

Client Side Scripting Language (22519)

THEORY PAPER	: 70
PROGRESSIVE ASSESSMENT	: 30 (PTT + MICRO PROJECT)
TERM WORK	: 25
EXTERNAL PRACTICAL EXAM	: 25

Chapter 4: Cookies and Browser Data (8 marks)

Cookies

Basic of cookies, reading a cookie value, writing a cookie value. creating a cookies, deleting a cookies, setting the expiration date of cookie.

Browser

Opening a window, giving the new window focus, window position, changing the content of window, closing a window, scrolling a web page, multiple windows at once, creating a web page in new window, JavaScript in URLs, JavaScript security, Timers, Browser location and history.

Basics of Cookie

A cookie is a small amount of named data stored by the web browser and associated with a particular web page or website.

The text of a cookie must contain a name-value pair, which is a name and value of the information. When you write your JavaScript, you decide on the name and the value. Suppose, for example, that a cookie is used to store a user ID; *userid* is the name of the information and *user123* is the value. Here's how this name value would be stored in the cookie:

`userid = 'user123' ;`

Diagram illustrating the components of the cookie text `userid = 'user123' ;`:

- `userid`: name
- `=`: Assignment operator
- `'user123'`: value
- `;`: delimiter

You cannot include semicolons, commas, or white space in the name or the value unless you precede these characters with the escape character (\). The escape character tells the browser that the semicolon, comma, or white space is part of the name or value and not a special character.

Types of cookies

Session cookies

- A session cookie resides in memory for the length of the browser session. A browser session begins when the user starts the browser and ends when the user exits the browser.

Persistent cookies

- A persistent cookie is written to the computer's hard disk and remains there until the expiration date has been reached; then it's deleted.

Creating a Cookie

Every cookie has four parts: a name, an assignment operator, a value, and a semicolon. The semicolon is a delimiter and not part of the value. A delimiter is a character that indicates where something ends, which in this case is the end of the cookie.

This statement creates a cookie, where CustomerName is the name and Sam is the value:

```
window.document.cookie = "CustomerName= Sam;"
```

In above statement cookie property of document is used to save cookie, a cookie is a combination of four parts.

Creating Cookie JS Example

```
<html>
<head>
<title>Write Cookie</title>
<script>
    function WriteCookie()
    {
        with (document.CookieWriter)
        {
            document.cookie = "CustomerName=" + customer.value + ";";
            alert("Cookie Written")
        }
    }
</script>
</head>
<body>
<form name="CookieWriter" action="" >
    Enter your name: <input type="text" name="customer" onchange="WriteCookie()"/>
</form>
</body>
</html>
```

Reading a Cookie

Reading a cookie is just as simple as writing one, because the value of the `window.document.cookie` object is the cookie. You can then use `window.document.cookie` whenever you want to access the cookie.

Following is the statement used to read cookie from ***document.cookie*** object, but as cookie contains name and value separated by assignment operator (=), we used `split()` method to separate name and value. `Split()` method will return array, which contains name of cookie as first element with index 0 and value of cookie as second element with index **1**.

```
cookie_name = document.cookie.split('=')[0]  
cookie_value = document.cookie.split('=')[1]
```

Reading Cookie JS Example

```
<html >
<head>
<title>Read Cookie</title>
<script>


function ReadCookie()
{
with (document.CookieReader)
{
if (document.cookie == "")
{
cookiecontent.value = "No cookies"
}
else
{
cookiecontent.value = document.cookie.split('=')[1]
}
}
}
</script>
</head>
```

```
<body>
<form name="CookieReader" action="" >
Cookie: <input type="text" name="cookiecontent" />
<BR>
<INPUT name="Reset" value="Get Cookie" type="button"
onclick="ReadCookie()"/>
</FORM>
</body>
</html>
```



Setting the Expiration Date

By default, cookies will be automatically deleted once the browser is closed. This prevents users from re-using the cookie values on subsequent visits to your page. You can override this by setting an expiration date for your cookie. This can be done easily by adding ***expires=expirationDate*** in UTC separated by semicolon from the ***name=value***, as seen in the following example:


```
document.cookie = "CustomerName1="+ CustomerName+";expires=" + expireDate.toGMTString()
```




Name of a
cookie



CustomerName is a
variable contains
value of a cookie



expires attribute of a
cookie used to set
expiry date



expireDate is a date type
variable (contains date)
converted to string.

JS Function for Setting expiry date of cookie

OR

JS function to create a persistent cookie

```
function WriteCookie()  
{  
    var expireDate = new Date expireDate.setMonth(expireDate.getMonth()+3)  
    with (document.CookieWriter)  
    {  
        var CustomerName = customer.value document.cookie = "CustomerName1="+  
            CustomerName+";expires=" +expireDate.toGMTString()  
  
        alert("Cookie Written")  
    }  
}
```

Deleting a cookie

To ***delete*** a cookie, you can simply provide an *expiration date* that occurred in the past. When the browser sees that the cookie has expired, it will delete the cookie automatically.

`expireDate.setDate(expireDate.getDate()-1)`

The `getDate()` method returns the system date on the computer that is running the JavaScript. We subtract 1 from the current date and pass it to the `setDate()` method, which assigns the new date to the `expireDate` variable. The `expire Date` variable is then converted to a string and concatenated to the cookie string,

Opening a window

`window.open()`

It is a pre-defined window method of JavaScript used to open the new tab or window in the browser. This will depend on your browser setting or parameters passed in the *window.open()* method that either a new window or tab will open.

Syntax:

`open(URL, name, specs, replace)`

- **URL** : This optional parameter of the `window.open()` function contains the URL string of a webpage, which you want to open. If you do not specify any URL in this function, it will open a new blank window.

name:

Using this parameter, you can set the name of the window you are going to open. It supports the following values:

<code>_blank</code>	Passed URL will load into a new tab/window.
<code>_parent</code>	URL will load into the parent window or frame that is already opened.
<code>_self</code>	By passing this parameter, the URL will replace the previous output and a new window will open in the same frame.
<code>_top</code>	URL replaces any frameset that can be loaded.
<code>name</code>	Provide the name of the new window to show the text or any data on it. (Note - not the title of the window)

Specification / style of a window

- This parameter contains the settings that are separated by the comma. Element used in this parameter cannot have whitespaces, **e.g., `width=150,height=100`**.

replace:

- Like the other parameters of `window.open()` method, this is also an optional parameter. It either creates a new entry or replaces the current entry in history list. It supports two Boolean values; this means that it returns either `true` or `false`:

Style	Description	Values (on=1, off=0)
status	The status bar	status=1, status=0
toolbar	The standard browser toolbar	toolbar=1, toolbar=0
location	The Location entry field	location=1, location=0
menubar	The menu bar	menubar=1, menubar=0
directories	The standard browser directory buttons	directories =1, directories =0
resizable	Allow/disallow the window to be resized	resizable=1, resizable=0
scrollbars	Enable the scrollbars	scrollbars=1, scrollbars=0
height	The height of the window in pixels	height=250
width	The width of the window in pixels	width=250

Window Styles

Opening a New window - JS Example

```
<html>
```

```
<body>
```

```
  <script>
```

```
    function openWindow()
```

```
    {
```

```
      window.open('https://www.product.html');
```

```
    }
```

```
  </script>
```

```
    Click the button to open new window
```

```
    <br><br>
```

```
    <button onclick="openWindow()"> Open Window </button>
```

```
</body>
```

```
</html>
```

Giving the new window focus

You give a new window focus by calling the `focus()` method of the new window after the new window opens.

As shown in JS function below, the ***MyWindow*** variable receives a reference to the new window when ***window.open()*** is called, but we want to set focus on pervious window (main window) so to do that we used ***this*** to point previous calling window. ***this.focus()*** line will set focus to previous (main) window.

```
function OpenNewWindow()
{
    MyWindow = window.open('MyWebSite/MyAd.jpg', 'AdWin',
                           'status=0, toolbar=0, menubar=0, resizable=0, height=250, width=250')

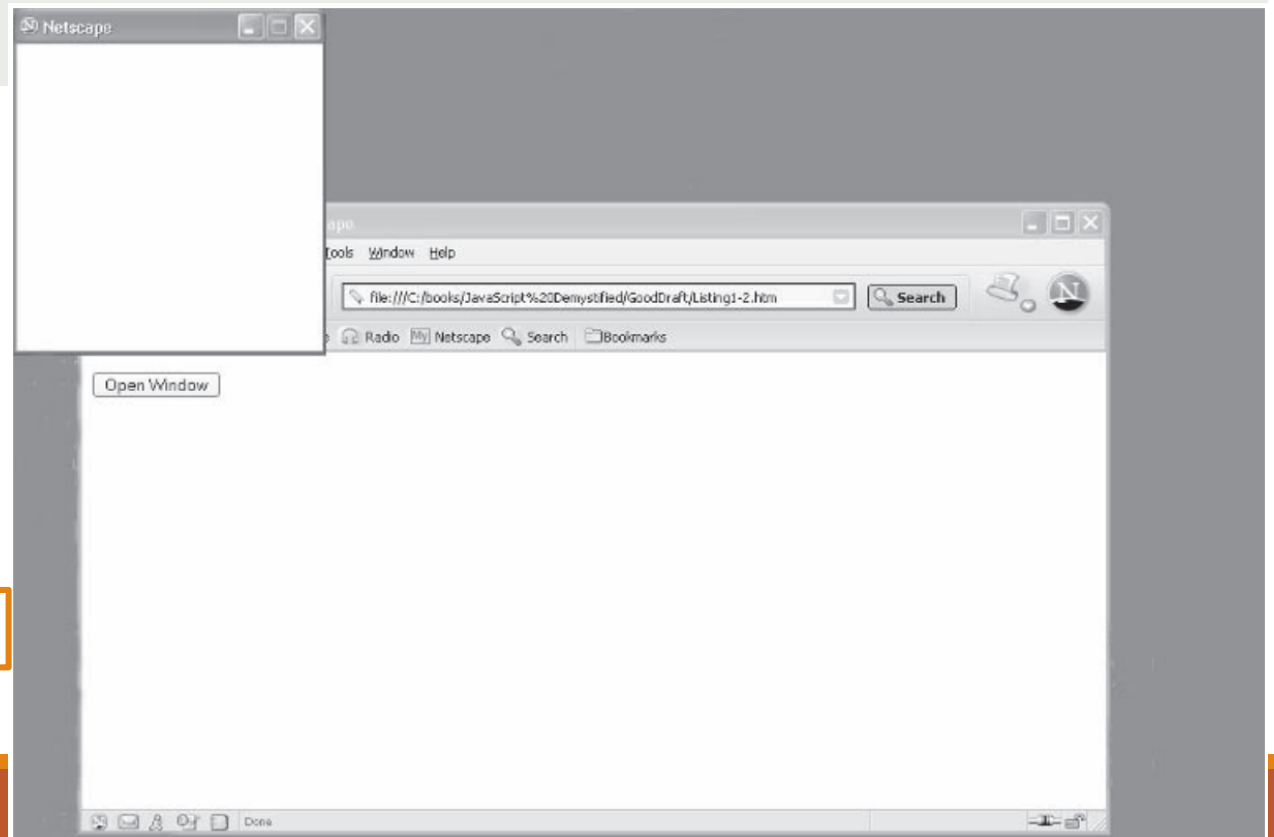
    this.focus()
}
```


Window position

We can specify the location by setting the **left** and **top** properties of the new window when you create it. The left and top properties create the x and y coordinates, in pixels, of the upper-left corner of the new window.

```
function OpenNewWindow() {  
  
    MyWindow = window.open('MyWebSite/MyAd.jpg', 'myAdWin', 'width=250,height=250,left=0,top=0')  
  
}
```

Output



Changing the content of window

Sometimes you'll want to change the content of an open window rather than having to close and open a new window each time that you want to display something different in the window. Suppose, for example, that you want the window to display a product each time a customer selects the item on your web page.

The first time a customer selects an item, you open a new window with a name ***productWindow*** and displaying the appropriate product in the window. The next time a customer selects an item, you again open a new window with a same name as ***productWindow*** and displaying a different product.

Since both windows have the same name, the browser replaces the first window with the second window. The result is that the window appears to remain open, but the content of the window changes

Detailed example is given on next page

JS Example - Changing Content of a New Window

```
<html>
<head>
  <title>Changing Content of Window</title>
  <script>
    function OpenNewWindow(Ad) {
      MyWindow = window.open(Ad, 'myAdWin', 'height=250, width=250')
    }
  </script>
</head>
<body>
  <FORM action="" method="post">
    <P>
      <INPUT name="ProductA" value="Product A" type="button" onclick="OpenNewWindow('MyWebSite/MyAd1.jpg')"/>
      <INPUT name="ProductB" value="Product B" type="button" onclick="OpenNewWindow('MyWebSite/MyAd2.jpg')"/>
    </P>
  </FORM>
</body>
</html>
```

Scrolling a webpage

JavaScript is used to scroll the web page automatically by calling the ***scrollTo()*** or ***scrollBy()*** methods of the window object, or a link led directly to a relative link in the page.

- The method ***scrollBy(x,y)*** scrolls the page relative to its current position. For instance, `scrollBy(0,10)` scrolls the page 10px down.
- The method ***scrollTo(pageX,pageY)*** scrolls the page to absolute coordinates considering current position as origin (0,0) .

It's like setting `scrollLeft/scrollTop` properties.

To scroll to the very beginning, we can use `scrollTo(0,0)`

```
</head>
<body>
  <FORM action="http://www.jimkeogh.com" method="post">
    <P>
      <INPUT name="GoToTop" value="Go To Top"
        type="button" onclick="Top()" />
    </P>
  </FORM>
</body>
</html>
```

Opening Multiple Windows at Once

Some web sites bombard you with windows as soon as you enter the site. New windows pop up all over the screen. This is a nasty and annoying feature, because most users probably don't know how to get out of this maze. Nevertheless, the following example shows you how to open multiple windows onscreen.

This example displays five new windows when the Windows Gone Wild button is clicked, which is at least better than those annoying web sites that launch a battery of windows when the `onload` event occurs (Figure 9-4).

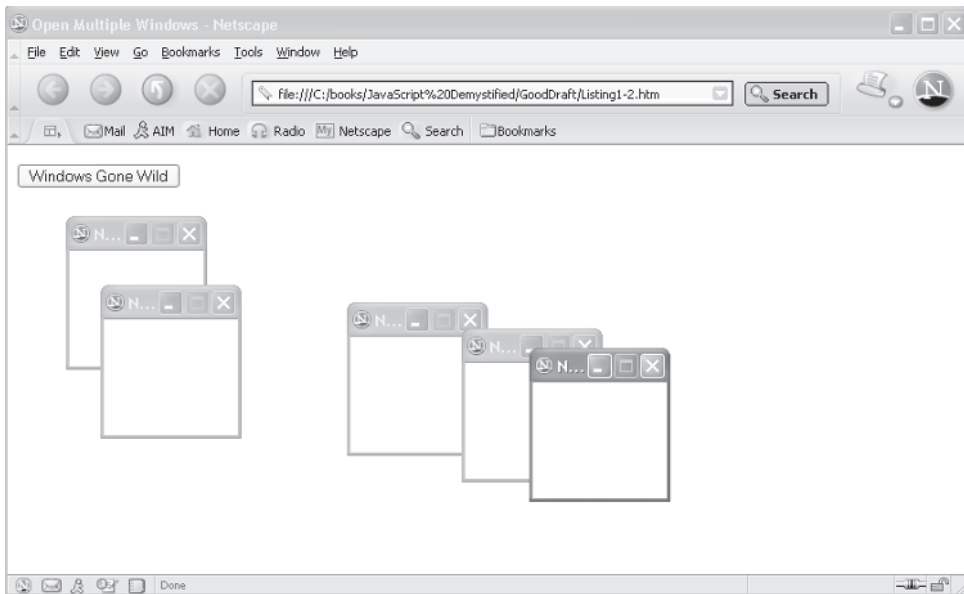


Figure 9-4 A JavaScript can be used to open multiple windows.

The Windows Gone Wild button calls the `Launch()` JavaScript function, which uses a for loop to execute the `open()` method five times to open five empty windows. Notice that the first parameter of the `open()` method is empty because we want to display blank windows. You can, of course, insert a URL for the content you want to display in the window.

```
<!DOCTYPE html PUBLIC
    "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Open Multiple Windows</title>
    <script language="Javascript" type="text/javascript">
    <!--
        function Launch() {
            for (i=0; i < 5;i++)
            {
                Win =
                    window.open('', 'win'+i, 'width=50,height=50')
            }
        }
    -->
    </script>
</head>
<body>
    <FORM action="http://www.jimkeogh.com" method="post">
        <P>
            <INPUT name="WindowsGoneWild"
                value="Windows Gone Wild" type="button"
                onclick="Launch()" />
        </P>
    </FORM>
</body>
</html>
```

Creating a Web Page in a New Window

You can place dynamic content into a new window by using the `document.write()` method to write HTML tags to the new window. Though these sorts of script are a little tricky to write, you'll develop the knack for doing this after studying the next example.

This example displays a button that, when clicked, calls the `Window()` JavaScript function that creates a new window and writes HTML tags to the new window. The HTML tags are passed a string to the `MyWindow.document.write()` method. `MyWindow` is referenced to the new window object that was created by the `open()` method. The `document` is the document object contained within the new window. The `write()` method is a method of the document object.

Anything written by the `write()` method appears in the new window. Remember that when the browser sees an HTML tag, the browser interprets it according to HTML rules.

Look carefully, and you'll notice that the string passed to all the `write()` methods contains HTML tags that display a form in the new window. The form consists of an input text box, where the customer is expected to enter a name. Also on the form is a Submit Query button that, when clicked, sends the customer name to the server CGI application for processing (Figure 9-5).

The most efficient way to create dynamic content is first to create the content as a web page—that is, write the HTML tags as if the content were being written for your home page. Once you are satisfied with the content, place double quotation marks around each line to create a string; then pass each string to the `write()` method after the new window is opened.

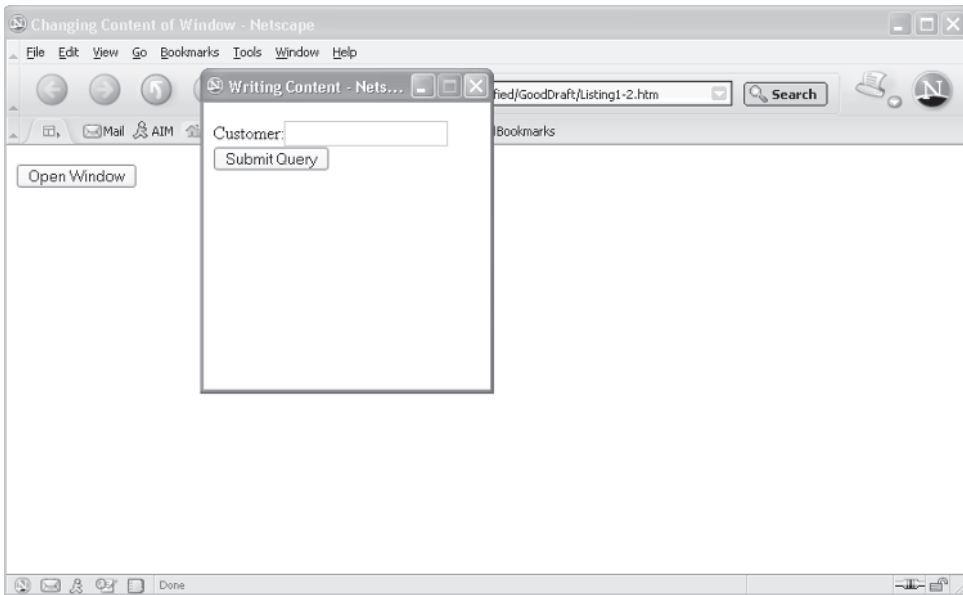


Figure 9-5 Dynamic content of a new window can be created by a JavaScript.

Note that some HTML tags contain a forward slash (/), which has a particular meaning to the browser. You'll need to precede these with a backslash (\), which tells the browser to ignore the special meaning.

```
<!DOCTYPE html PUBLIC
    "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Changing Content of Window</title>
    <script language="Javascript" type="text/javascript">
        <!--
        function Window() {
            MyWindow = window.open
                ('', 'myWin', 'height=250, width=250')
            MyWindow.document.write('<html>')
            MyWindow.document.write('<head>')
            MyWindow.document.write
                ('<title> Writing Content<\/title>')
            MyWindow.document.write('<\/head>')
            MyWindow.document.write('<body>')
            MyWindow.document.write
                ('<FORM action="http://www.jimkeogh.com"
                    method="post">')
            MyWindow.document.write('<P>')
            MyWindow.document.write
                'Customer:<INPUT name="FirstName"
                    type="text" \/>')
            MyWindow.document.write('<BR>')
            MyWindow.document.write
                ('<INPUT name="submit" type="submit" \/>')
            MyWindow.document.write('<\/P>')
            MyWindow.document.write('<\/FORM>')
            MyWindow.document.write('<\/body>')
            MyWindow.document.write('<\/html>')
            MyWindow.focus()
        }
        -->
    </script>
</head>
<body>
    <FORM action="http://www.jimkeogh.com" method="post">
        <P>
            <INPUT name="OpenWindow" value="Open Window"
                type="button" onclick="Window()" />
```



```
        </P>
    </FORM>
</body>
</html>
```

Creating dynamic content in this way is possible only if you “own” the new window and its contents. For example, you can’t load a web URL, such as `www.cnn.com`, and write to or read any content within it, because this is a security violation and the browser won’t allow it. You also can’t place content from a window in one domain to a window in another domain. If two windows are located in different domains, you must use JavaScript to set them to the same domain before they can communicate in this manner.

Looking Ahead

You can open a new window by calling the `window.open()` method from within your JavaScript. The `window.open()` method causes the browser to open a new window on the screen. You don’t need to pass the `window.open()` method any parameters if you want to use the standard windows settings and position as determined by the browser.

However, you can specify the size and the style of the window by passing the `window.open()` method the appropriate parameters. The `window.open()` method accepts three parameters: a reference to the content of the new window, the name of the new window, and a string that sets various window styles that include the size and position of the window.

The position of the window can be set explicitly by specifying the pixel coordinates for the upper-left corner of the window. Some JavaScript developers set the upper-left corner of the new window relative to the resolution of the screen by adding or subtracting pixels from the `screen.width` and `screen.height` parameters.

After opening a new window, you can use the `document.write()` method to write HTML tags and text to the new window, enabling you to use JavaScript to create dynamic content for windows—but only if the windows are in the same domain.

Now that you have a good understanding of how to create new windows and dynamic content for those windows, it’s time to learn a powerful tool that JavaScript developers use to validate information that is provided by visitors to their web sites. You’ll learn about *regular expressions* in Chapter 10.