASP statement creates a cookie named" TimeZone" with the value "Eastern".

Response.Cookies("TimeZone") = "Eastern"

3. The Request object is used to retrieve the value of cookie. The following ASP statement retrieves the value of the cookie named NameOfCookie and stores this in a variable.

myvar = Request.Cookies(NameOfCookie)

5.13 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

UNIT IV

LESSON 6

WORKING WITH FILES AND FILE SYSTEM

CONTENTS

6.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Understand the concept of ASP files
- Discuss SQL statement with the connection object
- Understand session and connection pooling

6.1 INTRODUCTION

Information has been acknowledged as one of the most important resource in today's complex business environment. The organizations in order to survive in this competitive world need to take effective decisions. To be effective, the decision making process which relies mainly on the internal information processing needs to be in place and able to provide quality information. The effective decision-making therefore requires availability of right information at the right time in the right quantity to the right person. This essentially requires gathering and storing data on various aspects of the organizational working at one place; say on a computer system, in a form which aids in fast and easy access to data and its conversion into information. Besides easy access, a computerized system should also be able to manage data efficiently. The purpose of storing data is served in two possible ways:

- Using Application Files
- Using Databases

6.2 ASP FILES

The File System Component that is included with IIS performs all the file interactions in ASP. It contains many objects that provide you a window into your system's file system. Some of the more important objects involves:

- File System Object
- File Object
- Folder Object

6.2.1 ASP Creating the File System Object

The File System Object is used to manipulate files, folders, and directory paths. It is used to access the file system on a server.

You can also retrieve file system information using this object.

For creating file system object(FSO), create an object reference to it like any other object in ASP.

In the example below we create a file system object is created and then destroyed.

```
<%
Dim myFSO
Set myFSO = Server.CreateObject _
("Scripting.FileSystemObject")
Set myFSO = nothing
%>
```

6.2.2 Using the File Object

For retrieving a File Object in ASP, the relative or complete file path to the desired file must be known. In the example below, we will assume that a file exists at "C:\Inetpub\wwwroot\ASP\script.asp".

myFO means File Object. This small script will get a file object and print out the filename using the Name property.

```
<%
Dim myFSO, myFO
Set myFSO = Server.CreateObject _
("Scripting.FileSystemObject")
Set myFO = myFSO.GetFile _
("C:\Inetpub\wwwroot\ASP\script.asp")
Response.Write("The filename is: " & myFO.Name)
Set myFO = nothing
Set myFSO = nothing
%>
```

O/P: The filename is: script.asp

The File object is used to return information about a particular file. An instance of the File object is to be created through the FileSystemObject to work with the properties and methods of the File object. With the use of the GetFile method of the FileSystemObject object or the Files property of the Folder object, create a FileSystemObject object and then instantiate the File object.

To instantiate the File object and the DateCreated property to return the date when the specified file was created,the following code uses the GetFile method of the FileSystemObject object:

```
<%
Dim fs,f
Set fs=Server.CreateObject("Scripting.FileSystemObject")
Set f=fs.GetFile("D:\test.txt")
Response.Write("File created: " & f.DateCreated)
set f=nothing
set fs=nothing
%>
```

Output: File created: 9/19/2001 10:01:19 AM

The File object's properties and methods are described below: