20 Dynamic HTML Tips & Tricks: Defining Down the Browser War
by David Boles
February 16, 1998

During the Summer of 1997, Dynamic HTML (DHTML) was the new buzz word on the Internet. Webmasters the world over wanted to include it on their sites in order to easily add interesting animation, interactivity and multi-media features to their website.

But now, eight months later, how easy is it to code a site with DHTML in the midst of the browser war between Netscape Navigator 4.0x and Microsoft Internet Explorer 4.0x? Is it possible to code cross-browser DHTML? The answer to that question begins with who you ask. This article will deal with the problems, solutions and conventions associated with creating cross-browser compatible DHTML based upon the experiences of top internet technology individuals. These DHTML 20 Cross-Browser Tips and Tricks should help you understand the minefield awaiting you and help you avoid blowing up one browser in favor of another.

The problem with defining DHTML is discovering that it means different things to different people. The trick is to wade through the present mire of definitions, select the bits and pieces you like and label them "DHTML" for now. I consider DHTML to be a mishmash of JavaScript, Cascading Style Sheets, a broken DOM spec that will be made whole when implemented by both Microsoft and Netscape sometime in the future and some animation and other streaming mega-media presentations like Shockwave.

Here's why it presently isn't easy to code cross-browser DHTML:

1. Existing standards have yet to be fully realized by any vendor.

2. Current capabilities of both Netscape Navigator and Microsoft Internet Explorer haven't let the "standards process" catch up with their present functionality.

3. When the standards catch up to browser capability, there will be lag time while every vendor comes into full compliance.

What to do right now? I suggest strolling through these expert DHTML Tips & Tricks to discover some neat workarounds, fixes, and present ways to incorporate successful "DHTML" on your website. Once you accept these web realities, you'll have accomplished your first DHTML trick: Making DHTML into whatever you want it to be.

1: Memorize the "Pain Meter"
Microsoft's Scott Isaacs urges you, in his excellent whitepaper, "Cross Browser DHTML Authoring" to know before you code, that each step up in functionality increases the chances of incompatibility problems between Netscape Navigator and Microsoft Internet Explorer. The "DHTML Pain Meter" scopes out like this:

Lowest Common Denominator - no CSS or Script (All browsers)

Simple Form Validation Scripts (IE3, IE4, NS3, NS4)
Image Rollovers (NS3, NS4, IE4)
Simple CSS-1 (IE3, NS4, IE4)
Advanced CSS-1 for formatting and CSS-P for positioning (NS4, IE4)
Animating Elements (NS4, IE4)
DHTML with user-interaction

2: Think Download Time
Jeff Rule, Senior Microsoft Developer, Discovery Channel Online urges you to consider the temptation with DHTML is to recreate CD-ROM interfaces on the web. Coming out of Macromedia's Authorware and Director schools, Rule was eager to create CD-ROM quality content for the debut of the 4.0 browsers. There's a danger in squeezing everything on a single page. Break up your experience into a new page now and then. Animations and image resizing are simple ways to add multimedia to your pages without downloading a lot of additional artwork. Adding code is cheap in download cost while adding graphics is expensive.

3: Mask Latency
Another Jeff Rule trick is to mask latency as much as you can. If you're going to force a large graphics download on your visitors, at least put something on the screen to entertain the visitor while the download is in progress. Something simple like a flashing "loading" text or status thermometer will make your visitors more tolerant of a long download.

4: Degrade Gracefully
Eric Krock of Netscape suggests the following for clean backward compatibility and graceful degrading of JavaScript in various Versions of Navigator:

- use `<SCRIPT>` and `<NOSCRIPT>` to separate content for JavaScript-capable browsers and non-JavaScript browsers.

- Note: Navigator 2 does not support the `<NOSCRIPT>` tag; it will display content that is enclosed in a NOSCRIPT tag even though it is a JavaScript-capable browser. Therefore, if your site supports both text-only browsers and Navigator 2, you cannot rely upon `<NOSCRIPT>` to mask the text-only content from Navigator 2.

Workarounds: if your site must support Nav2.x, do one of the following:

a) Place the text-only content and the JavaScript-enhanced contents in separate files and selectively direct browsers from your main page to the appropriate text-only or JavaScript page via via two-targeted links (as described below)

b) Make the JavaScript-enhanced content your main page (since the overwhelming majority of viewers will see this anyway), appropriately hiding the JS scripts from text-only browsers via enclosing HTML comments, and provide a "Click here for text only view" link that the relatively few text-only viewers can follow.

- use <SCRIPT LANGUAGE=version> where version is "JavaScript", "JavaScript1.1", or "JavaScript1.2" to separate code for Navigator 2, Navigator 3, and Navigator 4 respectively

- however, Navigator 3 has a known bug which causes it to evaluate code within a <SCRIPT LANGUAGE="JavaScript1.2">, so enclose those Nav4-specific statements within this explicit JavaScript version check:

```
// Workaround for Nav3.x bug in which JavaScript files
// with <SCRIPT LANGUAGE="JavaScript1.2" SRC=...> are
// loaded, even though they should be ignored.
// Make sure this code is only executed by 4.x and later.
if (parseInt(navigator.appVersion) > 3) { ... }
```

- in a <SCRIPT LANGUAGE="JavaScript"> tag, after checking the browser version and/or vendor, use document.write() statements to dynamically generate optimized markup for the current browser (and generate nothing, as desired, for non-JS browsers)

- use two-targeted links to selectively direct JavaScript browsers to JavaScript-enabled pages and non-JS browsers to non-JS pages, e.g.:

```
<A HREF="nonJSPage.html"
onclick="location='JSPage.html';return false">
Go to Info Page</A>
```

JavaScript browsers evaluate the onclick event handler first, are sent to the JavaScript page, and then stop because of the return false; non-JavaScript browsers ignore the JavaScript event handler and follow the normal HTML link to the non-JavaScript page.

5: Use Advanced Frames & Tables

Brandon Arnold, Web Developer for HYPERformance Website Design, urges the extensive use of advanced frames and tables: "I can't say enough about learning these two facets thoroughly. They can make or break a Website's appeal. If used correctly, a Website can shine and used incorrectly they can not only make a site look highly unorganized, but may not even load at all in frames view, so be sure to include something in the "noframes" portion of your framed pages."

6: Understand the Document Object Model

Kevin Lynch, Vice President of Internet Technology for Macromedia, dissects the DOM for you on Macromedia's DHTMLZONE.COM and he's graciously allowed us to use the following SuperTip from his online tutorial:

The Document Object Model is a common concept in both browsers, providing scripting access to web page elements. The actual Microsoft and Netscape object models are different, making it challenging to write JavaScript which works in both--we describe how to accomplish this below.

Microsoft model
Scripting access to all HTML page elements is supported in Internet Explorer, including style sheet properties. Page elements are reflected as objects contained in a document.all collection, and elements can be accessed by index, name or ID.

For example, to write out the names of all tags on a page:

```
for (i=0; i < document.all.length; i++)
   {
   document.write(document.all[i].tagName + "\n");
   }
```

To access a tag by ID:

```
document.write("myLayer visibility is:" +
    document.all['myLayer'].style.visibility);
```

Any changes to object properties appear instantaneously on the page.

Netscape model
Scripting access to specific collections of HTML page elements is supported, such as layers on the page. The layer collection in Navigator includes areas bounded by a <layer> tag and areas positioned using CSS attributes. Elements can be accessed by index, name or ID within those collections.

For example, to write out the names of all layers:

```
for (i=0; i < document.layers.length; i++)
   {
   document.write(document.layers[i].name + "\n");
   }
```

To access a layer by ID:

```
document.write("myLayer visibility is:" +
    document.layers['myLayer'].visibility);
```

Changes to layer position, visibility, and clipping appear instantaneously on the page.

Cross-Browser model
To access objects in either browser, a reference variable can be set when the page is loaded to the syntax for whichever browser is being used:

```
if (navigator.appName == "Netscape") {
   layerRef="document.layers";
   styleRef="";
} else {
   layerRef="document.all";
   styleRef=".style";
}
```

Then one script can be used in both browsers, by building references to objects. For example, to test the current visibility of an element named myLayer:

```
isVisible = eval(layerRef + '["myLayer"]' + styleRef + '.visibility');
```

7: Use Active Server Pages & Cold Fusion

Another Brandon Arnold trick is to proactively use Active Server Pages and Cold Fusion because these two technologies are where the web is headed. Using include files along with these two technologies along with database connectivity will set the new standard in web development. “Learn it now!”, Arnold urges.

8: Animate Your Text

Here’s an outstanding, fully annotated code example of DHTML text animation from George Olsen, Design Director/Web Architect, of 2-Lane Media.

```
<HTML>
<HEAD>
<TITLE>2LM Demo - DHTML coexisting with HTML
3.2</TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;
charset=iso-8859-1">

<!--
        For now I'm using the default scripts generated by Dreamweaver. However, I'm planning to
rework them to deal with a NS 2.0 bug.  NS 2.0 doesn't understand the new Array command
(introduced in JS 1.1) so it displays an error message. Otherwise the page functions fine.
Theoretically, changing SCRIPT LANGUAGE=JavaScript1.1 should fix this but it's not working.
However, it's possible use an alternate method to create arrays that's compatible
with NS 2.0, which is something Dreamweaver should offer as an option.  DW's scripts are a bit
cryptic so I'm still reverse engineering them.  So for the moment we'll assume "magic
happens" since my trick doesn't really concern the animation as much as combining the animation
in a way that's compatible with older browser. See comments below BODY tag.
-->

<!-- Run animation only if
it's a NS/IE 4.0+ brower -->
<SCRIPT LANGUAGE="JavaScript">
function
doCheckDHTMLcapable() {
       browserName = navigator.appName;

       browserVersion = parseInt(navigator.appVersion);
              if
(browserName == "Netscape" && browserVersion >=4) {

       MM_timelinePlay('Timeline1') // If it's NS 4+, run the animation

       }
       else;
       if (browserName == "Microsoft Internet Explorer" &&
browserVersion >=4) {
              MM_timelinePlay('Timeline1') // If it's IE
4+, run the animation
       }
       else;
       // otherwise don't run the
animation
}
</SCRIPT>

<!-- This script holds the positioning info of the
elements throughout the animation -->
<SCRIPT
LANGUAGE="JavaScript">
function MM_initTimelines() {
```

```
//MM_initTimelines() Copyright 1997 Macromedia, Inc. All rights reserved.

var ns = navigator.appName == "Netscape";  // set up check used to
customize syntax
    document.MM_Time = new Array(1);

document.MM_Time[0] = new Array(1);
    document.MM_Time["Timeline1"] =
document.MM_Time[0];
    document.MM_Time[0].MM_Name = "Timeline1";

document.MM_Time[0].fps = 15;
    document.MM_Time[0][0] = new
String("sprite");
    document.MM_Time[0][0].slot = 1;
    // The above
line handles alternate syntax needed by Netscape and Explorer

document.MM_Time[0][0].obj = (ns) ? document.animatedText :
document.all["animatedText"];
    document.MM_Time[0][0].keyFrames = new
Array(1, 15);
    document.MM_Time[0][0].values = new Array(2);
    // The
line below holds the horizontal positions for each move

document.MM_Time[0][0].values[0] = new
Array(300,279,257,236,214,193,171,150,129,107,86,64,43,21,0);

document.MM_Time[0][0].values[0].prop = "left";
    // The line below holds
the vertical positions for each move
    document.MM_Time[0][0].values[1] =
new Array(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);

document.MM_Time[0][0].values[1].prop = "top";
    // The lines below
handle alternate syntax needed by Explorer and Netscape
    if (!ns) {

document.MM_Time[0][0].values[0].prop2 = "style";

document.MM_Time[0][0].values[1].prop2 = "style";
    }

document.MM_Time[0].lastFrame = 15;
    for (i=0;
i<document.MM_Time.length; i++) {
        document.MM_Time[i].ID = null;

document.MM_Time[i].curFrame = 0;
        document.MM_Time[i].delay =
1000/document.MM_Time[i].fps;
    }
}
</SCRIPT>

<!-- This sets up the
autoplay -->
<SCRIPT LANGUAGE="JavaScript">
function
MM_timelinePlay(tmLnName, myID) { //v1.0
  //Copyright 1997 Macromedia,
Inc. All rights reserved.
  var
i,j,tmLn,props,keyFrm,sprite,numKeyFr,firstKeyFr,propNum,theObj,firstTime=false;

  if (document.MM_Time == null) MM_initTimelines(); //if *very* 1st time

tmLn = document.MM_Time[tmLnName];
  if (myID == null) { myID = ++tmLn.ID;
```

```
firstTime=true;}//if new call, incr ID
  if (myID == tmLn.ID) { //if Im
newest

setTimeout('MM_timelinePlay("'+tmLnName+'",'+myID+')',tmLn.delay);
    fNew
= ++tmLn.curFrame;
    for (i=0; i<tmLn.length; i++) {
      sprite =
tmLn[i];
      if (sprite.charAt(0) == 's') {
        if (sprite.obj) {

numKeyFr = sprite.keyFrames.length; firstKeyFr = sprite.keyFrames[0];

if (fNew >= firstKeyFr && fNew <= sprite.keyFrames[numKeyFr-1]) {//in
range
            keyFrm=1;
            for (j=0; j<sprite.values.length;
j++) {
              props = sprite.values[j];
              if (numKeyFr
!= props.length) {
                if (props.prop2 == null)
sprite.obj[props.prop] = props[fNew-firstKeyFr];
                else
sprite.obj[props.prop2][props.prop] = props[fNew-firstKeyFr];

} else {
                while (keyFrm<numKeyFr &&
fNew>=sprite.keyFrames[keyFrm]) keyFrm++;
                if (firstTime ||
fNew==sprite.keyFrames[keyFrm-1]) {
                  if (props.prop2 ==
null) sprite.obj[props.prop] = props[keyFrm-1];
                  else
sprite.obj[props.prop2][props.prop] = props[keyFrm-1];
        } } } } }

} else if (sprite.charAt(0)=='b' && fNew == sprite.frame)
eval(sprite.value);
      if (fNew > tmLn.lastFrame) tmLn.ID = 0;
} }
}
</SCRIPT>
</HEAD>
```

<!--
The ONLOAD in the BODY tag makes sure all the animated elements are loaded before the animation begins. Otherwise there might be missing pieces.
-->

```
<BODY BGCOLOR="#FFFFFF"
ONLOAD="doCheckDHTMLcapable()">
```

<!--
Here's where we combine the Dynamic HTML with the regular HTML 3.2 by nexting the SPAN tag within a TD cell. The 4.0 browsers will attention to the info in the SPAN tag and ignore the TD tag. while older browsers will do the opposite. So each is happy. The key to this technique is that the animated object *must* end up in the same place as it's located within the table. I use relative positioning so that the 4.0 browsers will place the SPAN in exactly the same place as the TD cells (since the SPAN is actually place relative to the TD cell). This assures that margin offsets will remain consistent and that the DHTML elements line up with other non-DHTML elements that have been laid out using tables without having to put all those other elements into SPANs as well (this may be necessary if you use absolute positioning). Using relative position also avoids a current bug with absolute position. If you use absolute positioning, once the "flying text" is animated, something seems to screw up the SPAN width so that each word gets put on its own line (graphics are unaffected nor is text within a non-animated SPAN). It's possible this is due to the specific JavaScript used by Dreamweaver, so I'm going to try setting up the animation using another method and see if the problem reoccurs.

```
      A major trade-off with using relative positioning is that Dreamweaver fails to recognize
the SPAN as a layer and will drop it from Dreamweaver's Timeline window -- even if you later
change the SPAN back absolute positioning. So only do this once you're sure the animation is
correct and be sure to make a back up copy with absolute positioning in case you need to re-edit
thepage.
-->

<TABLE BORDER="1" WIDTH="200" CELLSPACING="0" CELLPADDING="0">

<TR>
      <!-- The cell height tag is Netscape specific, but I put it in for this demo so that the
cell height and SPAN height would be the same without cluttering up the demo with extra code. In
a real page, I'd use a spacer graphic if ensuring a minimum height was necessary. -->

<TD WIDTH="200" HEIGHT="50" ALIGN="LEFT" VALIGN="TOP">
       <SPAN
ID="animatedText" STYLE="position:relative; width:200px; height:50px;
z-index:1; left:310px; top:0">If you're on a 4.0 browser you'll see
animated
      flying text. If you're on a 3.0- browser this text will be
static.</SPAN>
     </TD>
  </TR>
  <TR>
    <TD WIDTH="200" HEIGHT="50"
ALIGN="LEFT" VALIGN="TOP">
       <SPAN ID="regularText"
STYLE="position:relative; width:200px; height:50px; z-index:2; left:0;
top:0">Notice how it aligns with the non-animated text down here,
regardless of what browser you're using.</SPAN>
    </TD>

</TR>
</TABLE>
</BODY>
</HTML>
```

9: Use “All In One” Pages that Support Older Browsers

George Olsen offers another trick on how to create older browser compatibility in a single page:

Until this week I'd assumed we'd have to always provide duplicate DHTML and non-DHTML versions of a page -- something that definitely discouraged us from using DHTML since most of our client's sites are intended for mass audiences. However I've discovered that with some crafty coding you can have an all-in-one pages that contains DHTML animations but also includes HTML that older browsers can display correctly. This is mostly useful for DHTML animations, since you're not losing any functionality when the older browsers ignore the DHTML code.

The secret of how to do it:

1) Lay out the elements of your page with tables, the way you'd normally do for 3.0 browsers.

2) Surround each element (text or graphic) you want to animate with a <DIV> or <SPAN> tag that includes CSS-P positioning coordinates and z-index (layer) information. Since the 3.0 browsers understand <DIV> and <SPAN>, there are two important points to take into consideration. First, it's probably better to use a <SPAN> tag because the <DIV> contains a paragraph return, which can push table elements

apart. Second, if you need to use alignment, the <DIV> or <SPAN> tag will override the alignment set in the <TD> tag. However, the alignment doesn't seem to affect the DHTML animation).

3) Write the animation JavaScript to move these elements around. However, the elements need to end up in the same places as where they're located in the tables.

For 4.0 browsers, the DHTML code takes precedence over the HTML table code, so the animation runs and ignores the tables. Older browsers don't recognize the DHTML code so they ignore it and instead read the HTML table code. This works on the Netscape and Explorer 3.0 and AOL 3.0 browsers as well as old browser that don't understand JavaScript. The one problem browser can be Netscape 2.0, which can generate error messages when it runs across newer JavaScript functions it doesn't understand. For example, Dreamweaver uses JavaScripts arrays, which cause Netscape 2.0 error mesages. (However, I've only figured out this technique two days ago, so I'm hoping to add some errorchecking to the DHTML that eliminates the problem.

The code looks like this.

```
<TABLE BORDER="1" WIDTH="200">
<TR>
<TD WIDTH="200" HEIGHT="50" ALIGN="RIGHT" VALIGN="TOP">
<DIV ALIGN="RIGHT" ID="Layer1" STYLE="position:absolute; width:200px; height:115px; z-index:1;
left:25px; top:11px">
Animated flying text
</DIV>
</TD>
</TR>
</TABLE>
```

10: Steal (But Ask Permission First!)
Stealing pre-existing code (especially JavaScript) is, unfortunately the way of the web! If you find a DHTML text animation or some other neat piece of code you'd like to use on your site, don't just do a VIEW | SOURCE | COPY ALL and paste the remains into your favorite HTML editor. Ask first if you can steal (reuse) the code you admire. 99% of the time the creator of the code will grant you permission to use what you wish on your site if you credit them in a coded comment in the raw HTML.

11: Know The Rules
Scott Isaacs has some simple pieces of advice for creating successful and exciting DHTML pages:

a) Always create properly structured HTML pages. Adding script to a page with improperly structured HTML can possibly cause unpredictable and unexpected results.

For example, a common mistake is to overlap elements:

- the following is common but is actually invalid: <STRONG>Strong <EM>and emphasized</STRONG> and incorrect </EM>. The EM element cannot end past the STRONG element it is contained within.

b) Style is separated from the structure and content.

Separate the presentation from the structure of the document. Instead of using tags to change the appearance of the text (eg., <FONT>), use global or linked style sheets. Style sheets allow you to define the appearance of your document separately from the content. This allows you to change the look and feel of your page and even your entire site without having to change the actual HTML or contents of the document.

c) Behavior is authored generically.

Separate behaviors from the structure. Generic scripts and DHTML Scriptlets allow you to create reusuable code that is more maintainable. For example, if you are adding DHTML outlining capabilities to multiple pages, author this behavior once and associate the page with the script. Do not include the script or different scripts in each page. Doing so makes your web-site more difficult to maintain.

12: Cross-Browser Compatibility Checks
Eric Krock suggests these tips for successful implementation of DHTML cross-broswer compatibility:

1. Use CSS-P for positioning as Microsoft Internet Explorer lacks support for the LAYER tag.

2. Use lowercase event handler names (onclick instead of onClick, etc.) as Internet Exploerer can't handle mixed-case event handler names.

3. Name each element you wish to manipulate via its ID attribute and declare it to be absolutely or relatively positioned via CSSP markup.

4. Element names must be alphanumeric characters only; non-alphanumeric characters in names may cause Nav4 to ignore the element.

5. Define a cross-browser stub function API to bridge the differences between the Navigator 4.0x and Internet Explorer 4.0x Document Object Models.

13: Beware of mouseover text.
Scott Wilkinson of Merrill Lynch Private Client Architecture warns against the overuse of DHTML mouseover text that changes color. “Yeah, it makes a point but it’s everywhere and there are a number of other options that can look much better.” Wilkinson prefers using the layers feature to create a dropshadow behind the original text when one does a mouseover. That way, you’re not highlighting the whole text and you get a cleaner look.

14: Use Layers to Add Description

Another Scott Wilkinson favorite is to use layers to create a description of a link on your page. When you mouseover the link text, a description or other image will appear somewhere on the page that describes the link.

15: Think Simple
Just because you can make everything on your site move, wink and/or <blink>, that doesn't meant you have to go wacky and push DHTML from every corner of the world of your website. Use discretion in what you code and ask yourself constantly if you're just showing off or if you're actually adding functionality.

16: Plan!
Brandon Arnold explains that the only way to ensure your DHTML websites are viewed correctly is to browser check them in all the browsers your marketplace uses. This can be hard to accomplish (especially since Internet Explorer 4.0 removes any trace of Internet Explorer 3.0x and won't co-exists on the same machine even if you install to another HDD), but it is still very important to do. Arnold uses two machines for this purpose, one that houses his old browsers (Navigator 2.0, 3.01 and 3.03 Gold and Internet Explorer 3.02) and one that has the 4.0 versions of Navigator and Internet Explorer.

17: Browser Checking
George Olsen advises putting all your DHTML inside a browser check JavaScript that only runs the DHTML script if the browser can understand it. There are two ways to check this.

The first method is to check whether the browser supports a particular JavaScript object.

```
if (name of object you're detecting) { // If the object exists
// do whatever you're going to do to that object
}
```

The advantage to this is that you don't have to worry about version changes. Either a particular can do something or it can't.

However, sometimes it is useful to check the browser and version by using a traditional browser-check JavaScript. This is good for checking for broad support, such as it is a 3.0- browser or a 4.0+ browser. It's also useful for dealing with specific situations where Netscape and Explorer need different code to do the same thing.

First you can check the browser version, which is good for checking for broad support (is it a 3.0 browser vs. a 4.0 browser). It's also useful for checking for whether the same function needs to be coded differently for Internet Explorer or Netscape and isn't not easy to check for a JavaScript object. For example, Explorer uses myElement.style.visibility = "hidden" while Netscape uses myElement.visibility = "hide" to make an element invisible.

18: Use a “Last Modified” JavaScript

Here’s a great piece of code from Brandon Arnold that will print “last modified” information on your web pages. This allows you to seamlessly let your site visitors know how frequently the information on your pages change.

Last Modified JavaScript:

```
<script language="JavaScript"><!--
function showdate(){
revdate = new Date(document.lastModified);
year = revdate.getYear();
month = revdate.getMonth() + 1;
day = revdate.getDate();
hour = revdate.getHours();
min = revdate.getMinutes();
var mon;
var hor;

var ampm;
if(hour>12) {
hor=hour-12;
ampm="pm";
}
else {
hor=hour;
ampm="am";
}

if(month==0)
mon="January";
if(month==1)
mon="January";
if(month==2)
mon="February";
if(month==3)
mon="March";
if(month==4)
mon="April";
if(month==5)
mon="May";
if(month==6)
mon="June";
if(month==7)
mon="July";
if(month==8)
mon="August";
if(month==9)
mon="September";
if(month==10)
mon="October";
if(month==11)
mon="November";
if(month==12)
mon="December";

document.write("<font size=1 face=\"arial\">Updated " +mon + " " +day  + ", " +
"19" +year + "</font>");
}
// --></script>
```

Where you want the “last modified date” to show up:

```
<script language="JavaScript"><!--
showdate();
// --></script>
```

19: Let Dreamweaver Write DHTML For You
Macromedia's Dreamweaver is a substantial productivity and HTML coding tool that will easily help you create good cross-browser compatible DHTML without the fuss. Think of Macromedia as "The Arbiter of DHTML" and you'll know that what you code once in Dreamweaver will work the wonders you seek in both Netscape Navigator 4.0x and Microsoft Internet Explorer 4.0x.

20: Give Up Now, Wait for HTML 4.0 & the Universal DOM
Beware that, in my estimation, DHTML is presently "The King without Clothes" – for the precise definitions of what constitutes DHTML are as wild and varied as the warp and woof of the Web. However, with these tips, you can begin to see the magnificent cape that can be stitched together with luck and a little bit of sweat and spit.

Both Netscape and Microsoft have committed to come into full compliance with the just-released HTML 4.0 specification and the still-under-development W3C DOM. That means that DHTML will finally become "cross-browser" compatible.

Sally Khudairi of W3C, sums up the present state of DHTML implementation pretty well: "DHTML is nothing more than a marketing term for generic animation and manipulation of multi-media events."

20: Stop The Presses!
Netscape's Erick Krock has provided exclusive to C|NET's BUILDER.COM, the "Ultimate JavaScript Client Sniffer: Determining Browser Vendor, Version, and Operating System with JavaScript" for inclusion, in total, for you here!

When creating a web page, you may wish to account for the possibility that viewers will be using browsers of various versions from various vendors. You may also wish to account for the possibility that certain page elements such as native plug-ins may not be available on all operating systems.

This sample code enables you to detect the browser's vendor, version number, and operating system. It has been tested on Navigator 2, 3, and 4, Internet Explorer 3 and 4, and Opera 3 on Windows 95, Windows NT, the Macintosh, and SunOS5. It is believed to be compatible with all JavaScript-capable browser versions on all platforms. It creates an object called "is" which has properties indicating the browser's vendor, version number, JavaScript version, and operating system.

After you have checked the browser vendor and version, you can dynamically generate optimized HTML markup, and your JavaScript code can conditionally branch to execute JavaScript code optimized for the dynamically generated page or the current browser's vendor or version number.

This tip is useful when developing a page with features not supported by all browsers. For example, users who are using Navigator 4 can view content that includes the new LAYER tag. Other users can view a "non-layers" version of the same file.
As another example, Navigator 4 and Internet Explorer 4 often implement the same Dynamic HTML functionality with different objects and methods. After checking the browser vendor, you can conditionally execute the correct JavaScript code fork for the current browser.

DYNAMICALLY GENERATING HTML MARKUP

When optimizing a page for multiple browsers, it is sometimes convenient to check the browser vendor and/or version and then use document.write() statements in <SCRIPT> elements in the page's BODY to dynamically generate HTML markup which is optimized for the current browser. Here is an example of the code necessary to dynamically generate vendor- or version-specific HTML markup. This code uses the "is" object defined by the sample JavaScript code below to check browser vendor and version.

```
<SCRIPT>
<!--
if (is.nav4)
{
   // document.write() statements to create Navigator 4-specific markup
}
else  (is.ie4)
{
   // document.write() statements to create IE 4-specific markup
}
if (is.nav3)
{
   // document.write() statements to create static HTML markup
}
// -->
</SCRIPT>
```

BROWSER- SPECIFIC JAVASCRIPT CODE FORKS

In your page's core JavaScript functions, you can then have vendor- and version-specific code forks where necessary. These code forks reference objects, properties, methods, and functions which are only available in specific browser versions. The code forks may also reference HTML page elements whose HTML markup is dynamically generated only for certain browser versions.

This code fragment also uses the "is" object defined by the sample JavaScript code below to check browser vendor and version.

```
if (is.nav4)
{
   // Navigator 4-specific JavaScript here
}
else  (is.ie4)
{
   // IE4-specific JavaScript here
}
if (is.nav3)
{
   // Navigator 3-specific JavaScript here
}
```

JAVASCRIPT CODE TO DETECT BROWSER VENDOR, VERSION, AND OPERATING SYSTEM

Here is the JavaScript code necessary to detect browser vendor, version number, and operating system. This code creates an object called "is" which has properties indicating the browser's vendor, version number, JavaScript version, and operating system. This code is believed to be compatible with all versions of all JavaScript-capable browsers on all platforms. It has been tested on the following operating systems and browser versions:

- Windows NT: Navigator 4, Navigator 3, and Navigator 2; Internet Explorer 3; Opera 3
- Windows 95: Navigator 4; Internet Explorer 4
- Macintosh: Navigator 4
- SunOS5: Navigator 3

```
// Ultimate client-side JavaScript client sniff.
// (C) Netscape Communications 1998.  Permission granted to reuse and distribute.

// Everything you always wanted to know about your JavaScript client
// but were afraid to ask ... "Is" is the constructor function for "is" object,
// which has properties indicating:
// (1) browser vendor:
//     is.nav, is.ie, is.opera
// (2) browser version number:
//     is.major (integer indicating major version number: 2, 3, 4 ...)
//     is.minor (float   indicating full  version number: 2.02, 3.01, 4.04 ...)
// (3) browser vendor AND major version number
//     is.nav2, is.nav3, is.nav4, is.ie3, is.ie4
// (4) JavaScript version number:
//     is.js (float indicating full JavaScript version number: 1, 1.1, 1.2 ...)
// (5) OS platform and version:
//     is.win, is.win16, is.win32, is.win31, is.win95, is.winnt, is.win98
//     is.os2
//     is.mac, is.mac68k, is.macppc
//     is.unix
//        is.sun, is.sun4, is.sun5, is.suni86
//        is.irix, is.irix5, is.irix6
//        is.hpux, is.hpux9, is.hpux10
//        is.aix, is.aix1, is.aix2, is.aix3, is.aix4
//        is.linux, is.sco, is.unixware, is.mpras, is.reliant
//        is.dec, is.sinix, is.freebsd, is.bsd
//     is.vms
//
// See http://home.kiss.de/~i_thum/JS_tutorial/bstat/navobj.html
// for a detailed list of userAgent strings.


function Is ()
{   // convert all characters to lowercase to simplify testing
    var agt=navigator.userAgent.toLowerCase()

    // *** BROWSER VERSION ***
    this.major = parseInt(navigator.appVersion)
    this.minor = parseFloat(navigator.appVersion)

    this.nav  = ((agt.indexOf('mozilla')!=-1) && ((agt.indexOf('spoofer')==-1)
                && (agt.indexOf('compatible') == -1)))
    this.nav2 = (this.nav && (this.major == 2))
    this.nav3 = (this.nav && (this.major == 3))
    this.nav4 = (this.nav && (this.major == 4))
    this.navonly      = (this.nav && (agt.indexOf(";nav") != -1))
```

```
this.ie   = (agt.indexOf("msie") != -1)
this.ie3  = (this.ie && (this.major == 2))
this.ie4  = (this.ie && (this.major == 4))

this.opera = (agt.indexOf("opera") != -1)

// *** JAVASCRIPT VERSION CHECK ***
// Useful to workaround Nav3 bug in which Nav3
// loads <SCRIPT LANGUAGE="JavaScript1.2">.
if (this.nav2 || this.ie3) this.js = 1.0
else if (this.nav3 || this.opera) this.js = 1.1
else if (this.nav4 || this.ie4) this.js = 1.2
// NOTE: In the future, update this code when newer versions of JS
// are released. For now, we try to provide some upward compatibility
// so that future versions of Nav and IE will show they are at
// *least* JS 1.2 capable. Always check for JS version compatibility
// with > or >=.
else if ((this.nav && (this.minor > 4.05)) || (this.ie && (this.major > 4)))
     this.js = 1.2
else this.js = 0.0 // HACK: always check for JS version with > or >=

// *** PLATFORM ***
this.win   = ( (agt.indexOf("win")!=-1) || (agt.indexOf("16bit")!=-1) )
// NOTE: On Opera 3.0, the userAgent string includes "Windows 95/NT4" on all
//        Win32, so you can't distinguish between Win95 and WinNT.
this.win95 = ((agt.indexOf("win95")!=-1) || (agt.indexOf("windows 95")!=-1))

// is this a 16 bit compiled version?
this.win16 = ((agt.indexOf("win16")!=-1)
           || (agt.indexOf("16bit")!=-1) || (agt.indexOf("windows 3.1")!=-1)
           || (agt.indexOf("windows 16-bit")!=-1) )

this.win31 = (agt.indexOf("windows 3.1")!=-1) || (agt.indexOf("win16")!=-1) ||
             (agt.indexOf("windows 16-bit")!=-1)

// NOTE: Reliable detection of Win98 may not be possible. It appears that:
//       - On Nav 4.x and before you'll get plain "Windows" in userAgent.
//       - On Mercury client, the 32-bit version will return "Win98", but
//         the 16-bit version running on Win98 will still return "Win95".
this.win98 = ((agt.indexOf("win98")!=-1)||(agt.indexOf("windows 98")!=-1))
this.winnt = ((agt.indexOf("winnt")!=-1)||(agt.indexOf("windows nt")!=-1))
this.win32 = this.win95 || this.winnt || this.win98 ||
             ((this.major >= 4) && (navigator.platform == "Win32")) ||
             (agt.indexOf("win32")!=-1) || (agt.indexOf("32bit")!=-1)

this.os2   = (agt.indexOf("os/2")!=-1)
             || (navigator.appVersion.indexOf("OS/2")!=-1)
             || (agt.indexOf("ibm-webexplorer")!=-1)

this.mac    = (agt.indexOf("mac")!=-1)
this.mac68k = this.mac && ((agt.indexOf("68k")!=-1) ||
                           (agt.indexOf("68000")!=-1))
this.macppc = this.mac && ((agt.indexOf("ppc")!=-1) ||
                           (agt.indexOf("powerpc")!=-1))

this.sun   = (agt.indexOf("sunos")!=-1)
this.sun4  = (agt.indexOf("sunos 4")!=-1)
this.sun5  = (agt.indexOf("sunos 5")!=-1)
this.suni86= this.sun && (agt.indexOf("i86")!=-1)
this.irix  = (agt.indexOf("irix") !=-1)    // SGI
this.irix5 = (agt.indexOf("irix 5") !=-1)
this.irix6 = ((agt.indexOf("irix 6") !=-1) || (agt.indexOf("irix6") !=-1))
this.hpux  = (agt.indexOf("hp-ux")!=-1)
this.hpux9 = this.hpux && (agt.indexOf("09.")!=-1)
this.hpux10= this.hpux && (agt.indexOf("10.")!=-1)
this.aix   = (agt.indexOf("aix")  !=-1)      // IBM
this.aix1  = (agt.indexOf("aix 1")  !=-1)
this.aix2  = (agt.indexOf("aix 2")  !=-1)
this.aix3  = (agt.indexOf("aix 3")  !=-1)
this.aix4  = (agt.indexOf("aix 4")  !=-1)
```

```
    this.linux = (agt.indexOf("inux")!=-1)
    this.sco   = (agt.indexOf("sco")!=-1) || (agt.indexOf("unix_sv")!=-1)
    this.unixware = (agt.indexOf("unix_system_v")!=-1)
    this.mpras    = (agt.indexOf("ncr")!=-1)
    this.reliant  = (agt.indexOf("reliantunix")!=-1)
    this.dec   = (agt.indexOf("dec")!=-1) || (agt.indexOf("osf1")!=-1)
           || (agt.indexOf("dec_alpha")!=-1) || (agt.indexOf("alphaserver")!=-1)
           || (agt.indexOf("ultrix")!=-1) || (agt.indexOf("alphastation")!=-1)
    this.sinix = (agt.indexOf("sinix")!=-1)
    this.freebsd = (agt.indexOf("freebsd")!=-1)
    this.bsd = (agt.indexOf("bsd")!=-1)
    this.unix  = (agt.indexOf("x11")!=-1) || this.sun || this.irix || this.hpux ||
                 this.sco ||this.unixware || this.mpras || this.reliant ||
                 this.dec || this.sinix || this.aix || this.linux || this.freebsd

    this.vms   = (agt.indexOf("vax")!=-1) || (agt.indexOf("openvms")!=-1)
}

var is = new Is()
```

PROFILE OF YOUR BROWSER

Here are the results of running that JavaScript code on the browser you are using. The below text has been dynamically generated after checking your browser vendor, version, and operating system from JavaScript.

Basic Data

```
navigator.appName Microsoft Internet Explorer
navigator.userAgentMozilla/4.0 (compatible; MSIE 4.01; MSN 2.5; MSN 2.5; Windows 98)
navigator.appVersion4.0 (compatible; MSIE 4.01; MSN 2.5; MSN 2.5; Windows 98)
```

Version Number

```
major:4
minor:4
```

Browser Version

```
nav:false
nav2:false
nav3:false
nav4:false
navonly:false
ie:true
ie3:false
ie4:true
opera:false
```

JavaScript Version

```
js:1.2
```

OS

```
win:true
win16:false
win31:false
win32:true
win95:false
win98:true
winnt:false
os2:false
mac:false
mac68k:false
macppc:false
unix:false
sun:false
sun4:false
sun5:false
suni86:false
irix:false
```

```
irix5:false
irix6:false
hpux:false
hpux9:false
hpux10:false
aix:false
aix1:false
aix2:false
aix3:false
aix4:false
linux:false
sco:false
unixware:false
mpras:false
reliant:false
dec:false
sinix:false
bsd:false
freebsd:false
vms:false
```

21: Stop the Presses, Part II!

Eric Krock isn't finished with you yet! Here's another "Hot off the Netscape Presses" C|NET BUILDER.COM exclusive for you!

Once you've detected the browser version, you often need to conditionally write out HTML markup that's appropriate for the particular browser version. That can get to be a lot of document.write() statements and version checks. Here's a little hack you might enjoy:

To save a bit of typing, here's a simple function to write out a string which (1) has a shorter name that document.write(), so it saves typing, and (2) only writes out the code if the current browser version is >= the second (optional) parameter and <= the third (optional) parameter. This function and the next rely on the is global object defined by the code I just submitted.

```
// ---------------------------------
// convenience function to save typing out document.write("STRING");
// if optional second argument version passed in, only write if >= that version;
// if optional third argument maxVersion passed in, only execute if <= that version;
function dw(str, version, maxVersion)
{ if ( ((dw.arguments.length < 3) || (is.major <= maxVersion))
&& ((dw.arguments.length < 2) || (is.major >= version)))
document.write(str) }
// ---------------------------------
```

Likewise, if you're concatenating a long string of markup depending on browser version and want to optionally insert some markup into the string, here's a function that returns the string or "" depending on browser version:

```
// ---------------------------------
// string version:
// convenience function to return str or "" depending on version;
// if optional second argument version passed in, only return str if >= that version;
// if optional third argument maxVersion passed in, only return str if <= that version;
function sv(str, version, maxVersion)
{ if ( ((sv.arguments.length < 3) || (is.major <= maxVersion))
&& ((sv.arguments.length < 2) || (is.major >= version)))
return str;
else return ""
```

```
}
// --------------------------------
```

Obviously, these two functions deal only with version number, but you could extend them to check browser vendor as well. Eric Krock leaves this exercise to you for implementation and experimentation.