Unit 4. Cookies and Browser Data

4.1. Cookies -

- A cookie is a small piece of information that a web site writes to our hard disk when we visit the site.
- Some site visitors may think that a cookie contains secret information used to spy on them or that the information is used to take over their computer when they least expect it.
- In reality, a cookie is plain text that can be used for a variety of purposes, but it's not intended to spy on you (though some web sites do track your visits to the site) and it definitely will not take over your computer.
- And because of the type of information contained in a cookie, it cannot give your computer a virus.

4.1.1. Basics of cookies

- A cookie is written to our hard disk by the browser when told to do so by a JavaScript. The developer, determine the contents of the cookie's plain text based on the nature of our JavaScript application.
- Some developers store user ID and password data to a cookie after a user successfully logs on to their web site. The cookie is then used for subsequent logons. Other developers use a cookie to store the date of the last time the user visited the website.
- Cookies can be used in countless ways and are limited only by your imagination and any restrictions placed by the browser.
- 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 ScubaBob is the value.
- Here's how this namevalue would be stored in the cookie:

```
userid='ScubaBob'
```

- *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.
- Cookies come in two flavors: session cookies and persistent cookies.
  1. 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. Even if the user surfs to another web site, the cookie remains in memory.
     - However, the cookie is automatically deleted when the user exits the browser application.
  2. A persistent cookie is a cookie that is assigned an expiration date.
     - 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.
- Each cookie contains the address of the server that created it. That means that only a web page from your server can read your cookie, and the browser prohibits a JavaScript from another server from reading the cookie.
- As a result, you won't be able to read a cookie that was written by another JavaScript application, and another JavaScript application cannot read your cookies.
- we can extend the life of a cookie by setting an expiration date, which becomes part of the cookie when the cookie is written to the user's hard disk.
- Note that information contained in a cookie identifies the computer that was used to visit your web site, not the person who used the computer to visit your site.
- You've probably noticed this if you and another person use the same computer to order books from an online bookstore.
- The cookie created by the online bookstore contains information about the last purchase. When you access the site, the online bookstore assumes that the person who's visiting the site is the same person who made the last purchase.
- It then uses the cookie to customize the web page by recommending titles based on the last purchase, unless the specific user logs on to the website using an ID and password.

- In that case, the cookie in the user's profile is used, and the tracking is done not by computer but by individual user.
- Furthermore, browser software will usually not retain more than 20 cookies per web server. This means that you are limited to 20 cookies stored on your hard drive, although some browsers might be able to store more than 20.

4.1.2 Reading a cookie value

- Reading a cookie is just as simple as writing one, because the value of the window.document.cookie object is the cookie.
- When the browser sees the window.document.cookie statement within a JavaScript, the browser copies the cookie to the window.document.cookie object.
- We use window.document.cookie whenever we want to access the cookie.
- The following example shows how to write JavaScript that reads a cookie.
- Here, form named CookieReader is displayed that contains two elements:
  a text box that will contain the value of the cookie and a button that, when clicked, executes the ReadCookie() function, which reads the cookie.

```
<! DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>Read Cookie</title>
 <script language="Javascript" type="text/javascript">
 <!--
 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>
```

- The ReadCookie() function begins with a with statement that contains other statements that are necessary to read and display the cookie.
  The first statement in the with statement determines whether any cookies exist by comparing the value of the cookie object to "", which is another way of saying NULL—or nothing.
- If no cookie is found, the value of the cookiecontent text box is set to No Cookies; otherwise, a cookie exists and the browser assigns the cookie object the name-pair value for the cookie.

- You probably noticed something strange within the statement that causes the cookie to be read: split('=')[1].
- The document.cookie is assigned the cookie by the browser.
- The cookie is plain text, which is a string. The split() is a string method that divides the string into an array that consists of two elements based on the character passed to the split() method.
- In this case, the split() method is being told to find the = character in the cookie, and then take all the characters to the left of the = and store them into array element [0].
- Next the split() method takes all the characters from the right of the = up to but not including the semicolon, and assign those characters to array element [1].
- It then takes everything up to the next =, including the semicolon.
- The semicolon separates cookies and the equal sign separates the name of the cookie with the cookie's value. You need to split at the semicolon, and then split on = to get all the values.
- Here's the cookie:

  "CustomerName=ScubaBob;"
- The split() function divides the text of the cookie into the following:

  Array[0] = "CustomerName"

  Array[1] = "ScubaBob"
- This statement assigns the value of Array[1] to the value of the cookiecontent text box on the form. The result is that ScubaBob is displayed in the text box, assuming that ScubaBob is the value of the cookie.

4.1.3 Writing a cookie value

See creating a cookie…..

4.1.4 Creating a cookies

- Creating a cookie is a easy. simply assign the cookie to the window.document.cookie object.
- The browser automatically writes the cookie to memory when it reads this assignment statement in our JavaScript, unless you set an expiration date for the cookie, which then causes the cookie to be written to the computer's hard disk.
- 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 ABC is the value: window.document.cookie = "CustomerName= ABC;"
- The web page in this example displays a form that contains an input for the customer's name, which is the only element that appears on the form.
- The user is prompted to enter a name, which then becomes the value of the cookie.
- The WriteCookie() JavaScript function is executed when the value of the element changes.
- The WriteCookie() function contains a statement that tells the browser to write the cookie to the hard disk.
- The function begins with a with statement that contains two statements: The first statement causes the cookie to be written.
- The name is CustomerName and the value of the cookie is the value of the customer element of the form, which is the name the person entered into the form.
- *Notice that the addition operator (+) is used to concatenate portions of the string to form the plain text of the cookie.*
- The second statement in the with statement causes an alert dialog box to be displayed, indicating that the cookie was written.
- You can exclude this statement in your JavaScript application because it is unnecessary to display anything when writing a cookie.
- This statement was included here simply to tell you when the cookie was written.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Write Cookie</title>
<script language="Javascript" type="text/javascript">
<!--
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>
```

4.1.5 Deleting a cookies

- Cookies are automatically deleted when either the browser session ends or its expiration date has been reached.
- However, you can remove a cookie at any time by setting its expiration date to a date previous to the current date. This forces the browser to delete the cookie.
- The most efficient way to reset the expiration date is to use the getDate() method of the Date variable, then subtract 1 from the date returned by this method, and then assign the difference to the Date variable.
- Here's how this is done. Assume that the expire variable is a Date variable.
- 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 expireDate variable is then converted to a string and concatenated to the cookie string, which is then written to the hard disk by the browser.
  expireDate.setDate(expireDate.getDate()-1)
- The following example demonstrates how to delete a cookie. It begins by displaying a form that contains a button.
- When the button is clicked, the JavaScript DeleteCookie() function executes by calculating the new date and passing the expireDate variable to the Date variable.
- The new expiration date is assigned to the cookie string. The browser then writes the cookie, notices that the date is expired, and deletes the cookie.

```
<! DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

 <title>Delete Cookie</title>

 <script language="Javascript" type="text/javascript">

 <!--
```

```
function DeleteCookie()
{
expireDate= new Date
expireDate.setDate(expireDate.getDate()-1)
with (document.CookieWriter)
{
var CustomerName = customer.value
document.cookie = "CustomerName1="+ CustomerName+";
expires="+expireDate.toGMTString()
alert("Cookie Deleted")
}
}
-->
</script>
</head>
<body>
<form name="CookieWriter" action="" >
Enter your name: <input type="text" name="customer" />
<INPUT name = "Reset" value = "Delete Cookie" type = "button" onclick = "DeleteCookie()" />
</FORM>
</body>
</html>
```

4.1.6 Setting the expiration date of cookie

- We can extend the life of a cookie beyond the current browser session by setting an expiration date and saving the expiration date within the cookie.
- The expiration date is typically an increment of the current date. For example, you might say that the cookie expires three months from the day the cookie was created.
- A date is stored in a variable of a Date data type.
- A Date variable contains a variety of methods that enable you to access various components of the date, such as month and year.
- For now, we'll concern ourselves with three of these methods.
- These are getMonth(), setMonth(), and toGMTString().
    1. The getMonth() method returns the current month based on the system clock of the computer running the JavaScript.
    2. The setMonth() method assigns the month to the Date variable.
    3. The toGMTString() method returns the value of the Date variable to a string that is in the format of Greenwich Mean Time, which is then assigned to the cookie.
- Let's set an expiration date for a cookie using these three methods.
- However, a few new statements in the WriteCookie() JavaScript function are used to create and write an expiration date.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>Write Cookie with Expiration Date</title>
 <script language="Javascript" type="text/javascript">
 <!--
 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")
}
}
-->
</script>
</head>
<body>
<form name="CookieWriter" action="" >
Enter your name: <input type="text" name="customer" onchange="WriteCookie()" />
</FORM>
</body>
</html>
```

- The first statement within the WriteCookie() function declares a variable called expireDate and assigns it a reference to a new Date data type. Only dates can be assigned to this variable.

- The second statement calls the getMonth() method to return the current month, which is then increased by three months. (So, for example, if the current month is May, the new month setting will be August.)
- The new month setting is passed to the setMonth() method, which sets the expiration date three months from the current date.
- The value of the expireDate value is then converted to a string in the GMT format by the toGMTString() method.
- Notice the statement that creates the cookie (document.cookie). Another name-pair value appears after the name-value pair of the cookie.
- This is the expires name-value pair, where expires is the name and the value is returned by the toGMTString() method.
- The browser then writes the entire string assigned to the document.cookie to the hard disk.
- The cookie will remain on the hard disk for three months, as long as the system clock on the computer isn't changed.

4.2. Browser Windows

- In this topic, you'll learn how to manipulate the browser window itself using a JavaScript.
- You can use JavaScript to open a new browser window while your JavaScript is running, to determine the size of the window, to determine whether or not the window has a toolbar or scroll bar, and to set up other styles that you've seen on many browsers windows.

4.2.1 Open the Window

- When we visit any web site pages then we click on button for next process or any other action to open new window.
- But some web sites don't even wait for us to do anything – Windows open "magically" When the web page loads or unloads. Of course, those windows display advertisements.
- The browser window is an object, similar to other objects.
- Whenever you want to do something with the browser window, you must reference a window and then reference the property or method of the window that you want to

access. For example, here's how to open an empty browser window that uses the default settings:

```
MyWindow = window.open()
```

- The `open()` method returns a reference to the new window, which is assigned to the `MyWindow` variable. You then use this reference any time that you want to do something with the window while your JavaScript runs.
- A window has many properties, such as its width, height, content, and name—to mention a few. You set these attributes when you create the window by passing them as parameters to the `open()` method:
    - The first parameter is the full or relative URL of the web page that will appear in the new window.
    - The second parameter is the name that you assign to the window.
    - The third parameter is a string that contains the style of the window. Table 9-1 shows a list of styles that you can set.

Sr. No.	Style	Description	Values (on=1, off=0)
1	status	The status bar	status=1, status=0
2	toolbar	The standard browser toolbar	Toolbar =1, toolbar = 0
3	location	The Location entry field	location=1, location=0
4	menubar	The menu bar	menubar=1, menubar=0
5	directories	The standard browser directory buttons	directories=1, directories =0
6	resizable	Allow/disallow the window to be resized	resizable=1, resizable=0
7	scrollbars	Enable the scrollbars	scrollbars=1, scrollbars=0
8	height	The height of the window in pixels	height=250
9	width	The width of the window in pixels	width=250

- Let's see, that we have to open a new window that has a height and a width of 250 pixels and displays an advertisement that is an image. All other styles are turned off. Here's how you'd do this:

  MyWindow =window.open('MyWebSite/MyAd.jpg','myAdWin', 'status=0, toolbar=0,location=0,menubar=0, directories=0,resizable=0, height=250, width=250')

- In this example, a web page is displayed in a new window(Figure 9-1).

```
<! DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>Open New Window</title>
 <script language="JavaScript" type="text/javascript">
 <!--
 function OpenNewWindow()
{
 MyWindow = window.open( 'MyWebSite/MyAd.jpg', 'myAdWin', 'status = 0, toolbar = 0, location = 0, menubar =0, directories =0, resizable = 0, height =250, width = 250')
 }
 -->
 </script>
</head>
 <body>
 <FORM action="http://www.jimkeogh.com" method="post">
 <P>
 <INPUT name = "OpenWindow" value ="Open Window" type= "button" onclick = "OpenNewWindow()" />
 </P>
 </FORM>
 </body>
```

```
</html>
```

Output :

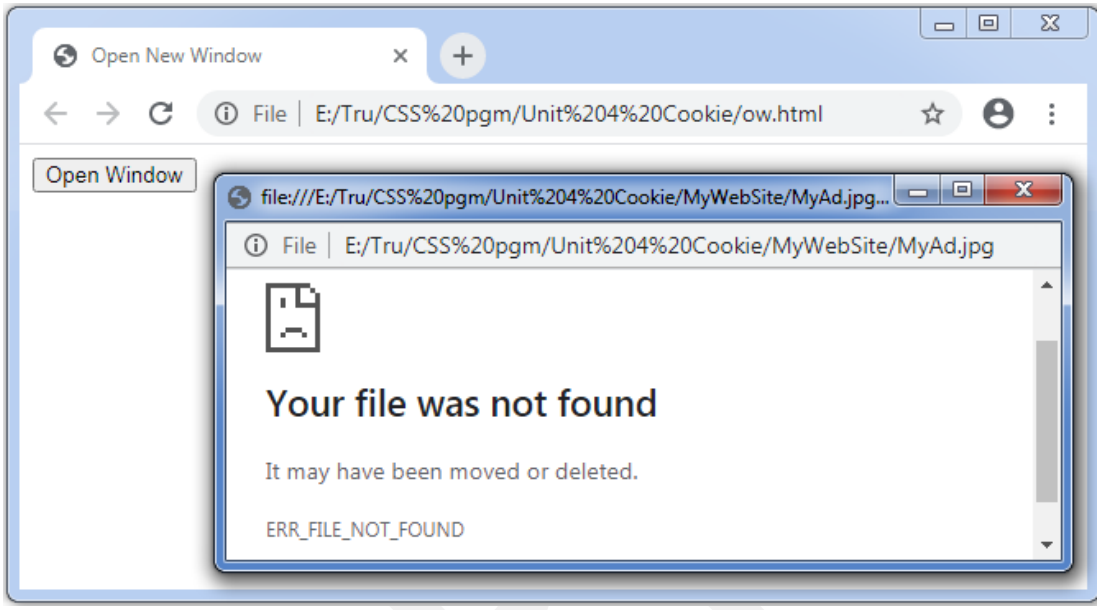


Fig. 9.1 A new window is opened by calling the `open()` method of the window object.

4.2.2 Giving the new window focus

- Usually, only one window is displayed when you visit a web site, although some sites display multiple windows filled with ads.
- The traditional web site displays the initial web page in a window and gives that window focus automatically.
- This means that anything you type or click affects the window that has focus—that is, the window that appears up front on the screen.
- The most recently opened window—that is, the last window opened—usually has focus by default.
- You give a new window focus by calling the focus() method of the new window after the new window opens. As shown next, the MyWindow variable receives a reference to the new window when window.open() is called:

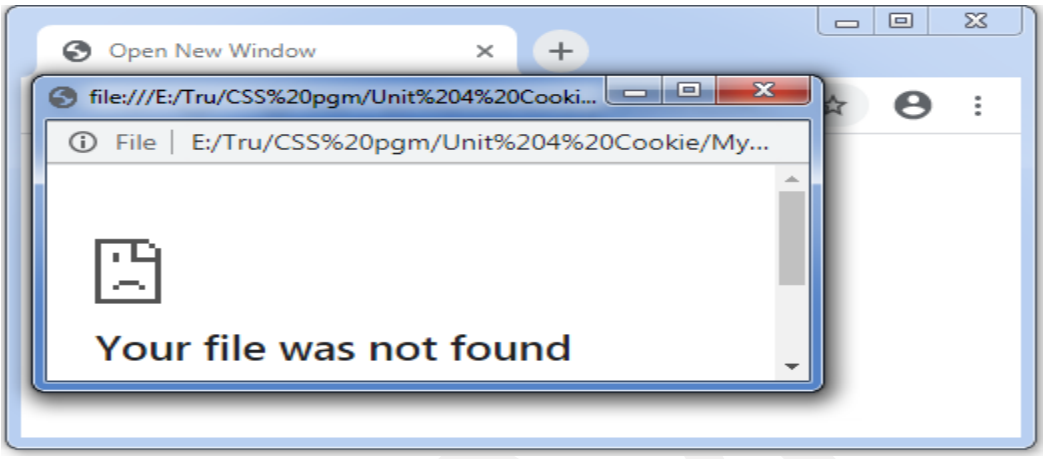
```
MyWindow.focus()
```

- Now our example opens a new window but gives the first open window focus.
- This is known as a pop-down window or a pop-back window. Any keystrokes the user makes will affect the first open window and not the new window.
- This can be an annoying web site feature, because it's contrary to the way window focus usually works; plus, the visitor may not even be aware of the first open window because it is obscured by the second open window.

```
<! DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>Open New Window</title>
 <script language="Javascript" type="text/javascript">
 <!--
 function OpenNewWindow()
{
 MyWindow = window.open ( 'MyWebSite/MyAd.jpg', 'myAdWin', 'status = 0, toolbar = 0, location = 0, menubar = 0, directories = 0, resizable = 0, height = 250, width = 250')
 this.focus()
 }
 -->
 </script>
</head>
 <body>
 <FORM action="http://www.jimkeogh.com" method="post">
 <P>
 <INPUT name = "OpenWindow" value = "Open Window" type = "button" onclick = "OpenNewWindow()" />
 </P>
 </FORM>
 </body>
```

```
</html>
```

Output :



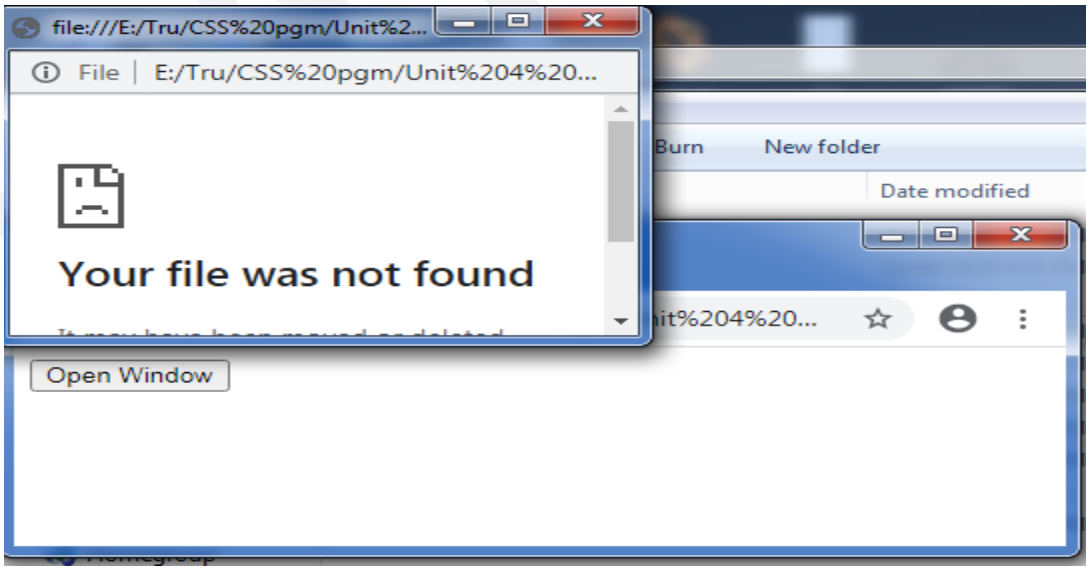
4.2.3 Window position

- The browser determines the location on the screen where a new window will be displayed; however, you 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.
- The following example shows how to position a new window in the upper-left corner of the screen by setting the left property to 0 and the top property to 0(Figure 9-2).
- This example displays an Open Window button on the screen. A new window is created on top of the current window after the button is clicked.

```
<! DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
 <title>Position New Window</title>\
 <script language="Javascript" type="text/javascript">
 <!--
 function OpenNewWindow()
 {
```

```
 MyWindow = window.open('MyWebSite/MyAd.jpg',‘myAdWin','width = 250, height = 250, left = 0, top = 0')
 }
 -->
 </script>
 </head>
 <body>
 <FORM action="http://www.jimkeogh.com" method="post">
 <P>
 <INPUT name = "OpenWindow" value = "Open Window" type ="button" onclick = "OpenNewWindow()"/>
 </P>
 </FORM>
 </body>
</html>
```

Output :



- The Screen resolution (the number of pixels that appear on the screen) settings differ from computer to computer, when you specify pixel locations while positioning a new

window on the screen at your computer, you may set left and top properties that will appear differently on other users' computers.

- Some computers use a higher (more pixels) or lower (less pixels) screen resolution than you use on your computer.
- The pixel settings that you specify for the position of your new window will appear differently if a user's screen is set at a resolution that differs from yours.
- Let's say, for example, that you want the upper-left corner of your new window to appear at pixel 160—that is, 160 pixels from the left edge of the screen.
- If the resolution of your screen is 640 pixels wide, then the left corner of the new window appears about a quarter of the way across the screen.
- However, if the resolution of another user's screen is 1024 pixels wide, then the left corner of the new window appears about 15 percent of the way across the screen.
- This difference in where the window appears might be meaningful to the presentation of your web page, depending on your application, so it's important that you try to account for its placement on different computer screens.

Sr. No.	Properties	Description
1	availHeight	Returns the height of the available screen in pixels
2	availWidth	Returns the width of the available screen in pixels
3	colorDepth	Returns the bit depth if a color palette is used
4	height	Returns the height of the display screen
5	pixelDepth	Returns the color resolution as bits per pixel
6	width	Returns the width of the display screen.

Table9.2 : Properties of Screen Object

- Table 9-2 lists the properties that are available to the screen object.

- The two properties used to set the relative position of the left and top properties of the window are the screen.width and screen.height properties.
- These properties contain the number of pixels across (the x value) and down (the y value) the screen, respectively. By knowing this information, you can add or subtract pixels from these values to set the left and top properties of the window respective to the screen resolution.
- The amount that you add or subtract depends on the size of your window and where you want to position the window on the screen.

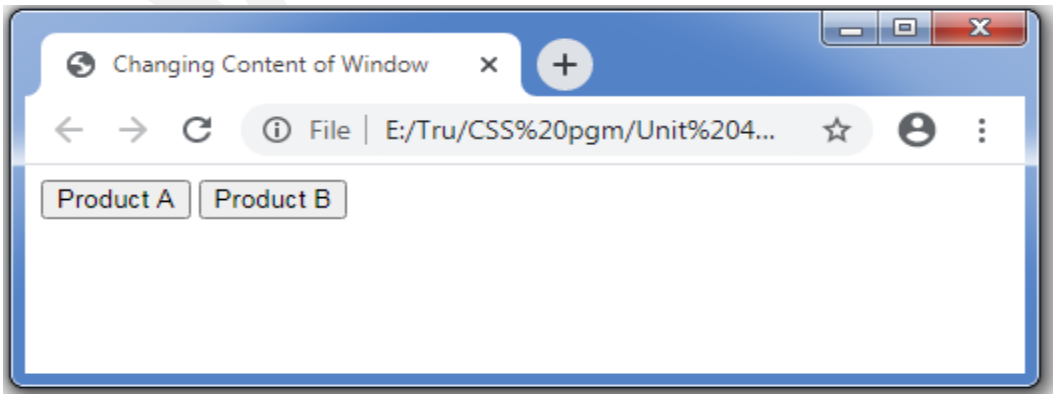
4.2.4 Changing the content of window

- The secret to changing the content of a window is to call the open() method using the same window name each time you change the content of the window.
- Suppose that the window is called MyWindow. The first time a customer selects an item, you open a new window, calling it MyWindow and displaying the appropriate product in the window.
- The next time a customer selects an item, you again open a new window, calling it MyWindow 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.
- The following example shows that, Two buttons appear on this page; each button displays an advertisement in a new window by calling the OpenNewWindow() JavaScript function and passing reference to the advertisement to the function.
- The OpenNewWindow() function is defined in the <head>tag, the content of the new window is passed as a parameter to the function, which enables you to change the content of the window each time the function is called.

```
<! 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 OpenNewWindow(Ad)
{
MyWindow = window.open(Ad, 'myAdWin', 'status = 0, toolbar = 0, location =
0, menubar = 0, directories = 0, resizable = 0, height = 250, width = 250')
}
-->
</script>
</head>
<body>
<FORM action="http://www.jimkeogh.com" 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>
```

Output :



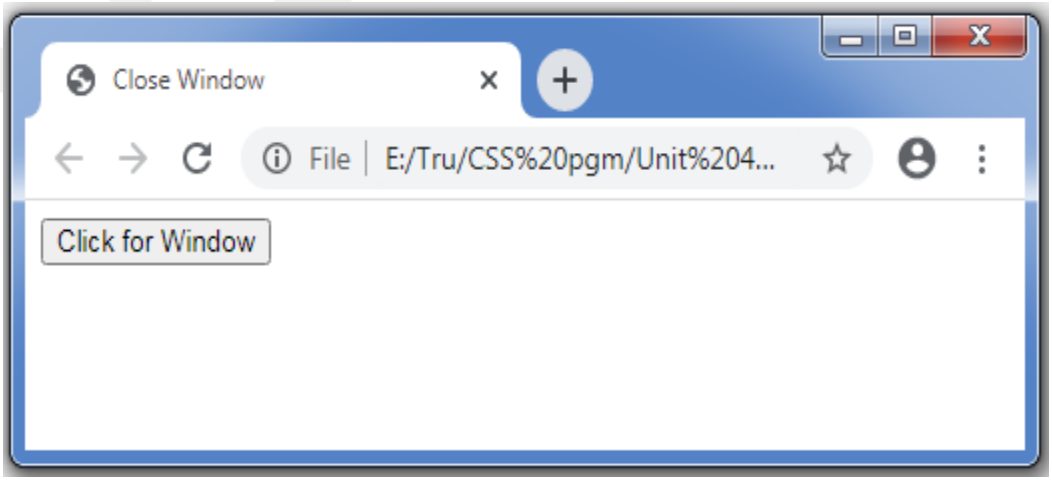
4.2.5 Closing a window

- we can close any window that you open by calling the window's close() method from within your JavaScript.
- As we know, the open() method returns a reference to the newly opened window, which is a window object.
- we use the reference to call the close() method. This tells the browser which window you want to close.
- The following example shows how to use the close() method. One button is used both to open and close the window (Figure 9-3).
- The button is labeled "Click for Window" at first.
- The Window() JavaScript function is called when the button is clicked; this function is defined in the <head> tag.

```
<! DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>Close Window</title>
 <script language="Javascript" type="text/javascript">
 <!--
 var WindowStatus
 function Window()
{
 if (WindowStatus != '1')
 {
 MyWindow = window.open('MyWebSite/MyAd1.jpg', 'myAdWin',
'status = 0, toolbar = 0, location = 0, menubar = 0, directories = 0,
 resizable = 0, height = 250, width = 250')
WindowStatus ='1'
 }
 else
```

```
{
MyWindow.close()
WindowStatus = '0'
}
}
-->
</script>
</head>
<body>
<FORM action="http://www.jimkeogh.com" method="post">
<P>
<INPUT name = "OpenWindow" value = "Click for Window"
type = "button" onclick = "Window()" />
</P>
</FORM>
</body>
</html>
```

Output :



- Here, we declare a variable called WindowStatus, which is used within the function to determine whether the window is opened or closed.

- When the function is called, the browser is told to determine whether the WindowStatus variable is a value other than 1. Since we didn't initialize this variable, its value is not 1, and therefore statements within the if code block are executed.
    i. The first statement opens a new window.
    ii. The second statement gives the new window focus.
    iii. The third statement assigns 1 to the WindowStatus variable, indicating that the window is opened.
- Basically, the same process occurs the next time the button is clicked. However, the value of the WindowStatus variable is 1. This means that statements within the if code block are skipped and statements within the else code block are executed.
- Two statements are included within the else code block.
    i. The first statement calls the close() method, which closes the new window.
    ii. The second statement resets the value of the WindowStatus variable to 0. This indicates that the window is closed.

4.2.6 Scrolling a web page

- In some web sites, the web page "magically" scrolls to a section on the site. Actually, no magic is involved; instead, a JavaScript is used to scroll the web page automatically by calling the scrollTo() method of the window object, or a link directly to a relative link in the page.
- The scrollTo() method requires two parameters, which are the x and y coordinates of the top-left corner of the viewable area of the web page that you want to display.
- Each parameter is an integer and represents the coordinate in pixels.
- A button is displayed in this example that, when selected, calls the Top() JavaScript function to scroll to the top of the web page.
- This function, which is defined in the <head> tag, calls the scrollTo() function, passing it coordinate 0,0—the upper-left corner of the web page, which means that the top of the web page is displayed.
- *Notice that self is used to reference the window when calling the scrollTo() function.* This refers to the window that contains the button.

- You could replace self with a reference to another window that you opened, which would cause the other window to scroll when the button is clicked.

```
<! DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>Scrolling</title>
 <script language="Javascript" type="text/javascript">
 <!--
 function Top()
{
 self.scrollTo(0,0)
 }
 -->
 </script>
</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>
```

Output :

4.2.7 Multiple windows at once

- New windows pop up all over the screen.
- Anyway, here, we see 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).
- 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>
```

Output :

4.2.8 Creating a web page in 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.
- 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.
- 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.

```
<!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>
```

Output :

- 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.