Software Craftsmanship Guild

526 S. Main St, Suite 609

Akron, OH 44311

Browser Object Model

Software Craftsmanship Guild

What is the BOM?

The Browser Object Model (BOM) provides objects that expose common browser functionality.

Back in the day, every browser vendor implemented the BOM however they saw fit, but over time commonalities have become defacto-standards. The HTML5 specification now covers major aspects of the BOM.

So it's getting better… this used to be a major source of frustration in web programming.

The window Object

The core of what happens in the browser object model is tied to the window object, which not only represents an instance of a browser, but is also the global scope.

So every object, variable, and function defined in a web page uses *window* as its global object.

This is where common methods like parseInt() are attached.

Window as a Global Scope

Here we declare a variable and function in the global scope, which places them on the window object.

* Note that you can not delete global variables unless they are directly defined on window.

```
var name = "Bob";

function sayName() {
    alert(this.name);
}

alert(window.name); //Bob
sayName(); //Bob
window.sayName(); //Bob
```

Opening Windows

We can open new windows using window.open(). Be aware that this is the standard popup that most browser block by default nowadays.

The first parameter is the URL, second is the target, third is the options as key value pairs.

```
<script>
    window.open("http://www.swcguild.com", "_blank", "height=400,width=400,top=10,left=10,resizable=yes");
</script>
```

Checking For Popup Blocking

To see if an opened window was blocked, just assign the open function to a variable and check for null:

```
var swc = window.open("http://www.swcguild.com", "_blank");
if (swc == null) {
    alert("The popup was blocked!");
}
```

Other Useful Window Functions

- resizeTo(x,y)- resizes a window to x by y pixels
- moveTo(x,y)- moves window to the coordinates, top left being 0,0
- close()- closes a window

Timeouts

A timeout allows us to execute code after a certain amount of time has elapsed (milliseconds).

setTimeout(function, milliseconds) sets up a timeout and clearTimeout() cancels it.

```
//set the timeout
var timeoutId = setTimeout(function() {
    alert("Time's up!");
}, 1000);

// never mind - cancel it
clearTimeout(timeoutId);
```

Intervals

Where a timeout only runs once, intervals will run continuously until cleared. setInterval() and clearInterval() handle this.

Let's say you wanted to run something 10 times every half second:

You can achieve the same effect using recursive timeouts.

```
var num = 0;
var max = 10;
var intervalId = null;

function incrementNumber() {
    num++;
    //stop interval when max is reached
    if (num == max) {
        clearInterval(intervalId);
        alert("Done");
    }
}

intervalId = setInterval(incrementNumber, 500);
```

System Dialogues

We have already seen alert, but there are several other System Dialogues:

- **confirm(prompt) -** can be used inside an if statement, it shows an ok/cancel prompt.
- **prompt(question, defaultAnswer) -** shows an entry dialogue box
- **find()** – shows the browser search box
- **print()** – shows the browser print dialogue

window.location

The location object has properties useful for parsing URL information

Property	Description
host	Name of server and port number if present
hostname	Server name with no port number
href	Full URL of the current loaded page
pathname	Directory and filename of the page
port	port number only
protocol	typically http or https
search	The querystring of the URL, always starts with ?. Split on & then = to get querystring values

Changing Locations

We can set the location property in a few ways to move to a new URL. The most common way is to use assign(), but note that we can access the location property directly, since it is on the global window object.

```
window.location.assign("http://www.swcguild.com");

window.location = "http://www.swcguild.com";

location.href = "http://www.swcguild.com";
```

location.replace()

The previous location modifiers will put an entry in the browser history. However if you use location.replace(url) it will not make an entry in the history stack, so the back button will not work!

location.reload()

Reloading a page has two versions:

- reload() refreshes a page, but possibly from cache
- reload(true) forces a trip to the server

Browser History

We can also navigate through the browser's history (assuming it exists).

It's typically used to create custom back and forward buttons, but it's generally frowned upon to make heavy use of it since you can violate the principle of least surprise.

```
if (history.length == 0) {
    //this is the first page in the user's window
}

//go back one page
history.go(-1);

//go forward one page
history.go(1);

//go forward two pages
history.go(2);

//go to nearest espn.com page
history.go("espn.com");

//go to nearest swcguild.com page
history.go("swcguild.com");

//go back one page
history.back();

//go forward one page
history.forward();
```