David’s JavaScript Notes

…use them at your own risk

Last modified: 1st December 2002

Index

Preface

These notes were written as an *aide memoire* for the JavaScript code I use (or think I might have cause to use), and the comments are very much from the point of view of someone who is familiar with Java (sic) but not with scripting languages such as Perl, which have more in common with JavaScript. They assume familarity with the Document Object model and event handlers. If they are of interest to anyone other than myself I shall be surprised, but it costs me nothing to offer them over the internet and it would be nice to be able to give back a little. I might even get feedback correcting the inevitable mistakes they will contain.

Ammending this preface in 2002 I am aware that some of the specific issues that these notes address are no longer relevant. Nobody, as far as I am aware, is supporting anything earlier than Netscape 4 or Internet Explorer 5, and Javascript 1.1 can be taken as read. However, I hope they still may be of use.

Acknowledgments
The original *Babble* mailing list and especially its JavaScript stalwart, Porter Glendinning.
Flanagan, D. (1998) *JavaScript, The Definitive Guide — 3rd Ed* (O'Reilly).
Heinle, N. (1997) *Designing with JavaScript* (O'Reilly).

…and if you recognize your own code and I haven't acknowledged it, let me know and I'll do so.

David Leader
(Q7design@yahoo.co.uk)

Introduction: Manipulating Window Objects

JavaScript can be used to achieve dynamic effects in web browsers by manipulating certain window objects. To specify these objects you need to know the object hierarchy, which can be found in any book on JavaScript. You also need to be aware that generally there can be a collection of objects of any type, and that these can be referred to by an array number corresponding to their position of occurrence, or, more easily, a name. I summarize the ones that I have found to be most useful here.

The 'document' object

Position in hierarchy

```
(window).document
```

Typical use

The 'write()' method of the document object can be used to generate html content dynamically by a statements such as:

```
document.write("html code");
```

Dynamic use is obviously achieved by alternative 'write' statements for different conditions.

The 'location' object

Position in hierarchy

```
(window).location
```

Typical use

As the equivalent to a `<href>` link to another page. This would typically be used with dynamic generation of pages (i.e. without user control), or to change more than one frame of a frameset in response to a user selection. To replace the current html page the code might be:

```
self.location = 'anotherPage.html';
```

whereas to change a frame the code would have to refer back to the parent and use the name of the frame specified in the frameset (or its number), e.g.

```
parent.rightFrame.location = 'differentPage.html';
```

The 'images' object

Position in hierarchy

```
(window).document.images
```

Typical use

To change the bitmap graphic assigned to a particular image using the 'src' property of the images. In the example below the particular member of the images array is referred to by its name 'myImage':

```
document.myImage.src = 'pretty.gif';
```

The 'forms' object

Position in hierarchy

```
(window).document.forms
```

Typical use

To set the 'value' of a 'text' object (an input field). e.g.

```
document.myForm.Surname.value = 'Smith'
```

where 'myForm' is the name of the form and 'Surname' is the name of the field.

Client Detection: JavaScript and Java

A note on JavaScript version support

To a first approximation, JS 1.0 is not supported before NS2 and IE3, JS 1.1 is not supported before NS3 and IE4, and JS 1.2 is not supported before NS4 and IE4. This gives an idea of which browsers one will be supporting if one depends on a particular JS version, but one needs to check a reference source and should use object-, rather than browser-, detection in code.

Hiding JavaScript from browsers that do not support it

The strategy is to use html comment tags (<!-- and -->) within the javascript tags, but to add a javascript single-line comment tag (//) before the closing html comment tag:

```
<SCRIPT TYPE="text/javascript">
<!-- Start html block comment. Also JS single line comment
Code here is executed by JS-enabled browsers but ignored by others.
// Need JS comment tag in front of html end-comment tag-->
</SCRIPT>
```

Alerting users to the need to turn JavaScript on

Although I have never used this, it would seem that the <NOSCRIPT> html tag can be used to send a message that will also be seen by NS2. By specifying the javascript version you can do something like this:

```
<SCRIPT TYPE="text/javascript">
<!--
Code here is ignored by NS2, and by later browsers with JS disabled.
//-->
</SCRIPT>
<NOSCRIPT>
Either you are using Netscape 2 (or earlier) or Internet Explorer 3
(or earlier). If so, you will need to upgrade to view this page.
If you are using a later browser version, please turn JavaScript on.
</NOSCRIPT>
```

Detecting whether Java is enabled

One assumes that there are very few none java-capable browsers in use. These can, in any case be dealt with by including a message in the Java applet tag after any param tags.

```
<applet...>
    <param...>
    You need Java to view this applet.
</applet>
```

However it may be preferable to detect java-capable browsers with java not enabled. This can be done with the JavaScript 1.1 *javaEnabled()* method, which might be used in the following sort of way:

```
if (navigator.javaEnabled())
{code to be executed if java is enabled}
else
{code to be executed if java is not enabled}
```

Of course, someone who has *JavaScript* disabled will get neither response.

JavaScript 1.0 Workarounds

There are a few features absent from JavaScript 1.0 that can be achieved by a little trickery. As only version 2 browsers (NS2 and IE3) — now obsolete — need these tricks, they have been relegated to this section, where they are presented with minimal comment.

Arrays

```
    // Function to make psuedo-array of length len
    function makeArray(len)
    {
        for (var i=0; i<len; i++)
        this[i] =null;
        this.length = len;
    }

    // Make new pseudo-array of size needed (here number of images)
    var imgArray = new makeArray(imgNo);

    // Fill array, here with a set of images
    for (var i=0; i< imgNo ; i++)
    {
        imgArray[i] = prefix + (i +1) + ".jpg";
    }
```

Random Numbers

```
    // From Heinle
    var now = new Date();
    var seed = now.getTime() % 0xffffffff;

    // Generates a random number from 0 to (n-1)
    function rand(n)
    {
        seed = (0x015a4e35 * seed) % 0x7fffffff;
        return (seed >>16) %n;
    }
```

Client Detection: Browser, Version, Platform &c.

Purpose

To identify aspects of the user’s software and hardware platform so as to be able to handle situations in which ones code might break.

Browser

The `navigator.appName` string property can be used, e.g.

```
    if (navigator.appName == "Netscape")
    {code here}
    else if (navigator.appName == "Microsoft Internet Explorer")
    {other code here }
```

Version

The `navigator.appVersion` string property contains both this and platform information in the format:

```
    version (platform; encryptation[;detail])
```

The function returns strings like 4.08 (Macintosh; I; PPC, Nav) or 4.0 (compatible; MSIE 4.01; Windows 95), although on the Mac platform IE gives some weird information. To get the version number, use parseInt().

```
    var browVers = parseInt(navigator.appVersion);
    if (browVers >= 3)
    {code here}
```

It is easier to check for objects than for the combination of version and browser number, if this will satisfy your requirements (*see* below).

Platform

The platform information provided by `navigator.appVersion` is neither consistent nor reliable. MSIE 4.5 on the Mac reports itself as ‘compatible’, with the word Macintosh appearing further on in the string. An alternative, `navigator.platform`., which returns something like ‘MacPPC’ or ‘Win32’, is only available in JavaScript 1.2. However as the words ‘Win’ or ‘Mac’ should appear somewhere in the string, you can use something like:

```
    if (navigator.appVersion.indexOf("Mac") != -1)
    {do the Mac-specific thing}
```

(If the string ‘Mac’ is absent, its index is -1.)

Object detection

This is most used for detecting whether ‘rollover’ code will cause errors by checking whether the *Image* object exists with:

```
    if (document.images){rollover code here}
```

It is also used in relation to DHTML to detect whether the IE4 or NS4 models are supported:

```
    if (document.all){IE DHTML code here }
    if (document.layers){NS  DHTML code here }
```

Or whether one is dealing with v. 4 browsers:

```
    if (document.all || document.layers) {v.4 Browser code here }
```

However it should be noted that neither ‘`all`’ or ‘`layers`’ are included in the HTML 4 standard.

Client Detection: Monitor Size and Colour Depth

Purpose

To detect the user’s monitor characteristics so as to be able to serve appropriate pages or alert him to problems, I suppose.

The problem is that both the methods below use the ‘screen’ object, which was not introduced until JavaScript 1.2. Many of the users with the low-end monitors that you wish to detect are probably using versions of browsers that do not support the screen object. However, I present the code below out of interest.

Screen width

```
var w=screen.width;
if (w<=640)
{serve small-screen page}
else
{serve large-screen page}
```

Colour Depth

There appears to be some difference between the way the property screen.colorDepth works on IE and NS (which also has a property screen.pixelDepth, although varying the colour resolution on my Mac between 256 colours (8 bit), thousands (16 bit) and millions (32 bit) gave similar results with the code:

```
var cd = screen.colorDepth;
document.write(ColorDepth: " + cd);
```

which returned the bit values, 8, 16 and 32.

Screen Object detection

The code above should be enclosed in a screen-object detection ‘if’ statement:

```
if (window.screen){screen detection code here}
```

Frames: Maintaining or Breaking out of Frames

Breaking out of Frames

Purpose

To prevent one's page being linked to by someone else's page in a way that encloses it in one of their frames.

Code

```
1    if (top.frames.length!=0)
     {
2        top.location=self.document.location;
     }
```

Comments

`<HEAD>` location.

Explanation

1 This translates as "if the number of frames possessed by the top window is not 0"...
2 This translates as "make me the top window".
i.e. If I am in a frameset, put me in a window by myself.

Maintaining Frames

Purpose

To prevent a single frame of a frameset being opened in isolation.

Code

```
1    if (self.parent.frames.length == 0)
     {
2        self.parent.location="FramesetFrame.html ";
     }
```

Comments

- `<HEAD>` location.

Explanation

1 This translates as "if the number of frames possessed by the window above me *is* 0"...
2 This translates as "then give me as parent the document at the location specified".
i.e. If I am *not* in a frameset, put me in a frameset by making my parent the document specified. Of course, if the document specified does not define the frameset this will not work.

Frame cross-talk: Checking for loading

Purpose

To avoid possible JavaScript errors through attempts to access code or objects in other frames before they have loaded.

Head Code in target frame

```
1    var loaded = false;
```

Body Code in target frame

```
2    <BODY onLoad="loaded = true;" onUnload="loaded = false;">
```

Head Code in frame making call

```
     function myFunction()  // function called e.g. by an 'onClick'
     {
4        if (parent.targetFrame.loaded == false)
         {
5            setTimeout("myFunction", 5000);
         }
         else
6        {code aimed at 'targetFrame'}
     }
```

Body Code in frame making call (example)

```
3    <A HREF="#" onClick="myFunction();">linked image</A>
```

Explanation

1 The head of the target frame initializes a variable ‘loaded’ to ‘false’.

2 The html ‘onLoad’ event handler sets this to ‘true’ when the document has loaded.

3,4 The function called from the `<a href>` tag first checks the loading status of the target frame, which has been named ‘targetFrame’ in the parent frameset page. The variable, ‘loaded’ needs to be specified by the parent frameset and the name of its own frame.

5 If the target frame has not yet loaded the *setTimeout()* waits 5 sec and tries again.

6 When the target frame has loaded the appropriate code is executed.

Comments:

If the frame never loads you are presumably stuck in an infinite loop.

History: Back or Forward through the Pages

Purpose

I’m not quite sure why you would want to set up your own ‘back’ button, when the browser has it own — perhaps to help naïve users.

Code

```
1    onClick = "history.go(-1);"
     or
2    <a href="javascript:history.back()"> back </a>
```

Comments

1 Using a form or v. 4 browser `<a href>` tag.

2 Using a `<a href>` tag on all JavaScript versions. In this case you may wish to manipulate the status line appropriately (if you don’t suppress it) by:

```
onMouseOver = "self.status=document.referrer; return true;"
```

Alternative

You can, of course, go forward in the history, if appropriate, with `history.forward()` or `history.go(1)`.

Link Focus: Suppression

Purpose

To prevent the dotted focus rectangle in IE Windows that indicates that an image (or piece of text) is a link. This can destroy the aesthetics of graphical links.

Code

```
     // Provided by Magnus Mellin
     function fixIE()
     {
1         for (var n in document.links)
          {
2              document.links[n].onfocus = document.links[n].blur;
          }
     }
3    document.onmousedown = fixIE;
```

Comments

- `<HEAD>` location.
- This also works for image maps, unlike alternatives that rely on the `<a href>` tag for onClick() statements.

Explanation

1 The “for/in” loop in JavaScript allows one to iterate through the different instances of an object. Here *n* is an integer used to look at all the links objects in a document which are represented as document.links[0], document.links[1] etc.

2 The body of the loop just says, for each of the links the ‘onfocus’ property should be replaced by the ‘blur’ property, i.e. focus should be prevented.

3 Note the lack of parentheses after the function name. This is because we require a definition for interpretation on subsequent execution. If the parentheses had been included the function would be automatically executed immediately (although this would do nothing — but you can test it with an ‘alert’ call in the function), but would not be invoked interactively on ‘mouseDown’.

Randomization

Purpose

To select a random item for a page. This could be a simple text string, or an image (the example here) from a set of such items.

Head Code

```
1    var prefix = 'Pic';    // prefix to set of image names
     var max = 5;           // total number of images
2    var picArray = new Array(max);
     for (var i=0; i<max ; i++)
     {
3        picArray[i] = prefix + (i + 1) + ".jpg";
     }
4    function makeRan(n)
     {
5        return = Math.floor(Math.random()*n);
     }
6    var ranNum = makeRan(picArray.length);
```

Head Code Explanation

1 One needs to chose a systematic naming system and specify the number of images

2 Construct an array (JS 1.1) of a size equal to the number of images.

3 Fill the array with the src of the images.

4,6 Generate a random number (JS 1.1) between 0 and one less than the array length.

5 Math.random() generates a pseudo-random 16-digit floating-point number between 0 and 1, which is multiplied by the number of items, *n*. Math.floor() rounds this down to the nearest integer, so that a range of integers between 0 and *n* is possible in theory, although in practice the range is 0 to *n*–1, as an exact hit on *n* is most unlikely.

Body Code

```
1    document.write('<img name="myPic" src="' + picArray[ranNum] + '">');
```

Body Code Explanation

1 The line specifying the image is in a javascript ‘document.write’ statement with the src pointing to a random position in the array of images.

Comments

- Images would normally be preloaded and the size specified in the document.write statement.
- See the section ‘JavaScript 1.0 Workarounds’ for alternatives to the JavaScript 1.1 array and random number generation.

Resize Bug-fix

Purpose

To deal with the bug in Netscape 4 which causes a frame with css to ignore the css when redrawing the page after a resize.

Head Code in Frameset Page

```
/**
* resize.js 0.3 970811
* by gary smith
* js component for "reloading page onResize"
*/

if(!window.saveInnerWidth)
{
    window.onresize = resizeIt;
    window.saveInnerWidth = window.innerWidth;
    window.saveInnerHeight = window.innerHeight;
}

function resizeIt()
{
    if (saveInnerWidth < window.innerWidth ||
    saveInnerWidth > window.innerWidth ||
    saveInnerHeight > window.innerHeight ||
    saveInnerHeight < window.innerHeight )
    {
        window.history.go(0);
    }
}
```

Comments

This is code supplied by Netscape that I don’t completely understand. The basic idea is that on resize the window is reloaded (`window.history.go(0)`). The `innerWidth` and `innerHeight` are NS4-only properties, which give the code browser-specificity. What the `saveInnerWidth` does I do not know.

Rollovers: Preloading Images

Purpose A: For rollovers (Pre-load in Head)
To load images into memory before they are required by the user. This is most used to make sure the alternative images needed for rollovers can be loaded instantly. For this reason the loading is done in the head.

Purpose B: For rapid image loading (Pre-load after page load)
An example of this would be in an art-gallery series, where the image for the next page is loaded while the user is viewing the image on the current page.

Preloading in the head will delay the loading of the current page (at least for the first of a series). In this case the pre-loading code (in the examples that follow) is placed in a function in the head which is invoked by an 'onLoad' call in the body tag, i.e. after the page has loaded:

```
1    function preLoader()
     {…}

2    <BODY onLoad = "preLoader()">
```

Code 1: Basic preload

```
1    var preload = new Image(220,310);
2    preload.src="pics/art_0.gif";
```

Explanation 1

1 Constructor for a new Image object ('preload') specifying width and height, respectively.
2 Assigns the source of the Image object to a real file (art_1.gif in directory pics).

Code 2: Pre-load into an simple array
For rollovers where you are pre-loading many images with systematic names it is more compact to use a loop to load into an array. Here we have a set of six images, art_0.gif to art_5.gif, that we pre-load into an array:

```
var picArray = new Array();     // create a new array (js 1.1)
var len = 6;                    // number of items in array
for (i = 0; i < len; i++)
{
    picArray[i] = new Image();
    picArray[i].src = 'pics/art_' + i + '.gif';
}
```

In fact, with rollovers you would only have two or three images (over, out and click) with the same base name. You could create two or three arrays and fill them in the same loop.

(The fact that JavaScript 1.0 does not support real arrays is irrelevant for rollovers as the Image object is not defined in 1.0 either.)

Code 3: Pre-load into an associative array
There is another strategy for handling rollovers that involves creating an array in the preload, but the difference is that the sort of array used is an associative array, equivalent to a JavaScript object, and if you test for the length of the array you will find it to be 0. The key to this difference is the difference between the way the new Image object is related to the array. In the above case the code for the first iteration of the 'for' loop translates to something like

```
picArray[0] = new Image();
```

whereas in the example overleaf the code translates to something like

```
picArray[home_0] = new Image();
```

The associative picArray is more akin to an object in the page hierarchy, which can contain sets of other objects (Images). In this case ‘home’ is a name specified within the `<image>` tag, and the strategy is to pass this as the parameter to the different head functions called by the event handlers, ‘onMouseOver’, ‘onMouseOut’, ‘onMouseClick’, so that the appropriate replacement image can easily by referenced. An example of this will be given in full in the section on Rollovers. For the moment, we just present example code for the construction of the associative array.

```
1    menuArray = new Array('home','products','news','contact');

2    picArray = new Array();
3    for (i = 0 ; i<menuArray.length ; i++)
     {
4        for (j = 0; j < 3; j++)
         {
5            var temp = menuArray[i] + '_' + j;
6            picArray[temp] = new Image();
7            picArray[temp].src = menuArray[i] + '_' + j + '.gif';
         }
     }
```

1 This is the real array, which must be hard coded with the names from the image tag.
2 This is the associative array.
3,4 The outer ‘for’ loop (variable i) iterates once for each base name, whereas the inner ‘for’ loop (variable j) iterates once for each type of image (over, out, click).
5 In generating a unique identity for each image, counter i for the outer loop is used to reference the contents of the other array (menuArray), whereas j is used directly to add a suffix, 0, 1, or 2, to this base name.
6 One might still have used a counter to assign this name to a numbered cell in the array, but this is not done. Instead one uses this as the name of an Image to assign to the picArray object.
7 And one uses a variant on 5 to assign an src to the Image, completing the preload.

Rollovers: Simple

Purpose

To exchange the image on a page with a second image when the mouse ‘rolls over’ the image.

Absolute image reference in the event handler

Code

```
1    <A HREF = "link.html"
2    onMouseOver = "document.imgName.src = 'new.gif';"
3    onMouseOut = "document.imgName.src = 'old.gif';">
4    <IMG NAME = "imgName" SRC = "old.gif">
     </A>
```

Comments

- `<BODY>` location — obviously, as that is where the images are.
- JavaScript 1.1, but browsers that don’t support this will just ignore it.
- ‘document’ may be omitted in this case, but is required if the rollover affects a different image (*see* Indirect Rollovers).
- The semi-colons are generally omitted at the end of the event handlers. However, they are useful reminders that one is writing a JavaScript statement, and are needed when there are two or more statements in the event handler.

Explanation

1 The code uses html event-handlers that can appear in the `<A HREF>` tag.

4 First look at the `<IMG>` tag — it not only specifies the actual name of the initial image to be displayed (old.gif), but a variable name for the image held in this position (imgName).

2,3 The event-handlers, ‘onMouseOver’ and ‘onMouseOut’, specify the source (SRC) of the image named ‘imgName’. The ‘onMouseOver’ handler typically specifies a different source from the default image specified in the `<IMG>` tag, and the ‘onMouseOut’ handler typically restores the original.

Relative image reference in the event handler

By itself, the code above has the limitation that there will be a lag the first time the alternative image is displayed because of the time required to fetch the image from the server. Therefore rollovers are always used on images already preloaded as described elsewhere. In this case in the event handler one can refer to the variable name used in the preloads, e.g.

```
var preload1off = new Image(220,310);
preload1off.src="old.gif";

onMouseOut = "document.imgName1.src = preLoad1off.src;"
```

This may make the code clearer for a set of images.

Calling a function from the event handler

Where there is a set of alternative images that have been systematically named, one can make the code much more compact by calling functions, defined in the head, which handle the image replacement.

Body Code

```
1    <A HREF = "link.html"
2    onMouseOver = "activate(pic1);"
3    onMouseOut = "inActivate(pic1);">
4    <IMG NAME = "pic1" SRC = "old1.gif">
     </A>
```

Body Code Explanation

4 The variable name given to an image typically has an alphabetic stem and a numeric suffix.

2,3 The event-handlers, ‘onMouseOver’ and ‘onMouseOut’, pass this systematic name to functions, ‘activate’ and ‘inActivate’ in the head.

Head Code for the functions

```
     function activate(pic)
     {
1        document[pic].src = eval (pic + "on.src");
     }
     function inActivate(pic)
     {
2        document[pic].src = eval (pic + "off.src");
     }
```

Function Code Explanation

1,2 The effect of these functions is not difficult to deduce. It generates a line equivalent to:

```
document.pic1.src = 'new1.gif';
```

given that ‘pic1off.src’ has been defined as ‘new1.gif’ in the preload (*see* below). However it involves two rather aspects of JavaScript that reflect its similarity to other scripting languages, such as Perl.

The first is the eval() method which is needed because we are not just concatenating ‘on.src’ to the string, ‘pic1’, passed to the function, but we are evaluating the resulting variable, ‘pic1on.src’, which has previously been defined as ‘new1.gif’ in this example.

The second is the use of the associative array syntax for the document object instead of the dot syntax. This is because the dot syntax requires a literal identifier and cannot take a variable, unlike the associative array nomenclature that can accept a string.

Head Code for the preload

```
pic1on = new Image(200,300);
pic1on.src = "new1.gif";

pic1off = new Image(200,300);
pic1off.src = "old1.gif";

pic2on = ...etc
```

Head Code comments

Need to check for javascript version 1.1 by ‘`If (document.images){…}`’

Rollovers: Indirect

Purpose

In the simple rollover, when the mouse rolls over an image, that image is replaced by an alternative version, and the original is restored when the mouse moves out of the image. In indirect rollover, when the mouse moves over one image (No. 1), it is a separate image (No. 2) that is replaced. This is generally used to exchange a blank image (transparent or background colour) for one with information. It should be used with care.

Code

```
1    <IMG NAME = "img2" SRC="blank.gif"><p>
     <A HREF = "link.html"
2    onMouseOver = "document.img2.src = 'message.gif';"
3    onMouseOut = "document.img2.src = 'blank.gif';">
4    <IMG NAME = "img1" SRC = "picOff.gif">
     </A>
```

Comments

- `<BODY>` location — obviously, as that is where the images are.
- In the `<AREA>` tag instead of the `<A>` tag for image maps.
- JavaScript 1.1, but browsers that don't support this will just ignore it.
- 'document' must be included.
- You would preload at least the image for 'onMouseOver'.

Explanation

`1` This is the image, initially blank, that is going to be exchanged. It must have its name defined.

`2,3` The event handlers are within the tags for the same image as before (old.gif), but they refer to the other image, entitled text.

`4` There is no need to name the image with the event handlers in this case.

Alternative 1: Double rollover

It is possible to have two images replaced on rolling over one. All you need to do is add a second image definition statement after the first (with a semi-colon separating the two):

```
<IMG NAME = "img2" SRC="blank.gif"><p>
<A HREF = "link.html"
onMouseOver = "document.img2.src = 'message.gif';
document.img1.src = 'picOn.gif';"
onMouseOut = "document. img2.src = 'blank.gif';
document.img1.src = 'picOff.gif';">
<IMG NAME = "img1" SRC = "picOff.gif">
```

Alternative 2: Without link

If you are just wanting a visual effect, there need not be an actual link within the `<A HREF>` tag. You can substitute:

```
<A HREF = "#"
```

Returning 'true' at the end of the onMouseOver suppresses the status text.

Alternative 3: Targeting another Frame

This is quite straightforward. One just needs to specify the full path to the image using the frame name. For example if the target frame is given the name 'right' in the frame setup:

```
onMouseOver = "parent.right.document.img2.src = 'message.gif' "
```

You have to include 'document' (and you might include 'images' if you wish).

Rollovers: Latched

Purpose

To provide a menu that changes from a default state to a rollover state on mouseOver, but has a third state when the menu button has been clicked. The explanation assumes the section on use of functions in simple rollovers has already been read.

Body Code

```
1    <A HREF = "link.html" target="contentFrame"
2    onMouseOver = "activate('pic1');"
     onMouseOut = "inActivate('pic1');">
3    onClick="latch('pic1');">
     <IMG NAME = "pic1" SRC = "old1.gif">
     </A>
```

Body Code Explanation

1 In this sort of implementation, the link would be targeted to another frame.
2 The image name, ‘pic1’, *must* be passed in single quotes.
3 The additional event-handler, ‘onClick’, calls a third function (below).

Function Code

```
1    var activePic = '0';

     function latch(pic)
     {
2        document[pic].src = eval (pic + "latch.src");
3        activePic = pic;
     }

     function activate(pic)
     {
4        if(pic != activePic)
         {... as before

     function inActivate(pic)
     {
5        if(pic != activePic)
         {... as before
```

Function Code Explanation

1 We need to declare a global variable to indicate which is the ‘sticky’ pic.
2 This generates a line of the type:
`document.pic1.src = sticky1.gif`
given that ‘pic1latch.src’ has been defined as ‘sticky1.gif’ in the preload*.
3 Then we need to this as the ‘activePic’.
4,5 And our other two functions need to contain ‘if’ statements to prevent rollover and rollout on the active pic.

***Final Comment**

For latched rollovers on menus, preloading into an associative array, as described in the Preload section, has much to recommend itself.

Rotation: Frames or Images by Button Control

Purpose

To rotate through a series of frames or images (if they are the same size) using forward and back buttons. The example is for frames.

Head Code

```
    var prefix = 'pref';   // common prefix to names of html pages
    var current = 1;       // initial serial number
    var total = 7;         // total number of pages numbered from ‘1’
    function goForward()
    {
1       if (current <total)
        {
            current++;
        }
        else
        {
            current = 1;
        }
2       parent.rotateFrame.location.href = prefix + current + '.html';
    }

3   function goBack()
    {
        if (current != 1)
        {
            current--;
        }
        else
        {
            current = total;
        }
        parent.rotateFrame.location.href = prefix + current + '.html';
    }
```

Head Explanation

1 The counter is advanced or returned back to ‘1’.

2 This sets the page to appear in the target frame (‘rotateFrame’) using the prefix and the incremented value of ‘current’.

3 The back() function is analogous to forward().

Body Code

```
1   <A HREF="javascript: goForward()"><img src="forward.gif" border=0></A>
```

Body Explanation

1 The usual url javascript to call the function. The back button is analogous.

Alternatives

- Rotating images are done in an analogous fashion, changing the code to something like:
  ```
  document.imgName.src = prefix + current + ".jpg";
  ```
  where ‘imgName’ is the name of the image specified in the <img> tag.
- If starting with a random image, set ‘current’ initially to the random number used for this.

Rotation: Automatic

Purpose

A number of JavaScript effects depend on automatic changes to JavaScript objects, either in a predefined sequence or in a random fashion. Both aspects of the code to do this are dealt with elsewhere, and indeed together in the horizontal text Scroller It was felt that it might be useful to reiterate the principle here.

Head Code

```
     function rotateStuff()
     {
1         Execute the code in an expression using either a random number
          or an incremented counter variable

2         Get a new random number or increment/reset counter variable

3         setTimeout("rotateStuff()", delayTime);
     }
```

Body Code

```
4    <BODY onLoad="rotateStuff()">
```

Explanation

1 The code that does the business is here. In the scroller code it wrote a string to a form field, but it could be used to change frames with different banner ads etc.

2 After the code has been executed, some integer that determines the way the code is executed needs to change ready for the next execution.

3 The setTimeout() method allows one to set a delay time, *delayTime* (this variable will have been defined in the head code), before the function is called again recursively.

4 The `<BODY>` tag seems the most reasonable place to initiate this, but one could have it activated by a button click, I suppose.

Scrolling Text

Purpose

To present a continuously scrolling text message in a small ‘window’.

Head Code

```
     var pause = 100;       // millisec pause between movement
1    var subSize = 50;      // size of substring for form
     var position = 0;      // start index of sub-string
     function scroller()
     {
          var msg = "If msg < form input field, pad with spaces.       ";
2         do
          {
               msg += " " + msg;
          }
          while (msg.length < 2*subSize);
3         document.myform.scrollfield.value
                    = msg.substring(position, position + subSize);
4         position ++;
          if (position == (msg.length-subSize) )
          {
5              position = 0;
          }
6         setTimeout("scroller()", pause);
     }
```

Body Code

```
7    <BODY onLoad="scroller()">
     <form name="myform">
8    <input type="text" name="scrollfield" size="50"
          value = "The initial message can differ       ">
     </form>
```

Explanation

1,2 The key trick to get the effect of scrolling is to concatenate the message so one can generate substrings in which the start of the message string is tacked onto the end. The substring size is set to the length of the form field, and the do/while loop ensures the message is concatenated at least once and is long enough for the substring.

3 This generates a substring from a starting position ‘position’ — initially set at ‘0’ — to be allocated as the ‘value’ in the input field ‘scroller’ in the form ‘myform’, defined in the body.

4,5 This advances the substring start index (‘position’) ready for the next time the function is called (so as to give the effect of scrolling), but sets it back to ‘0’ just before the end index falls outside the string.

6 This sets a delay time of ‘pause’ and then makes a recursive call to the function, ensuring once started the scrolling goes on for ever.

7 The scroller function is started by the onLoad call in the `<BODY>` tag.

8 The form set up needs to ensure that its name and the name and size of the input field conform to the code in the scroller() function, and that there is a suitable initial message. The msg string can be much larger than the size of the field, if desired.

Status Line Manipulation

Purpose

To change the url appearing at the bottom of the browser window, especially when it either doesn't lead anywhere or where it references another frame in an unhelpful manner.

Code

```
     <A HREF = "link.html"
1    onMouseOver="document.imgName.src='new.gif;status='New!';return true;"
2    onMouseOut ="document.imgName.src='old.gif;">
     <IMG NAME = "imgName" SRC = "old.gif">
     </A>
```

Comments

`<BODY>` location.

Explanation

1 'status' refers to the text on the status line. 'window.status' would be more complete
You must return 'true' for the text to appear instead of the link url.

2 There is no point in returning 'true' for 'onMouseOut' as this clears the status line anyway. You could include `status=''` for the sake of users of NS3 (PC), where there was a bug that prevented the status line clearing.

Alternative 1: No status message

You can suppress any message just by returning 'true', without the need to specify a status string.

Alternative 2: Automating status messages for menu items

If you have a function for your rollovers (see Latched Rollovers) you can include code in that function for specifying your messages and then all you need to do in the body is to return 'true':

```
activate(pic)
{
     ...
     if (pic == 'home')
     {
          window.status = 'Home Page';
     }
     ...
}
```

Alternative 3: 'Return false' for dummy links

This is not really manipulating the status line, but it has an effect on it. If you have a dummy link with an 'onClick' (`<a href="#" onClick...`) 'below the fold', clicking on it (to call a function) will cause the page to refresh to the top in NS. Returning 'false' in the link tag will prevent this, although the dummy link will be visible on the status line.

Windows: Closing

Purpose

Although the user can close a pop-up window in a platform-dependent manner, I would encourage the provision of a close-button. This makes it clear that you are not trying to take over his desktop with extra windows, and makes the window look designed for its purpose.

Head Code

```
     function closeIt()
     {
1        self.close();
     }
```

Head Comments

1 One could also use ‘this.close()’, ‘close()’ or ‘window.close(self)’.

Body Code

```
1    <form><div>
2        <input type=button value="Close" onClick="closeIt()">
     </div></form>
```

Body Comments

1 Using a form allows means you don’t have to use graphics for the button.
2 The button will be labelled ‘Close’ and clicking calls the head function.

Alternatives

- Custom buttons. Only works for browser versions that support ‘onClick’ for Images.
- Automatic self closure. I use this for applet ‘help’ windows where one cannot set the parameters of the additional browser window created from within the applet. The head section of the html page for the new window has the following lines:

  ```
  var newWin = window.open("Help.html","Help","width=400,height=250")
  var closeIt = window.close(self);
  ```

  The first line opens a new small window, after which the second line closes the current large one.

Windows: Setting Characteristics for Pop-ups

Purpose

To open a new window with characteristics other than the standard ones. In particular having chromeless windows with specific dimensions appearing in the centre of the browser window.

Head Code

```
// Glenn Davis of Project Cool, Inc.
     var args;
1    function openWin(url, w, h)
     {
2         args = 'width=' + w + ','
          + 'height=' + h + ','
          + 'toolbar=0,'
          + 'location=0,'
          + 'directories=0,'
          + 'status=0,'
          + 'menubar=0,'
          + 'scrollbars=0,'
          + 'resizable=0';
3         if (parseInt(navigator.appVersion) >= 4)
          {
4              var xPosition = (screen.width - w)/2;
5              var yPosition = (screen.height - h)/2;
               args += ','
6              + 'screenx=' + xPosition + ','  //NN
               + 'screeny=' + yPosition + ','  //NN
7              + 'left=' + xPosition + ','     //IE
               + 'top=' + yPosition;           //IE
          }
8         window.open(url,"New One", args);
     }
```

Head Comments

1 The parameters of the function are url, width (w) and height (h).

2 The string ‘args’ will encompass all the features of the window that we wish to set. Those that are not self-evident include: ‘location’ (url entry field), ‘status’ (status line), and ‘directories’ (directory buttons).

3 To prevent JavaScript errors in v. 3 browsers, one has to test for browser version here.

4,5 xPosition and yPosition are the x and y zero co-ordinates to centre the pop-up.

6,7 NS4 and IE4 use different JavaScript variables, but ignore the non-relevant information.

8 This calls the window.open() method. The second variable is the window name.

Body Code

```
1    <a href="javascript:openWin('NewWin.html',150,300)">Open Me!</a>
```

Body Comments

1 This example calls the function from a link.

Alternatives:

- One could modify the function to allow other features of the window to be set.
- One can call the function from buttons or automatically by assigning it to a variable.

Windows and Focus

Purpose

To maintain a small pop-up window at the front so that it does not get hidden.

Code

```
<body onBlur="self.focus()">
```

Comments

- On NS 4.72 (Mac) — but not on NS 4.08 — there is a problem regaining proper focus of the original window. I tried without success to eliminate it by `<body onUnload = "self.blur()">` or `<body onUnload = "self.opener.focus()">`
- This is JavaScript 1.1 code.

Alternative

If all one wants is to ensure that the pop-up has *initial* focus, use this function in the head:

```
function grabFocus()
{
    window.focus();
}
window.onload = grabFocus();
```

Again, this is JavaScript 1.1 code. It was suggested to fix a problem when the link to the pop-up was at the bottom of a page, which moved back to the top and gained focus on launching the pop-up.