PIC 40A

Lecture 15: JS: More events, JSEE, Cookies

JS events

Events are what JavaScript was created for

They make your scripts interactive

When a user does something an event takes place

Sometimes events are browser created

Event-driven programming focuses on detecting and responding to events

Typical events

User clicks a button

Mouse hovers over an element

Page finishes loading

How do I respond to an event?

1. Create an *event handler*, a script that is designed to respond to a particular event

2. Connect the handler to the event (*registration*)

Two approaches to handler registration

- Assign an appropriate attribute to some tag.
- Assign a handler address to a DOM object property

Registration Example 1

Assign a tag attribute:

```
<head>
   <title>Event Handler Example</title>
   <script type="text/javascript">
   <!--
      function load_greeting()
      { alert("whatup!");}
   // -->
   </script>
</head>

<body onload="load_greeting()">
</body>
```

Registration Example 2

In a head JavaScript script

```
<script type="text/javascript">
<!--
    function message() {alert("Thanks for clicking");}
// -->
</script>
```

In the body

```
<form id="messageBoard" action="">
    <input type="button" name="mybutton" id="mybutton"
                                value="See a message" />
</form>
<script type="text/javascript">
<!--
    document.getElementById("mybutton").onclick = message;
// -->
</script>
```

Some remarks on method 2)

Cannot specify parameters this way

But does keep XHTML and JavaScript separate in the document (modular, cleaner design)

Allows the handler property to be changed during use (when some other event occurred)

Events for forms

- `onclick`
  - The user clicks the element
- `onchange`
  - The element is changed and loses focus
- `onfocus`
  - The element gains focus
- `onblur`
  - The element loses focus

Events

- `onload`
  - document finishes loading
- `onunload`
  - user exits the doc
  - good for last minute cleanup
- `onselect`
  - user selects text in a text area
  - user moves mouse cursor over input element

Events

- `onmouseup`

–mouse button is released

- `onmousedown`

–mouse button is pressed

- `onmouseover`

–mouse is moved over an element

- `onmouseout`

–mouse is moved off an element

- `onmousemove`

–mouse is moved

Acessing events

Assume I have the following:

```
element.onclick = handler;

function handler(e)
{
// e gives access to the event.

alert ("Event was of type" + e.type);

}
```

Some useful event properties are:

type – What is the event type.
target – The node object associated with the event.
pageX, pageY – X,Y coordinates of the mouse

Note for explorer you need to do:

```
if (!e)
var e = window.event;
```

Acessing element that triggered an event

In JS the keyword `this` refers to the owner of the function we are executing:

```
function doSomething() {
   this.style.color = 'blue';
}

element.onclick = doSomething;
```

Now `this` refers to the element.

However, this does not work for onclick attributes...

Acessing element that triggered an event

To do this for attributes you either have to use the `e.target` method I described earlier or:

```
<a href="#" onclick="dosomething(this)">Click here</a>
```

With a corresponding handler function

```
function dosomething(object)
{

// Stuff

}
```

Can I Get a Cookie?

Originally invented by Netscape to give memory to web browsers

Cookie is a small text file saved on your computer

Consists of:

1. Name-value pair containing the stored data
2. Expiration date of the cookie
3. The domain and path of the server the cookie is sent to

Cookies are used to store information about a visit to a web page. This could include:

User preferences, password information, date information, information used by the server to track how you use their web pages

Creating Cookies

Cookies can be created, read and erased by JS

Accessible through the property `document.cookie`

Setting a cookie

```
document.cookie = "Some string"
```

The some string could be anything. However, there is a standard way of writing cookies so that they are useful.

Creating Cookies

Syntax for setting a cookie

```
document.cookie = "name=value; expires=date;"
```

- `name` is the identifier for a cookie.
- `value` is what we want to store.
- `expires` sets the expiration date.

We will only sometimes use `expires`

`expires` must be in GMT date format
You can use `toGMTString()` method of Date object

If no expires is given the cookie will expire when you close the browser

Example:

```
document.cookie = "my_cookie = Chocolate chip;
expires = Thu, 18 Nov 2010 12:00:00 UTC";
```

document.cookie is now a string:

```
my_cookie = Chocolate chip;
```

```
Note: The expires part will not show up if you do
something like:
```

```
alert(document.cookie);
```

```
or
```

```
var my_cookie_string = document.cookie;
```

Example:

We could have done:

```
var cookiedate = new Date( 2011, 10, 24, 12);
document.cookie = "my_cookie = Chocolate chip;
expires = " + cookiedate.toGMTString();
```

Example

```
document.cookie = "my_cookie = Chocolate chip;
expires = " + cookiedate.toGMTString();

document.cookie = "visitors=Fry,Leela,Bender;
expires = " + cookiedate.toGMTString();
```

Note: `document.cookie` is not overwritten it simply contains these two cookies now

The full cookie string looks something like:

```
my_cookie = Chocolate chip;
visitors=Fry,Leela,Bender;
```

Space

Example

To modify already existing cookie simply redefine it.

```
document.cookie = "my_cookie = Chocolate chip; expires
= " + cookiedate.toGMTString();

document.cookie = "my_cookie = Oatmeal raisin;
expires = " + cookiedate.toGMTString();
```

Retrieving Stored Cookies

Cookie name value pairs are delimited by semicolons

To get cookies you can do

```
var cookies = document.cookie;
```

This returns a string of the name value pairs delimited by ;
split method of strings is useful in dealing with cookies

Retrieving stored cookies

Example:

Assume we have a cookie string consisting of name-value pairs.

```
var cookie_array =
document.cookie.split(";");
```

To get the first cookie

```
var cookie = cookie_array[0];
```

`cookie` is now a string consisting of `cookie_name=value`

```
cookie_data_array = cookie.split("=");
```

Retrieving stored cookies

If we have more general cookies with expires values things are slightly more complicated.

```
function readCookie(name)
{
    var nameEQ = name + "=";
    var ca = document.cookie.split(';');
    for (var i=0; i < ca.length; i++)
        {
        var c = ca[i];
        while (c.charAt(0)==' ')
                    c = c.substring(1,c.length-1);
           if (c.indexOf(nameEQ) == 0)
            return c.substring(nameEQ.length,c.length-1);
        }
          return null;
}
```

To think about at home

Fine, but what happens if you do not know what the names of the cookies are.

All you know is that the format of the cookie is

```
name1=value1;name2=value2;...
```

Deleting cookies

To delete a cookie set its expiration to 1 second in the past

Example:

```
function delete_cookie(cookie_name)
{

   var cookie_date = new Date();
   cookie_date.setTime(cookie_date.getTime()-1);
   document.cookie = cookie_name +=  "=; expires="
             + cookie_date.toGMTString();

}
```

JavaScript Execution Environment (JSEE)

The highest level of object hierarchy under which JS operates

`Window` object
`Document` object

`Window` object

Represents the window of the client browser that displays the XHTML doc

Its properties are visible to all JavaScript scripts that appear in its doc

Contains all the global variables as properties

`Window` Object

Provides the largest enclosing referencing environment for JavaScript scripts

Can be more than one of them.

A variable declared in one `Window` object is not a global variable in another.

Some `Window` Object properties

• `closed`	a Boolean value
• `parent`	a reference to its parent window
• `document`	a reference to the `document` object the client browser window displays. (See document object slides)
• `opener`	reference to the `window` object that opened it (if it exists)
• `location`	location object (see additional slides on location)
• `history`	History object
• `outerHeight` • `outerWidth`	Set the dimensions of a window

Some `Window` Methods

- `alert(`*`message`*`)`
- `confirm(`*`question`*`)`
- `prompt(`*`prompt`*`, `*`default`*`)`
- `blur() //Removes focus from window`
- `focus() // Sets focus to window`
- `open(`*`url`*`, `*`name`*`, `*`options`*`)`
- `close()`

`Document` Object

Represents the XHTML document displayed by a client browser

Used to access all XHTML document elements as objects

Some `Document` Properties

Special arrays of DOM objects

`anchors` (the `<a>` elements)
`forms` (the `<form>` elements)
`images` (the `<img>` elements)
`links` (the `<link>` elements)

Example:

```
document.write("The number of anchor elements
                  in this document is: ");
document.write(document.anchors.length);
```

More `Document` Properties

- `body` (DOM address for `body` element)
- `cookie` (sets or returns all doc cookies)
- `domain` (the doc server's domain name)
- `referrer` (returns URL of doc that loaded current doc)
- `title` (title of the document)
- `URL` (URL of the document)

Some `Document` Methods

- `open()` (opens a doc stream for writing)
- `write()` (writes a string to opened doc)
- `writeln()` (writes, appends a newline)
- `close()` (close the doc stream and display it)
- `getElementById()` (a single element)
- `getElementsByName()` (array of elts)
- `getElementsByTagName()` (array of elts)

The `navigator` Object

- Indicates which browser is being used to view the XHTML document
- Useful for handling browser incompatibility issues

Some `navigator` properties

- `appName` — the name of the browser
- `appVersion` — the platform and version of the browser
- `platform` — returns the operating system platform
- `cookieEnabled` — boolean
- `plugins[]` — a reference to all embedded objects in the document

Navigator example

The `location` Object

Contains information about the current URL

Used to load a new document in the current browser window:
–just assign a URL to the `window` object's `location` property

Example:
```
window.location= "http://www.pic.ucla.edu";
```

Some `location` properties

- `host` the host name and port number of the current URL
- `href` the entire URL
- `search` the URL from the question mark
- `port` port number of the current URL
- `protocol` protocol of the current URL

Some `location` methods

- `assign()` loads a new document

```
location.assign("www.pic.ucla.edu")
```

- `reload()` reloads the current doc

```
location.reload();
```

- `replace()` replaces current doc with a new one

```
location.replace("www.pic.ucla.edu")
```

Difference between assign and replace is that the with replace the current doc is removed from history

Some `location` methods

- `assign()` loads a new document

```
location.assign("www.pic.ucla.edu")
```

- `reload()` reloads the current doc

```
location.reload();
```

- `replace()` replaces current doc with a new one

```
location.replace("www.pic.ucla.edu")
```

Difference between assign and replace is that the with replace the current doc is removed from history