The Dynamic Duo
Cross-Browser Dynamic HTML

Overview

Dynamic HTML

Dynamic HTML (DHTML) is an all-in-one word for web pages that use Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), and rely on JavaScript to make the web pages interactive. DHTML is a feature of Netscape Communicator 4.0, and Microsoft Internet Explorer 4.0 and 5.0 and is entirely a "client-side" technology. It relies only the browser for the display and manipulation of the web pages and is unrelated to other client-side technologies like Java, Flash.

Someone once asked me for a non-technical definition of DHTML, my reply was:

"A way to build web interfaces by using the built-in capabilities of Netscape and Internet Explorer"

DHTML excels in creating low-bandwidth effects that enhance a web page's functionality. It can be used to create animations, games, applications, provide new ways of navigating through web sites, and create out-of-this world page layouts that simply aren't possible with just HTML. Although many of features of DHTML can be duplicated with either Flash or Java, DHTML provides an alternative that does not require plugins and embeds seamlessly into a web page.

Although the underlying technologies of DHTML (HTML, CSS, JavaScript) are standardized, the manner in which Netscape and Microsoft have implemented them differ dramatically. For this reason, writing DHTML pages that work in both browsers (referred to as cross-browser DHTML) can be a very complex issue.

Links for more DHTML information:

Microsoft DHTML Documentation
http://msdn.microsoft.com/workshop/author/default.asp

Netscape DHTML Documentation
http://developer.netscape.com/docs/manuals/communicator/dynhtml/index.htm

Cascading Style Sheets

Cascading Style Sheets (CSS) is an addition to HTML that gives developers a sophisticated manner to structure web pages. It does this by separating the content of a web page (the text) from the display (the colors, styles, and positioning).

Cascading Style Sheets Positioning (CSSP) is an extension to CSS that

allows pixel-level control over the position of HTML elements.

Links for more CSS information:

W3C CSS-Positioning
http://www.w3.org/TR/WD-positioning.html

Builder.com's CSS Guide
http://builder.cnet.com/Authoring/CSS/index.html

JavaScript

Contrary to its name, JavaScript is very much unrelated to Java. JavaScript is scripting language built into web browses that controls HTML elements, whereas Java is a high-level programming language for building cross-platform applications (among other things like Applets which are Java programs that can be displayed in a web page).

JavaScript first appeared in Netscape 2.0, and was primarily for scripting the contents of a web page, and providing added functionality to HTML forms, frames, and windows. Netscape 3.0 added more features like image rollovers and audio/video controls. Microsoft Internet Explorer 3.0 (released shortly after Netscape 3.0) also implemented JavaScript, but marketed it as JScript which is essentially the same as JavaScript with a few minor incompatibilities that Microsoft threw in to lure developers into using their version of JavaScript.

Extensions to JavaScript were added in Netscape 4.0 and Internet Explorer 4.0 to give developers a way to manipulate DHTML (HTML elements that use CSS). However these extensions were not standardized before the release of the 2 browsers. And as a result we now have two versions of JavaScript that are largely incompatible.

Links for more JavaScript information:

Netscape JavaScript Guide
http://developer.netscape.com/docs/manuals/communicator/jsguide4/index.htm

JavaScript Reference
http://developer.netscape.com/docs/manuals/communicator/jsref/index.htm

Microsoft JScript
http://msdn.microsoft.com/scripting/default.htm

The Dynamic Duo

The Dynamic Duo, is a tutorial written by me, Dan Steinman, and is the result of my experimentation and successes in creating Cross-Browser DHTML.

This tutorial focuses primarily on the JavaScript issues involved in DHTML. It only covers the portions of CSSP and JavaScript that can be used in both Netscape and Internet Explorer. By no means does this tutorial cover everything, or necessarily offer the best solutions to a

particular task, but rather the things that I have tried and have had good results with.

If you are not unfamiliar with JavaScript and CSS, this tutorial may not be the best starting point for you. However, I do start out slowly and cover much of the ground knowledge needed to understand how DHTML works. The programming concepts in this this tutorial are not extremely complex. However, cross-browser DHTML requires a level of debugging skills that can be quite daunting to a beginner. You will be working with browsers that are only partially compatible, and computer languages that are only partially implemented. You will encounter bugs and limitations not only between the 2 browsers, but between different operating systems, as well as between incremental versions of the browsers. This tutorial only scratches the surface of the problems that you will encounter with building your own DHTML pages... and do believe me on this one, I am not kidding.

With that said, I have done my best to lay down a set of guidelines that makes cross-browser DHTML feasible. Following the tips and techniques in this tutorial you can create just about anything you can think of. By time you are finished reading, you will understand nearly all the concepts involved in DHTML, learn a rich set of JavaScript programming techniques using my DHTML API (The DynLayer), as well as learn how to build your very own DHTML objects for creating reusable widgets and components for your website.

I'd appreciate any comments or suggestions you have about this tutorial, I'm always looking for ways to improve it. Also, I'd be very gracious of any contributions you have, including modifications to any code in this tutorial, or any DHTML objects that you've created and want to share with other people. And if you have built a website, game, or application using this tutorial, I would be more than happy to provide a link from my DHTML Resources links page.

Cascading Style Sheets Positioning

Cascading Style Sheets (CSS) are the basis for Dynamic HTML in both Netscape Navigator 4.0 and Internet Explorer 4.0. CSS allows a way to create a set of "styles" that define how the elements on your page are rendered. Cascading Style Sheets Positioning (CSS-P) is an extension to CSS that gives a developer pixel-level control over the location of anything on the screen. Due to the fact there are already good CSS and CSS-P documentation and tutorials I won't be duplicating them - rather I'll be building on top of them.

Here are 2 documents/tutorials that explain the syntax of CSS and CSS-P:

W3C CSS-Positioning
http://www.w3.org/TR/WD-positioning.html

Builder.com's CSS Guide
http://builder.cnet.com/Authoring/CSS/index.html

Those sites will give a complete overview of CSS and how to implement it. But I'll just quickly re-iterate the parts of CSS that will be used throughout this tutorial.

Using DIV Tags:

When using CSS-Positioning, these properties are usually applied to the DIV (division) tag - an empty, non-formatting tag, that is best suited for CSS. When you put HTML/text into a DIV tag it is commonly referred to as one of: "DIV block", "DIV element", "CSS-layer", or as I say, just a "layer". When you read about Dynamic HTML on websites or in newsgroups, if someone is talking about any of these terms they're all talking about the same thing - some piece of HTML that is inside a positioned "DIV" tag.

To markup an empty DIV tag is no different than any other tag:

<DIV> This is a DIV tag </DIV>

Using DIV tag by itself has the same results as using <P></P>

But by applying CSS to DIV tags we can define where on the screen this piece of HTML will be displayed, draw squares or lines, or how to display the text that's inside it. You do this by first giving the DIV an ID (sort of like a name):

<DIV ID="truck"> This is a truck </DIV>

What you use for your ID is up to you. It can be any set of characters (a-z,A-Z,0-9, and underscore), but starting with letter.

Then you apply your CSS in one of 2 ways:

Inline CSS:

Inline is the way most commonly used. And it is the way I will begin showing how to write DHTML and JavaScript.

<DIV ID="truck" STYLE="styles go here"> This is a truck </DIV>

External STYLE tag:

Using the external method will work as well, however there are a few issues involved with writing CSS like this, so I suggest you wait until you get to the Nesting Layers lesson before trying it on your own. For right now just take a look to see how it is done...

<STYLE TYPE="text/css"> <!-- #truck {styles go here} --> </STYLE>

<DIV ID="truck"> This is a truck </DIV>

Notice how the ID is used in the STYLE tag to assign the CSS styles.

Cross-Browser CSS Properties:

Because the goal of this site is to produce DHTML that works in both Netscape and Internet Explorer, we are somewhat limited to which CSS styles/properties we can use. Generally, the following properties are the ones that work (fairly closely) to the standards as defined by the W3C.

position: Defines how the DIV tag will be positioned - "relative" means that the DIV tag will flow like any other HTML tag, whereas "absolute" means the DIV will be positioned at specific coordinates. Absolute positioning will be the topic of most of this tutorial.
left: Left location (the number of pixels from the left edge of the browser window).
top: Top location (the number of pixels from the top edge of the browser window).
width: Width of the DIV tag. Any text/html that is inserted into the DIV will wrap according to what this value is. If width is not defined it will all be on one line.
Important: When using layers for animation you should always define the width. This is because in IE the default is the entire width of the screen. If you move the layer around the screen a scrollbar will appear at the bottom, which is annoying and causes the animation to slow down.
height: Height of the DIV tag. This property is rarely needed unless you also you want to clip the layer
clip: Defines the clipping (crop) rectangle for the layer. Makes the DIV into a precisely defined square. You define the size of the rectangle with the values of the four edges: clip:rect(top,right,bottom,left);
visibility: Determines whether the DIV will be "visible", "hidden", or "inherit" (default).
z-index The stacking order of DIV tags.
background-color: Background color of the DIV. In Netscape this property only applies to the background color of the text. When you want to draw squares with CSS you must also define the layer-background-color property to the same value.
layer-background-color: Background color of the DIV for Netscape.

background-image: Background image for Internet Explorer. In Netscape this property only applies to the background-image for the text.
layer-background-image: Background image of the DIV for Netscape.

The syntax for CSS differs from HTML, you use colons to separate the property and it's value, and semi-colons to separate the different properties:

```
position: absolute;
left: 50px;
top: 100px;
width: 200px;
height: 100px;
clip: rect(0px 200px 100px 0px);
visiblity: visible;
z-index: 1;
background-color:#FF0000;
layer-background-color:#FF0000;
background-image:URL(filename.gif);
layer-background-image:URL(filename.gif);
```

You have a bit of flexibility when assigning CSS properties. You do not have to define all of them. White space is ingored so you can either have them all on the same line, or on separate lines, tabs between values etc. As well, the default unit value is pixels, so you do not necessarily have to have the "px" after the left, top, width and height values, although it is recommended to do so.

```
position:absolute; left:50px; top:100px; width:200px; height:100px;
clip:rect(0px 200px 100px 0px); background-color:#FF0000;
layer-background-color:#FF0000;
```

Inline Example:

```
<DIV ID="divname" STYLE="position: absolute; left:50px; top:100px;
width:200px; height:100px; clip:rect(0px 200px 100px 0px);
visiblity:visible; z-index:1;"> </DIV>
```

External Example:

```
<STYLE TYPE="text/css">
<!-- #divname {position: absolute; left:50px;
top:100px; width:200px; height:100px; clip:rect(0px 200px 100px 0px);
visiblity:visible; z-index:1;} -->
</STYLE>

<DIV ID="divname"> </DIV>
```

Cross-Browser JavaScript

You can use JavaScript to access and change the properties of your CSS-P element. However, some of the syntax differs between Netscape 4.0 and Internet Explorer 4.0. By knowing where the differences lie, I'll show you an easy way to create cross-browser JavaScripts - scripts that will work in both N4 and IE4.

Browser Checking:

I'm now using ns4 and ie4 for browser checking instead of n and ie

First things first: we have to know how to check which browser someone is using. This little chunk of code will be the standard browser check in nearly all the examples in this tutorial:

```
ns4 = (document.layers)? true:false
ie4 = (document.all)? true:false
```

The document.layers object is specific to Netscape 4.0, while the document.all object is specific to IE 4.0. So by checking if the object exists we can create the boolean variables ns4 (for Netscape 4.0) and ie4 (for Internet Explorer 4.0) and assign them true or false depending on which browser is being used. Now whenever you need to check which browser someone is using you just have to use if (ns4) or if (ie4):

```
function check() {
     if (ns4) {
      // do something in Netscape Navigator 4.0
     }
     if (ie4) {
     // do something in Internet Explorer 4.0
     }
 }
```

Using JavaScript and CSS-P:

Say we had a DIV tag that looked like this:

```
<DIV ID="blockDiv" STYLE="position:absolute; left:50; top:100;
width:30;"> <IMG SRC="block.gif" WIDTH=30 HEIGHT=30 BORDER=0> </DIV>
```

Remember that this is an example, you can rename blockDiv to whatever you want and it will still work exactly the same.

For Netscape the general way to access the CSS-P properties is like this:

```
document.blockDiv.propertyName
```

or

document.layers["blockDiv"].propertyName

And then for Internet Explorer it's:

blockDiv.style.propertyName

or

document.all["blockDiv"].style.propertyName

Where propertyName can be any one of left, top, visibility, zIndex, width, or any of the other CSS-P properties.

The Cross-Browser Method (Pointer Variables):

I've found that the best way to make cross-browser scripts is to have a variable, that depending on whether you're in Netscape or IE, points directly to either document.blockDiv or blockDiv.style, look below. I call these variables, pointer variables.

```
if (ns4) block =  document.blockDiv
if (ie4) block =  blockDiv.style
```

You see, after you do this, you can now access the properties using a shortcut way. For example if you wanted to check the left coordinate of our DIV tag called "blockDiv", it would simply be:

block.left

It doesn't matter which browser is used because for Netscape, block points to document.blockDiv, and in IE, block points to blockDiv.style.

Aside: Throughout this tutorial I will be naming my DIV tags with a "Div" on the end of them (squareDiv, blockDiv etc.). This is because when you initialize a layer using the pointer variable method, you have to choose a variable name that is totally unique - it cannot be the same name as one of your DIV tags. I just make it a standard in my code that all layers that are going to be initialized with pointer variables automatically have a "Div" and I make the pointer variable name without the "Div" - because as you'll see you end up using the pointer variable many more times than the name of the layer itself.

A Full Example:

This example will pop up an alert of the left, top and visiibilty properties of a CSS-P element.

The script:

```
<SCRIPT LANGUAGE="JavaScript">
<!-- ns4 = (document.layers)? true:false
ie4 = (document.all)? true:false

function init() {
```

```
        if (ns4) block = document.blockDiv
        if (ie4) block =  blockDiv.style
}

//-->
</SCRIPT>
```

The HTML:

```
<BODY onLoad="init()">

<A HREF="javascript:alert(block.left)">left</A> -
<A HREF="javascript:alert(block.top)">top</A> -
<A HREF="javascript:alert(block.visibility)">visibility</A>

<DIV ID="blockDiv" STYLE="position:absolute; left:50px; top:100px;
width:30px; height:30px; clip:rect(0px 30px 30px 0px);
background-color:red; layer-background-color:red;"> </DIV>

</BODY>
```

Important: I call the init() function in the BODY onLoad="" so that it will execute after the rest of the page is completed loading. This is because when defining your pointer variable, the DIV tag must already exist. If you try and define the variable before the page is done loading you'll recieve a JavaScript error like "block is not defined".

The Differences

If you open up both Netscape and IE and try that last example in each, you'll notice that you don't recieve the same values.

Property	Netscape 4 Value	IE 4 Value
left	50	50px
top	100	100px
visibility	show	visible

These differences can cause some problems but in the next few sections I'll show how to get around them.

Showing and Hiding

You might ask yourself: "Why does Netscape display the visibility as 'show'?"

Well the answer is that Netscape's CSS properties are based around it's proprietory LAYER tag. However, even Netscape is now downplaying it's LAYER tag in favour of the W3C's recommended CSS-P. So the "show" and corresponding "hide" values of the visibility property are left-overs from Netscape's layers. I believe this is the only glaring defect in how Netscape represents CSS-P.

Until there is a unified standard you'll usually have to write separate code to hide a particular element.

For Netscape

To show an element in Netscape you have to use:

```
document.divName.visibility = "show"
```

and to hide it's:

```
document.divName.visibility = "hide"
```

For Internet Explorer

To show an element in Internet Explorer you have to use:

```
divName.style.visibility = "visible"
```

and to hide it's:

```
divName.style.visibility = "hidden"
```

Generic Show and Hide Functions

Instead of always rewriting the same code over and over again to show and hide elements, you can use the following functions:

```
function showObject(obj) {
        if (ns4) obj.visibility = "show"
        else if (ie4) obj.visibility = "visible"
}

function hideObject(obj) {
        if (ns4) obj.visibility = "hide"
        else if (ie4) obj.visibility = "hidden"
}
```

These functions must be used along with pointer variables - see the code in the following example.

Whenever you want to change the visibility of an element, you just go:

showObject(objectName)

or

hideObject(objectName)

Where objectName is your pointer variable to a particular DIV tag.

Show/Hide Functions Without Pointer Variables

I've been finding that it's not always necessary, and sometimes cumbersome if you have a lot of layers that need only to be hidden and shown. I've showed the pointer variable technique first because that is general idea that I'll be building on to make more powerful JavaScripts in further lessons. But on occasions where you won't need to have any more functionality, the following simplified functions can also be used:

```
// Show/Hide functions for non-pointer layer/objects

function show(id) {
        if (ns4) document.layers[id].visibility = "show"
        else if (ie4) document.all[id].style.visibility = "visible"
}

function hide(id) {
        if (ns4) document.layers[id].visibility = "hide"
        else if (ie4) document.all[id].style.visibility = "hidden"
}
```

To use these are similar except now the exact name of the layer must be used and it also must be in quotes:

show("divID")

or

hide("divID")

Where divID is the ID of the DIV tag that you want to show/hide.

Moving

There is generally no compatibility problems when assigning a new location for your CSS-P element.

To move an element named "myelement" to the coordinate (100,50), you simply assign new left and top values:

```
myelement.left = 100 myelement.top = 50
```

But don't forget that myelement must be a pointer variable defined something like this:

```
function init() {
        if (ns4) myelement = document.myelementDiv
        if (ie4)  myelement =  myelementDiv.style
}
```

From now on, this will be inherent in all examples so don't forget!

As I said, there is no compatibility issues with assigning a new location, however there is a problem when capturing the current location of an element. It's due to fact that IE stores it's locations with a "px" at the end of the values (as seen in the Cross-Browser Javascript example).

To get rid of the "px" you can parse the value into an integer.

So instead of just writing

```
myelement.left
```

You have to write

```
parseInt(myelement.left)
```

For example, if you wanted to pop up an alert of the current left and top location you'd write:

```
alert(parseInt(myelement.left) + ", " + parseInt(myelement.top))
```

Adding New Properties

Now believe me, to always have to write parseInt() before all your variables will tend to get very annoying. You will soon ask yourself if there is a better way... and I think I have a pretty good answer to that.

There is nothing stopping you from adding more properties onto our pointer variable, or object, as I will tend to call it from now on.

What I suggest you do, is keep the current location of the element in separate properties aside from the left and top properties. To make

these new properties you just directly assign them. I'd start by setting them to actual location:

```
myelement.xpos = parseInt(myelement.left)
myelement.ypos = parseInt(myelement.top)
```

Now after this point, if you ever need to find out the left or top position, you just check the value of myelement.xpos and myelement.ypos respectively. Our new alert() would look like so:

```
alert(myelement.xpos + "," + myelement.ypos)
```

And The Catch?

When you want to change the location of the element, you FIRST have to change the values of xpos and ypos. THEN you set the left and top values equal to xpos and ypos respectively. For example:

```
function move() {
        myelement.xpos = 200
        myelement.ypos = -40
        myelement.left = myelement.xpos
        myelement.top = myelement.ypos
}
```

You must always keep the xpos and ypos values in synch with the left and top values. That way when you check myelement.xpos, you know that it will always be the same as myelement.left.

Not too difficult right? This idea will be the basis for everything I'll show in future examples. It may seem a little dumb to have these extra variables but once you get into more complicated things you'll find this technique does help smooth out your code.

Aside: You may be wondering why am using xpos and ypos as my properties instead of just x and y... Well I did that for a reason. It is a little known fact that Netscape has already included these properties into CSS-P. I found that if you use x and y then your values will always be stored as integers. Now you may think "who cares?"... but there are instances where you need to store the current left and top positions with more than just integers (ie. real numbers with decimals and everything) and this is just not possible if you use x and y.

Generic Move Functions

In that last example I "hard-coded" the movements - I wrote separate functions for each movement. Now of course if you want to move many different layers to various locations you don't always want to keep writing more functions. So what we can do is create some generic functions that will take care of most types of movements.

The moveTo() Function

The moveTo() function takes your layer/object and moves it directly to a

new location.

```
function moveTo(obj,x,y) {
        obj.xpos = x
        obj.left = obj.xpos
        obj.ypos = y
        obj.top = obj.ypos
}
```

To use the function is really easy - all you do is tell it what layer/object to use and the new x and y locations. For example, if you initialize your layer with:

```
if (ns4) mysquare = document.mysquareDiv
if (ie4) mysquare = mysquareDiv.style
mysquare.xpos = parseInt(mysquare.left)
mysquare.ypos = parseInt(mysquare.top)
```

Then to move the square to a new location you'd write:

```
moveTo(mysquare,50,100)
```

The moveBy() Function

MoveBy works exactly the same but instead of moving it directly to a new location it shifts the layer by a given number of pixels.

```
function moveBy(obj,x,y) {
        obj.xpos += x
        obj.left = obj.xpos
        obj.ypos +=y
        obj.top = obj.ypos
}
```

To shift mysquare 5 pixels right, and 10 pixels up you'd write:

```
moveBy(mysquare,5,-10)
```

Sliding

Sliding is just what I call a animated movement or scrolling effect. By using looping (or iterating) functions and moving the layer in small increments, you can put together any sort of animated movement you can think of.

The basic idea is that you have your movement code:

```
block.xpos += 5 block.left = block.xpos
```

Which moves the layer 5 pixels to the right. Then you stick that code into a looping function:

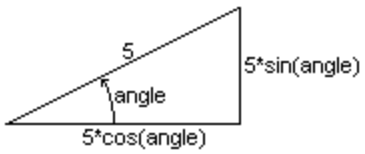
```
function slide() {
        if (block.xpos < 300) {
                block.xpos += 5
                block.left = block.xpos
                setTimeout("slide()",30)
        }
}
```

The if statement is there to determine when to stop the animation. In this case the function will stop when the x-position is at or over 300 pixels. The setTimeout() is what creates the loop. After a certain amount of milliseconds, it will execute whatever's inside the quotes. So this function will repeat itself every 30 milliseconds.

Of course it's not much more difficult to move on a diagonal - you just change both the xpos (left) and the ypos (top) values.

Moving at a Given Angle

Using some high school trigonometry we can figure out how to move an element on any angle. In case you forgot, here's a quick diagram to refresh your memory:



Now to initialize your object to include angles you'll need 4 new properties:

```
object.angle = 30
object.xinc = 5*Math.cos(object.angle*Math.PI/180)
object.yinc = 5*Math.sin(object.angle*Math.PI/180)
object.count = 0
```

We calculate the x and y incrementation and use them to determine how far to move the left and top values. You have to multiply the angle by Math.PI/180 to convert the angle into radians - sin and cos are always calculated in radians. The count property will be used in the iterating function to determine how many times to loop.

Using my block example again, here's some full code to move an element at a given angle.

```
function init() {
        if (ns4) block = document.blockDiv
        if (ie4) block = blockDiv.style
        block.xpos = parseInt(block.left)
        block.ypos = parseInt(block.top)
        block.angle = 30
        block.xinc = 5*Math.cos(block.angle*Math.PI/180)
        block.yinc = 5*Math.sin(block.angle*Math.PI/180)
        block.count = 0
  }

  function slide() {
        if (block.count < 25) {
              block.xpos += block.xinc
              block.ypos -= block.yinc
              block.left = block.xpos
              block.top = block.ypos
              block.count += 1
              setTimeout("slide()",30)
        }
        else block.count = 0
  }
```

The if (block.count < 25) means that the function will execute 25 times before stopping - ie4. the block will slide a total 125 pixels units.

Mouse Click Animation

Using clever mouse events we can use a single hyperlink to start and stop an animation. When pressed the hyperlink will slide a block and when released the slide will stop.

The slide script is nothing new. We'll need an active variable in there again, and the move function is a carbon copy of previous functions:

```
function init() {
        if (ns4) block = document.blockDiv
        if (ie4) block = blockDiv.style
        block.xpos = parseInt(block.left)
        block.active = false
  }

  function slide() {
        if (block.active) {
              block.xpos += 5
              block.left = block.xpos
              setTimeout("slide()",30)
        }
  }
```

The trick is with what I do with the hyperlink:

```
<A HREF="javascript:void(null)" onMouseDown="block.active=true; slide();
return false;" onMouseUp="block.active=false"
onMouseOut="block.active=false">move</A>
```

The onMouseDown sets the active variable to true, and then calls the slide() function which begins our animation. While the link is held, nothing changes. It continues to slide until you release the hyperlink - and hence execute whatever is in the onMouseUp handler. It sets the active variable to false which stops the slide.

The onMouseOut also sets the active variable to false for error proofing reasons. I found that if you move the mouse off the link and then release, it wouldn't stop the animation - because you're not executing an MouseUp over the link. But if you include the onMouseOut it accounts for this loop-hole.

Keystroke Events

Capturing keystrokes is the most powerful type of interaction you have at your disposal. You can have total control over (almost) any key that has been pressed or released. Note however, Netscape did not include the ability to capture Keystroke events into the Unix versions of Communicator 4.0. If you're planning on using keystrokes in a JavaScript game it will not be playable on any version of Unix including Linux.

The first thing that you have to understand is how to initialize your events. Here is a basic initialization for the "onkeydown" event.

```
document.onkeydown = keyDown
```

When this code is read by the browser it will know that whenever a key is pressed, the keyDown() function will be called. It doesn't matter what function you call, and the code does not need the brakets after the funtion name.

To capture what key was pressed works a little bit differently between the browsers. So I'll first show each individually.

Netscape

Netscape is a little more picky than IE is with respect to event handling. You have to put an extra line in to tell Netscape to always check for the keydown event. If you don't have this line, it will mess up when other events like mousedown occur.

```
document.onkeydown = keyDown if (ns4)
document.captureEvents(Event.KEYDOWN)
```

Your keyDown() has to pass a hidden variable - I'll use the letter "e" because that is what's commonly used.

```
function keyDown(e)
```

This "e" represents the key that was just pressed. To find out out what key that is, you can use the which property:

```
e.which
```

This will give the index code for the key - not what letter or number was pressed. To convert the index to the letter or number value, you use:

```
String.fromCharCode(e.which)
```

So putting it all together, we can make a function that pops up a message telling the keycode and the real key values of the key that was pressed:

```
function keyDown(e) {
```

```
        var keycode = e.which
        var realkey = String.fromCharCode(e.which)
        alert("keycode: " + keycode + "\nrealkey: " + realkey)
  }

  document.onkeydown = keyDown
  document.captureEvents(Event.KEYDOWN)
```

Internet Explorer

IE works similarly except you don't need to pass the "e" value.

Instead of using e.which, you use window.event.keyCode.

And conversion to the real key value is the same: String.fromCharCode(event.keyCode).

```
function keyDown() {
        var keycode = event.keyCode
        var realkey = String.fromCharCode(event.keyCode)
        alert("keycode: " + keycode + "\nrealkey: " + realkey)
  }

  document.onkeydown = keyDown
```


```
  document.onkeydown = keyDown
```


Combining the Two

Now, if you were to open both browsers and compare the examples, you'll realize the results are not always the same. The keycodes are different because each browser uses a different character set. Because of this you'll always have to make separate code for each browser - there's no way around it.

What I'd suggest is totally forgetting about the real key values entirely, and only work with the keycodes. The following chunk of code will assign nKey to the keycode and ieKey to 0 if you're using Netscape or or it will set ieKey to the keycode and nKey to 0 if you're using Internet Explorer. Then it shows an alert of both values:

```
function keyDown(e) {
        if (ns4) {var nKey=e.which; var ieKey=0}
        if (ie4) {var ieKey=event.keyCode; var nKey=0}
        alert("nKey:"+nKey+" ieKey:" + ieKey)
  }

  document.onkeydown = keyDown
  if (ns4) document.captureEvents(Event.KEYDOWN)
```

Now on to the good stuff....

Moving Elements with the Keyboard

Now you can activate your movement functions from the keyboard. You do a check of which key was pressed, and then call the appropriate function to move your object. For the following example I use the "A" key to initiate a sliding function. For the "A" key, the nKey value is 97, and the ieKey is 65. So I do a check for those values in order to call the "slide" function.

```
function init() {
        if (ns4) block = document.blockDiv
        if (ie4) block = blockDiv.style
        block.xpos = parseInt(block.left)

        document.onkeydown = keyDown
        if (ns4) document.captureEvents(Event.KEYDOWN)
  }

  function keyDown(e) {
        if (ns4) {var nKey=e.which; var ieKey=0}
        if (ie4) {var ieKey=event.keyCode; var nKey=0}
        if (nKey==97 || ieKey==65) {    // if "A" key is pressed
              slide()
        }
  }

  function slide() {
        block.xpos += 5
        block.left = block.xpos
        status = block.xpos       // not needed, just for show
        setTimeout("slide()",30)
  }
```

Understanding "Active" Variables

That last script is somewhat limited. After the movement is started, there's no way to stop it, and if you hit the key several times it moves faster and faster. So we'll have fix that up.

I've developed a technique of using what I call "active" variables to represent the current state of movement... is it moving? or is it not moving? Once you get used to working with them, they can be very handy. Because most movement functions are recursive, they have no built in way of stopping, and that's where the active variables come into play. By inserting the appropriate "if" statment into the slide function, you can have control of whether that function will repeat or not. Usually you make the function something like this:

```
function slide() {
        if (myobj.active) {
              myobj.xpos += 5
              myojb.left = myobj.xpos
              setTimeout("slide()",30)
        }
  }
```

In this case, the slide() function will only operate when the myobj.active value is true. Once you set myobj.active to false the movement function will stop. Knowing this, we can insert some code into our script that will give us more control of what's happening.

Using onKeyUp and "Active" Variables

The onkeyup event works exactly the same way the onkeydown did. You can initialize both keydown and keyup with the following:

```
document.onkeydown = keyDown
document.onkeyup = keyUp
if (ns4) document.captureEvents(Event.KEYDOWN | Event.KEYUP)
```

And the keyUp() function is the same too. But we want to make so that when a key is released, it will stop whatever movement is currently running. To do that we can set our block's active variable to 0:

```
function keyUp(e) {
        if (ns4) var nKey = e.which
        if (ie4) var ieKey = window.event.keyCode
        if (nKey==97 || ieKey==65) block.active = false
  }
```

But to totally "error" proof our code, we have to put some more checks into the other functions. Take a look at the code below and see if you can understand what I'm doing. In the keyDown function, the && !block.active is to make sure that we can only call the function if the block is not active. In other words, if the block is moving we do not execute the slide() function again. Then we set the active value to true and move the block. The slide() function has the if (block.active) statement so that it only moves when the block.active value is true - that way when we release a key it will stop executing.

```
function init() {
        if (ns4) block = document.blockDiv
        if (ie4) block = blockDiv.style
        block.xpos = parseInt(block.left)
        block.active = false

        document.onkeydown = keyDown
        document.onkeyup = keyUp
        if (ns4) document.captureEvents(Event.KEYDOWN | Event.KEYUP)
  }

  function keyDown(e) {
        if (ns4) {var nKey=e.which; var ieKey=0}
        if (ie4) {var ieKey=event.keyCode; var nKey=0}
        if ((nKey==97 || ieKey==65) && !block.active) {   // if "A" key is pressed
              block.active = true
              slide()
        }
```

```
}
function keyUp(e) {
	if (ns4) {var nKey=e.which; var ieKey=0}
	if (ie4) {var ieKey=event.keyCode; var nKey=0}
	if (nKey==97 || ieKey==65) {
		block.active = false   // if "A" key is released
	}
}

function slide() {
	if (block.active) {
		block.xpos += 5
		block.left = block.xpos
		status = block.xpos	// not needed, just for show
		setTimeout("slide()",30)
	}
}
```

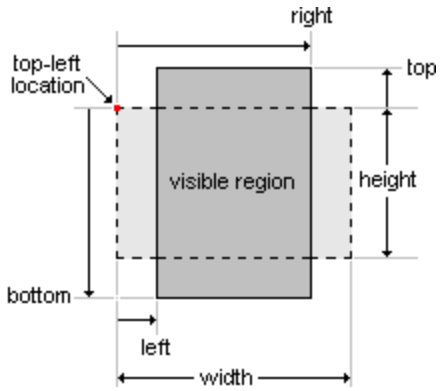
What Keys can I use?

As I mentioned earlier, the character sets for Netscape and Internet Explorer differ. In general, all letters, numbers, symbols, Space, and Enter will work fine. For a quick way to find out the nKey and ieKey values of particular keys you can view my nKey and ieKey Finder.

Clipping Layers

Clipping refers to what part of the layer will be visible. You have understand the difference between the clip values, and the width and height - they are not the same. Width and Height really just tell the browser how to wrap the HTML elements inside it. Whereas clipping makes a window to view the layer through - it has no effect on any of the other properties of the layer (left or top location, width, height, visibility, etc.).

The clipping region is defined as a square by setting the clip value for each of the 4 edges (top, right, bottom, left). For each edge you can clip a portion of the viewing space away, or to add extra viewing space. All the clip values are with respect to that layer - the values are taken from the top-left location of the layer.



The CSS syntax for clipping is:

clip:rect(top,right,bottom,left)

Where the top, right, bottom, and left values are in pixels. And don't forget the order that they go in - it will be confusing if you mess them up.

Here's a DIV tag using clipping to define the viewable area:

<DIV ID="blockDiv" STYLE="position:absolute; left:50; top:80; width:100; height:50; clip:rect(-10,110,60,-10); background-color:#FF0000; layer-background-color:#FF0000;"> </DIV>

In this case it creates an extra 10 pixel border around the edge of the layer because clip top is -10 and clip left is -10. The clip right is 110 which is 10 more than our width, and the clip bottom is 60 which is 10 more than the height.

I put a few extra CSS properties in there too. The background-color (for ie4) and layer-background-color (for Netscape) are used to do just that - color the entire layer in whatever color you wish. This enables us to see our layer as a square and will help to visualize what's going on when we clip it. Usually you don't have to have the height of the layer,

but when you're using clipping you should put it in because if you don't IE won't color the extra space below the last element in the layer.

You can also have a background image in your layer. The CSS for IE is background-image:URL(filename.gif) and for Netscape it's layer-background-image:URL(fildname.gif). But in order for Netscape to display it properly you must have one more CSS property repeat:no. Here's the full CSS for a layer with a background images:

```
<DIV ID="blockDiv" STYLE="position:absolute; left:50; top:80; width:100;
height:50; clip:rect(-10,110,60,-10);
background-image:URL(filename.gif);
layer-background-image:URL(filename.gif); repeat:no;}
```

JavaScript and Clipping

Once you've clipped the layer can then obtain or change those clip values using JavaScript just like we can the location.

Clipping in Netscape:

In Netscape, you can obtain or change any of the clip values individually:

```
document.divName.clip.top
document.divName.clip.right
document.divName.clip.bottom
document.divName.clip.left
```

To show an alert of the top clip value you'd write:

```
alert(document.divName.clip.top)
```

Then to change the top clip value to 50 pixels you'd write:

```
document.divName.clip.top = 50
```

Clipping in Internet Explorer:

In IE you have to obtain all the clip values at the same time. For example you could pop up an alert showing the clip value:

```
alert(divName.style.clip)
```

That will return a string which represents what the clip values were defined as. Here's an example of what would be returned:

```
"rect(0px 50px 100px 0px)"
```

When you want the change the clip values you cannot just clip one of the edges like you can in Netscape - you must reset all your clip values at the same time:

```
divName.style.clip = "rect(0px 100px 50px 0px)"
```

This makes it a little awkward to clip in ie4. I've come up with a generic function that you can use to check the clip value for both browsers:

The clipValues() Function

The clipValues() function can be used to obtain the clip values for each edge of a layer.

```
function clipValues(obj,which) {
        if (ns4) {
                if (which=="t") return obj.clip.top
                if (which=="r") return obj.clip.right
                if (which=="b") return obj.clip.bottom
                if (which=="l") return obj.clip.left
        }
        else if (ie4) {
                var clipv = obj.clip.split("rect(")[1].split(")")[0].split("px")
                if (which=="t") return Number(clipv[0])
                if (which=="r") return Number(clipv[1])
                if (which=="b") return Number(clipv[2])
                if (which=="l") return Number(clipv[3])
        }
  }
```

What you do is tell it what layer (defined as a pointer variable) and what edge you want to find the clip value for. For example, once we've defined a pointer variable for a "blockDiv" layer, we show an alert of the top clip value by writting:

```
alert(clipValues(block,"t"))
```

The edge that you want to check only has to be the first letter in quotes - "t" (top), "r" (right), "b" (bottom), "l" (left).

Changing the Clip Values

To change the clip values I've written 2 generic functions that can be used pretty easily.

The clipTo() Function:

clipTo() allows you to re-clip the layer to specific values.

```
function clipTo(obj,t,r,b,l) {
        if (ns4) {
                obj.clip.top = t
                obj.clip.right = r
                obj.clip.bottom = b
                obj.clip.left = l
        }
        else if (ie4) obj.clip = "rect("+t+"px "+r+"px "+b+"px "+l+"px)"
```

```
  }
```

To use it you must tell it what layer/object to use, and the clip value for each edge - top, right, bottom, left respectively.

```
clipTo(block,0,100,100,0)
```

The clipBy() Function:

clipBy() allows you to shift the clip value by a given amount of pixels - it adds or subtracts from the current clip value for the edges:

```
function clipBy(obj,t,r,b,l) {
        if (ns4) {
                obj.clip.top = clipValues(obj,'t') + t
                obj.clip.right = clipValues(obj,'r') + r
                obj.clip.bottom = clipValues(obj,'b') + b
                obj.clip.left = clipValues(obj,'l') + l
        }
        else if (ie4) obj.clip = "rect("+(this.clipValues(obj,'t')+t)+"px
"+(this.clipValues(obj,'r')+r)+"px "+Number(this.clipValues(obj,'b')+b)+"px
"+Number(this.clipValues(obj,'l')+l)+"px)"
  }
```

Similar to the clipTo() function you just set how much you want to clip each edge by:

```
clipBy(block,0,10,5,0)
```

This will add 10 pixels to the right and 5 pixels to the bottom.

Netscape will always show the layer's color no matter where it's been clipped. But in IE when you clip outside of the layer's boundaries (adding extra borders) you can't see the edges of the layer.

Animated Clipping (Wiping)

By putting clipBy() commands into looping functions you can create all those neat wiping effects that I'm sure you've seen before. I'll tell you right now it's probably easiest if you make your own wipe() functions. It is possible to make a generic wipe function but I've included that into the Dynamic Layer Object as an add-on method (read the Wipe Method lesson for more info). The truth is though it's probably easier and much less code if you just write your own little function to wipe your layers. You do it in the same manner as you do slides with. Create a looping function that re-clips the layer:

```
function wipe1() {
        clipBy(block,0,5,0,0)
        setTimeout("wipe1()",30)
  }
```

But we have to have a way of stopping the wipe so just do a check to see if the edge has reached the desired value:

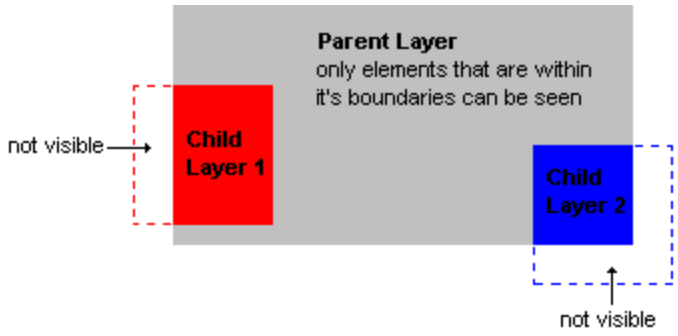
```
function wipe1() {
        if (clipValues(block,'r')<200) {
              clipBy(block,0,5,0,0)
              setTimeout("wipe1()",30)
        }
  }
```

Nesting Layers

I chose to leave the topic of nesting layers until now because they will effect your scripts in a number of places.

Nesting refers to when you want to group several layers together into one uniform element. In other words, nested layers are "layers within layers".

When you do this the child layers will be positioned with respect to their parent layer. Also, if the parent layer is clipped the child layers will appear as if in a window or like a plugin. If the child layers go outside the boundaries (the clip edges) of the parent layer they will become invisible - their visibility property will not change but they will appear as if they are offscreen:



I've found nesting comes in handy when you get into more complicated positioning. Since all the locations of the child layers are with respect to the parent layer, they are permanently "locked" into position. If you later want to move the location of the parent layer, you don't have to change the locations of the child layers because they will move accordingly. This is also true in sliding animations - all the child layers will slide in unison.

The JavaScript for nested layers differs quite dramatically between Netscape and Internet Explorer. I'll admit it is a bit of pain to make scripts with nested layers to work properly in both browsers - and that is the reason you don't see nesting used that often on the web. However I've developed some really great techniques to get around these problems so bear with me here as I assure you that nesting is so useful that once you start using it, you'll want to do just about everything that way.

Stylesheets and Nesting:

To nest layers all you do is wrap the parent DIV around all the child DIV's:

```
<DIV ID="parent1Div">

<DIV ID="child1Div"></DIV>

<DIV ID="child2Div"></DIV>
```

```
</DIV>
```

I have noticeably left out the styles for these DIV tags. This is because Netscape does not let you nest layers if the styles have been defined using the "inline" method like I have been using for this tutorial up until now. Netscape seems to only allow one set of nested layers, if you use any more then it will totally ignore all the styles for all the layers after it. So right off the bat we're going to ALWAYS define the styles using the STYLE tag. All the examples from this point forward will be done this way.

The CSS is basically the same except it's separated from the DIV tags:

```
<STYLE TYPE="text/css">
  <!--
  #parent1Div {position:absolute; left:100; top:80; width:230; height:120;
clip:rect(0,230,120,0); background-color:#C0C0C0; layer-background-
color:#C0C0C0;}
  #child1Div {position:absolute; left:-20; top:40; width:70; height:70;
clip:rect(0,70,70,0); background-color:#FF0000; layer-background-color:#FF0000;}
  #child2Div {position:absolute; left:180; top:70; width:70; height:70;
clip:rect(0,70,70,0); background-color:#0000FF; layer-background-color:#0000FF;}
  -->
  </STYLE>

  <DIV ID="parent1Div">

      <DIV ID="child1Div"></DIV>

      <DIV ID="child2Div"></DIV>

  </DIV>
```

I also included the clip regions to define the squares. In most cases where you use nesting you usually have to define the clip values and color the layers.

JavaScript and Nesting:

Nesting is where the JavaScript for Netscape and Internet Explorer go in completely opposite directions. In IE, there's no difference whether a layer is nested or not, you access the properties of the layer in the same manner as before:

```
childLayer.style.properyName
```

However in Netscape when you want to access the properties of a nested layer (a child layer) you have to reference it with respect to it's parent layer:

```
document.parentLayer.document.childLayer.propertyName
```

The extra "document" before the layer names are due to the fact Netscape

treates layers as separate documents - a child layer is part of the document of it's parent layer.

It is possible to nest layers an unlimited number of times as well - you just keep wrapping the DIV's over and over again. Say we changed that example set so that child2Div is inside child1Div

```
<DIV ID="parent1Div">

    <DIV ID="child1Div">

    <DIV ID="child2Div"></DIV>

    </DIV>

</DIV>
```

In that case to access the properties of child2Div you'd have to write:

```
document.parent1Div.document.child1Div.document.child2Div.propertyName
```

This concept will have to be incorporated into our pointer variables. Here's the way I'd define the pointer variables for that original example set:

```
function init() {
        if (ns4) {
              parent1 = document.parent1Div
              child1 = document.parent1Div.document.child1Div
              child2 = document.parent1Div.document.child2Div
        }
        if (ie4) {
              parent1 = parent1Div.style
              child1 = child1Div.style
              child2 = child2Div.style
        }
  }
```

Now onto some more problems...

CSS Properties Revisited:

Unfortunately Internet Explorer has a little technicality that poses quite a dilemma that had me baffled for quite a while. When you define the styles for your layers using the STYLE tag, IE does not let you read any the properties initially. So in IE if you were to check the current location of parent1 using:

```
alert(parent1.left)
```

You will find that you don't receive any value. This is true for all the CSS properties (left, top, width, height, visibility etc.).

I am still unclear why Microsoft made IE this way. It only occurs when

you use the STYLE tag, and only effects the initial values of properties. Once you start changing the properties in JavaScript you can then access them without problems.

How does this affect our situation? Well, if we want to assign other properties as we did earlier (xpos and ypos) we need a way to find the current location of the layer in some different way for IE4. It's fortunate that Microsoft included some extra non-standard CSS properties into IE4:

offsetX
offsetY
offsetWidth
offsetHeight

These extra properties are not effected by the IE4 STYLE tag problem so we can use those to obtain the current location of the layer. So here's our new code to add our xpos and ypos properties onto our pointer variables:

```
function init() {
        if (ns4) {
                parent1 = document.parent1Div
                parent1.xpos = parent1.left
                parent1.ypos = parent1.top
                child1 = document.parent1Div.document.child1Div
                child1.xpos = child1.left
                child1.ypos = child1.top
                child2 = document.parent1Div.document.child2Div
                child2.xpos = child2.left
                child2.ypos = child2.top
        }
        if (ie4) {
                parent1 = parent1Div.style
                parent1.xpos = parent1.offsetX
                parent1.ypos = parent1.offsetY
                child1 = child1Div.style
                child1.xpos = child1.offsetX
                child1.ypos = child1.offsetY
                child2 = child2Div.style
                child2.xpos = child2.offsetX
                child2.ypos = child2.offsetY
        }
  }
```

Once you've done that you can change the locations of the layers as before.

Visibility and Nesting:

Again, if you use the STYLE tag to define your layers you will not be able to obtain the original visibility value in IE4. But in my experience, obtaining the visibility is very rarely necessary. Usually you already know if a layer is visible or not. And remember that only

effects the initial visibility - after you change the visibility in JavaScript you will then be able to find the value.

Showing and hiding nested layers works pretty much the way you'd expect. Once you've defined the pointer variables you can use the same show/hide functions that I explained in the Showing and Hiding lesson.

But there is one thing that I should point out. If you don't define the visibility for the child layers, their visibility is "inherited" - it takes on the value of the parent layer's visibility. In that case when if you then hide or show the parent layer, all the child layers do the same. BUT... in Netscape if you either define the visibility for the child layers, or you start changing the visibility in JavaScript you lose the ability to hide or show all the child layers at once. In that case when you hide the parent layer, any child layer that is visible will still show through.

To avoid this situation, you have to set the visibility back to "inherit" instead of "visible" ("show" for Netscape). So instead of using the showObj() function, you have to manually set the visibility property:

```
mychild.visibility = "inherit"
```

Putting it back to inherit means if the parent is shown, the child layer will also be shown, and if the parent is hidden, it will also be hidden.

Changing Images

To create truly dynamic animations and demos you will eventually have to master the art of changing images on command.

For this lesson, I will dynamically change "imageA" into "imageB":


imageA.gif


imageB.gif

You must make sure that both images are exactly the same dimensions, otherwize when you change them, the new image will stretch itself to fit in the same area. In situations where you want to change images that are of different sizes you will not be able to use this type of code - you will have to resort to simply hiding and showing separate layers.

To start off, you have to initially show one of the images - so I decided to have a DIV tag named "imgDiv" with "imageA" inside it:

```
<DIV ID="imgDiv">
<IMG NAME="myImg" SRC="imageA.gif" WIDTH=75 HEIGHT=75
BORDER=0>
</DIV>
```

Notice that I assigned a NAME to the image (myImg), this name will be used when changing the image. The name must be totally unique, ie4. don't name the image the same as the DIV that it's inside otherwise it won't work. Usually what I do is append "Img" to the end of it as I append "Div" to the ID of a layer so that no naming conflicts occur.

Preloading Images

Before you can change the image, you have to preload the image into the browsers cache. This is the basic code to preload an image:

```
imagename = new Image();
imagename.src = "imagefilename.gif";
```

What this does is create an image object. Nothing to it really, just now we have an object by which we can access the image at any time. Whenever we need to switch an image it will already be available - you won't have to wait for the image to download because it will be cached. Since we'll be needing both imageA and imageB in the cache I have to have code to preload both of them:

```
imageA = new Image();
imageA.src = "imageA.gif";
imageB = new Image();
imageB.src = "imageB.gif";
```

The preload() Function

The more images you have to preload, the more you'll dislike having to rewrite the two lines each time. So instead of writing two lines, let's cut that down to one by using a generic preload() function:

```
function preload(imgObj,imgSrc) {
    if (document.images) {
        eval(imgObj+' = new Image()')
        eval(imgObj+'.src = "'+imgSrc+'"')
    }
}
```

where:

imgObj - the name of the object associated with the image
imgSrc - the source filename (url) of the image

Examples:

```
preload('imageA','imageA.gif')
preload('imageB','imageB.gif')
```

It's best to preload your images while the page is loading rather than waiting until after the page loads, so I'd recommend always calling the preload function immediately after defining it.

Changing the Image

Once you've preloaded the images you can the access and change any image on the page. Changing images that are inside layers works a little differently between Netscape and IE so first I'll show the explicit code for changing each, then I will show a generic function that you can use in any situation.

If the image is not in a layer, the general way to change an image is this:

```
document.images["imageName"].src = imageObject.src
```

Where imageName is the name you supplied in the IMG tag, and imageObject is the name of the preloaded image object.

So in my case I could use:

```
document.images["myImg"].src = imageB.src
```

But remember, that is if the image is not in a layer, as soon as it's in a layer things change.

In Netscape, you have to reference what DIV tag it is in. In my case it's in the imgDiv layer so you have to append document.imgDiv.document in front of the code:

```
if (ns4) document.imgDiv.document.images["myImg"].src = imageB.src
```

The extra "document" between the name of the DIV and the images is necessary because Netscape treats DIV's as a totally separate document.

But in Internet Explorer you don't have to do this, you just access it as if it weren't in a layer at all:

```
if (ie4) document.images["myImg"].src = imageB.src
```

And there you have it. All you gotta do now is put that code into a separate function and call that function when you want to change it:

```
function changeToA() {
        if (ns4) document.imgDiv.document.images["myImg"].src = imageA.src
        if (ie4) document.images["myImg"].src = imageA.src
    }

    function changeToB() {
        if (ns4) document.imgDiv.document.images["myImg"].src = imageB.src
        if (ie4) document.images["myImg"].src = imageB.src
    }
```

The changeImage() Function

The changeImage() function eliminates the need to have separate functions for each time you want to change an image. You just send it the layer it is in, the name of the image, and the name of the preloaded image object - layer, imgName and imgObj respectively:

```
function changeImage(layer,imgName,imgObj) {
        if (document.layers && layer!=null)
eval('document.'+layer+'.document.images["'+imgName+'"].src = '+imgObj+'.src');
        else document.images[imgName].src = eval(imgObj+".src");
    }
```

In my situation, I can replace the changeToA() function with simply:

```
changeImage('imgDiv','myImg','imageA')
```

And the same for imageB:

```
changeImage('imgDiv','myImg','imageB')
```

View images2.html for an example using the changeImage() function.

Notes:

The changeImage() function can also be used for nested layers, for the layer argument you can insert parentLayer.document.childLayer similarly to how the Dynamic Layer object handles nested layers.

You can use this function for images thar aren't even in layers, just

pass null for the layer argument:

changeImage(null,'myImg','imageB')

Also, the changeImage() function is backward compatible. If you have a layers page and view it in Netscape 3, the function will still work properly. You can try it out by viewing any of these examples with that browser. No other browser is capable of changing images, so if you wanted to error check for the ability to change images you can use this alteration of the changeImage() function:

```
function changeImage(layer,imgName,imgObj) {
        if (document.images) {
            if (document.layers && layer!=null)
eval('document.'+layer+'.document.images["'+imgName+'"].src = '+imgObj+'.src')
            else document.images[imgName].src = eval(imgObj+".src")
        }
    }
```

Both the changeImage() and preload() functions are part of the DynAPI and are in the images.js file:

Mouse Rollovers

I figured that the topic of mouse rollovers has kinda been beaten to death so I didn't bother covering it previously. However due to the number of people that asked me about it I'll quickly show how to do it using my changeImage() function.

The idea behind rollovers is dead simple, when you put the mouse over an image, it changes to a different image, and when you move your mouse out of the image, it changes back. To accomplish this you have to surround the IMG tag with an an anchor/hyperlink and call the changeImage() function using the onMouseOver and onMouseOut events. The onMouseOver and onMouseOut events have to be called from the hyperlink because in Netscape the IMG tag does not have those events built into it.

Remember though, you have to point the anchor to somewhere before you can use it. Most often rollovers are used in toolbars so you just stick in the page you want it to go to. But in situations where you don't want the hyperlink to go anywhere, you can instead insert javascript:void(null) for the HREF. That is is just a command that does absolutely nothing. The hyperlink will still exist - it just executes a javascript command that does nothing.

```
<DIV ID="imgDiv">
    <A HREF="javascript:void(null)"
    onMouseOver="changeImage('imgDiv','myImg','imageB')"
    onMouseOut="changeImage('imgDiv','myImg','imageA')">
    <IMG NAME="myImg" SRC="imageA.gif" WIDTH=75 HEIGHT=75 BORDER=0></A>
    </DIV>
```

Layer Writing

The contents of your layers (the text and HTML) can be re-written like a variable by using a trick. It's well known that Internet Explorer has the ability to change what's inside a DIV tag, but you can do a similar thing in Netscape - and that's to use document.write's to re-write the entire layer.

Internet Explorer 4.0:

In Explorer, you can access the HTML inside a DIV tag (or any other tag for that matter) by writing:

```
document.all.divID.innerHTML = "your new text"
```

Where divID is the ID of the DIV tag you want to change. You can also write it another way which I prefer more:

```
document.all["divID"].innerHTML = "your new text"
```

This way it evaluates "divID" as a string which will be more useful for making it cross-browser capable.

Netscape Navigator 4.0:

Since each layer is essentially it's own document, it has most of the capabilities that the main document does. Importantly for this lesson you can re-write what's in that document.

Now to normally re-write a document, you'd use the document.write() command for the text, and the document.close() command to signal that the writing process has finished:

```
document.open() document.write("my text here") document.close()
```

For a CSS layer, you can use the Netscape Layers() array before using document.write() and document.close():

```
document.layers["divID"].document.open()
document.layers["divID"].document.write("my text here")
document.layers["divID"].document.close()
```

Putting Them Together

The syntax differences between IE and Netscape lend themselves very nicely to being included in one generic function that can do both at the same time:

```
function layerWrite(id,nestref,text) {
        if (ns4) {
                var lyr = (nestref)?
eval('document.'+nestref+'.document.'+id+'.document') :
document.layers[id].document
```

```
                lyr.open()
                lyr.write(text)
                lyr.close()
        }
        else if (ie4) document.all[id].innerHTML = text
}
```

Using this function all you have to do is tell it what layer to change, and what text to change it too:

layerWrite("mylayer",null,"my text here")

Changing Styles

Internet Explorer's revolutionary DOM (Document Object Model) allows almost all the styles of a page element to be both readable and writeable at any time. This makes IE 4.0's Dynamic HTML system fundamentally superior. Most likely in the version 5 of the browsers this will be put on more even ground. However it is possible to mimic IE 4.0's functionality in Netscape 4.0 by using some clever tricks and document.write()-ing new layers with different styles. That is what most of this lesson is based around.

Changing the Background Color of a Layer

In Netscape to change the background color of a layer you have to set the bgcolor property of it's document object:

```
document.layer["layerName"].document.bgColor = "red"
```

In IE, you set the backgroundColor property of the layer:

```
document.all["layerName"].style.backgroundColor = "red"
```

So a cross-browser function for both would look like this:

```
function setBGColor(id,nestref,color) {
        if (ns4) {
                var lyr = (nestref)?
eval('document.'+nestref+'.document.'+id):document.layers[id]
                lyr.document.bgColor = color
        }
        else if (ie4) {
                document.all[id].style.backgroundColor = color
        }
}
```

Font Colors

To change the color of the text in a layer in IE is dead simple. Again, you'd just change the CSS color style of the element:

```
document.all[id].style.color = "#FF0000"
```

But to do the same in Netscape will require a document.write() to rewrite the layer with a different style. One way that works pretty good is to use the antient FONT tag:

```
<DIV ID="mytext"><FONT COLOR="#FF8000">My Text</FONT></DIV>
```

And then using the layerWrite() function, rewrite that layer with a new color:

```
layerWrite('mytext',null,'<FONT COLOR="#008000">My Text</FONT>')
```

However, I will pass that over and avoid the FONT tag altogether. The other general way to accomplish the task, is to predefine the styles you're going to use:

```
<STYLE TYPE="text/css">
 <!--
  .orange {color:#FF8000;}
  .green {color:#008000;}
  #mytext {position:absolute; left:50; top:100;}
  -->
 </STYLE>
```

And instead of using the FONT tag, I'll use the SPAN tag:

```
<DIV ID="mytext"><SPAN CLASS="orange">My Text</SPAN></DIV>
```

Then my javascript function rewrites the layer and has the CSS CLASS as an argument:

```
function mytextColor(color) {
        layerWrite('mytext',null,'<SPAN CLASS="'+color+'">My Text</SPAN>')
}
```

In my case I can execute the function with either of the following commands:

```
mytextColor('orange') mytextColor('green')
```

Text Rollovers

The primary use for changing font colors is to create text-based rollovers to replace the need for image-based rollovers. I've successfully created a system for doing this, however it's not a perfect solution, tedius and a little ugly. I'll show the way I first tried to do it, however I'll admit it doesn't work all that good. Still though I believe it's important to show you the thought process involved in fixing such problems.

Like the font color example, I just manually coded each of the styles I wanted to use. In my case I have 4 layers each containing the link which will change color when you roll over and off them:

```
<STYLE TYPE="text/css">
<!--
A.red {color:red;}
A.blue {color:blue;}
#link0Div {position:absolute; left:50; top:50;}
#link1Div {position:absolute; left:50; top:70;}
#link2Div {position:absolute; left:50; top:90;}
#link3Div {position:absolute; left:50; top:110;}
-->
</STYLE>
```

Since this is just a demo, I've used generic links and names for the layers:

```
<DIV ID="link0Div"><A CLASS="blue" HREF="page1.html"
onMouseOver="linkOn('link0Div','page1.html','Link 1')">Link 1</A></DIV>
<DIV ID="link1Div"><A CLASS="blue" HREF="page2.html"
onMouseOver="linkOn('link1Div','page2.html','Link 2')">Link 2</A></DIV>
<DIV ID="link2Div"><A CLASS="blue" HREF="page3.html"
onMouseOver="linkOn('link2Div','page3.html','Link 3')">Link 3</A></DIV>
<DIV ID="link3Div"><A CLASS="blue" HREF="page4.html"
onMouseOver="linkOn('link3Div','page4.html','Link 4')">Link 4</A></DIV>
```

Notice in the links I've only stated the onMouseOver events. This was a little trick I came up with. First it displays the onMouseOver, and when the link changes color, I rewrite the contents of the layer but replace the onMouseOver with an onMouseOut. That way there's never any interuption in the rollover sequence.

I have a separate JavaScript function for each state of the link:

```
function linkOver(id,link,text) {
        layerWrite(id,null,'<A CLASS="red" HREF="'+link+'"
onMouseOut="linkOut(\''+id+'\',\''+link+'\',\''+text+'\')">'+text+'</A>')
    }

    function linkOut(id,link,text) {
        layerWrite(id,null,'<A CLASS="blue" HREF="'+link+'"
onMouseOver="linkOver(\''+id+'\',\''+link+'\',\''+text+'\')">'+text+'</A>')
    }
```

The format of each of my links are the same, so I only needed to make the layerName (id), hyperlink location (link), and the displayed text (text), variables.

However, in Netscape that example doesn't seem to work quite properly. The onMouseOut's don't seem activate if you roll between links too quickly. You need a way of double-checking to see if some of links are still "on". So I decided to change how I set things up. I built a 2-dimensional array to keep track of the layer names, links, text, and a flag (true/fase) to indicate whether link is highlighted or not highlighted respectively.

```
    link = new Array()
    link[0] = new Array('link0Div','link1.html','Link 1',false)
    link[1] = new Array('link1Div','link2.html','Link 2',false)
    link[2] = new Array('link2Div','link3.html','Link 3',false)
    link[3] = new Array('link3Div','link4.html','Link 4',false)
```

Now we have a way of accessing any of the layer/links by numbers (0,1,2,3). To access any one of the link values I just write link[x][0] for the layer name, link[x][1] for the hyperlink etc. where x is the number of the link.

The linkOver() and linkOut() functions have to be updated to account for these changes. In those functions I added a line to set the flag value of the link (the link[x][3]) - true meaning "on" and false meaning "off". It was when onMouseOut occured that Netscape had the problems with. So to correct it, I do a little check the next time you do an onMouseOver (and hence call the linkOn() function) it goes through a loop to check the value of the flag of each link, if it's true it automatically calls the linkOut() function to force the link off.

```
function linkOver(num) {
        if (ns4) {
            for (var i=0;i<link.length;i++) {
                if (link[i][3]==true) linkOut(i)
            }
        }
        link[num][3] = true
        layerWrite(link[num][0],null,'<A CLASS="red" HREF="'+link[num][1]+'"
onMouseOut="linkOut('+num+')">'+link[num][2]+'</A>')
    }

    function linkOut(num) {
        link[num][3] = true
        layerWrite(link[num][0],null,'<A CLASS="blue" HREF="'+link[num][1]+'"
onMouseOver="linkOver('+num+')">'+link[num][2]+'</A>')
    }
```

The HTML for the links also have to be updated, the linkOver() and linkOut() functions only need to pass the index of the link now.

```
<DIV ID="link0Div"><A CLASS="blue" HREF="page1.html"
onMouseOver="linkOver(0)">Link 1</A></DIV>
<DIV ID="link1Div"><A CLASS="blue" HREF="page2.html"
onMouseOver="linkOver(1)">Link2</A></DIV>
<DIV ID="link2Div"><A CLASS="blue" HREF="page3.html"
onMouseOver="linkOver(2)">Link 3</A></DIV>
<DIV ID="link3Div"><A CLASS="blue" HREF="page4.html"
onMouseOver="linkOver(3)">Link4</A></DIV>
```

And there we have it, a working text rollover.

Of course, this technique can be used to change more than just the color, you can change other properties of the text by just defining your CSS values differently. The following makes the underline disappear, and when the link is on it makes the text in italics:

```
A.blue {color:blue; text-decoration:none;} A.red {color:red;
text-decoration:none; font-style:italic;}
```

Font Scaling

Font scaling (making text grow or shrink) is technically possible with Dynamic HTML, however the truth is it by no means is the best technology to accomplish it with. Something like Flash is much better and faster, and with the upcoming Flash 3.0 vector graphics/animation format it

looks like it'll definately be the choice of developers in the next version of the browsers. But until that time, you can certainly play around and accomplish something similar with just plain old JavaScript and CSS.

The concept of font scaling is exactly the same as the technique for changing colors. You first pre-define the CSS styles you're going to use:

```
.s10 {font-size:10pt;}
.s15 {font-size:15pt;}
.s20 {font-size:20pt;}
.s26 {font-size:26pt;}
```

And have a layer that first displays one of them:

```
<DIV ID="welcomeDiv"><SPAN CLASS="s10">Welcome</SPAN></DIV>
```

Then is just a matter of writing a function that re-writes that layer pointing to different style:

```
function fontScale(size) {
        layerWrite('welcomeDiv',null,'<SPAN
CLASS="s'+size+'">Welcome</SPAN>')
    }
```

For the following example I just have simple links that make the text bigger or smaller:

```
<A HREF="javascript:fontScale(10)">10 pt</A> <BR>
<A HREF="javascript:fontScale(15)">15 pt</A> <BR>
<A HREF="javascript:fontScale(20)">20 pt</A> <BR>
<A HREF="javascript:fontScale(26)">26 pt</A>
```

But of course that's not very fun. We can of course animate our font scaling so that the text appears to grow, or shrink if we want. To do that we'll need a lot of styles defined - one for each step in the sequence. It's easiest to do that in a loop to write out each style. I decided to have a font-scaling which goes from 10 to 50 points:

```
    var sizestr = '<STYLE TYPE="text/css">\n'
    for (var i=10;i<=50;i++) sizestr += '.s'+i+' {font-size:'+i+'pt;}\n'
    sizestr += '</STYLE>'
    document.writeln(sizestr)
```

I kept the function for cycling through the styles pretty simple. I just have a variable size which keeps track of which style is currently shown, and increment that by one each iteration. The if statement (if (size<50)) will determine when to end the loop.

```
    var size = 10
    function scaleWelcome() {
        if (size<50) {
            size++
```

```
                layerWrite('welcomeDiv',null,'Welcome')
                setTimeout('scaleWelcome()',30)
        }
}
```

Then just activate the function and it'll make the text grow incrementally larger.

That example has the text stuck to the left - as the text grows it doesn't re-adjust to the new size and ends up looking a little awkward. To solve that problem you can center the text by using either <DIV ALIGN="CENTER"> or the old <CENTER> tag. Although when you do this it's probably best to have a container layer with which you control the position with, then your text layer resides within it and centered. In the following example I've shown how to set that up.

External Source Files

The content shown in your layer can be called from an external file. However, the way you do it in Netscape is totally different than for IE so you will have to create totally browser specific code to do it.

There are 2 ways to accomplish this task:

Technique 1: Using LAYER and IFRAME

The CSS 1 standard does not have the ability to have it's contents to be initially called from an external file. But both browsers have a way that you can accomplish the same thing without using CSS at all.

Netscape's antiquated LAYER tag has an attribute (SRC) by which you can call an external file to be it's contents. Here's an example:

```
<LAYER NAME="textLayer" SRC="text.html" LEFT=50 TOP=50 WIDTH=300
HEIGHT=200 CLIP="0,0,300,200"></LAYER>
```

Layers work exactly the same way that CSS works, except the styles are made into attributes of the Layer tag.

However IE does not recognize the LAYER tag because it is a proprietory tag that Netscape introduced. When it reads the layer tag it will ignore it. But IE has it's own way of calling external files using the IFRAME tag. IFRAME, an inline frame, works just like normal frames except it's embedded into the page like a plug-in does. You can then position the IFRAME using CSS to put it anywhere on the page thereby mimicking what the LAYER tag does.

```
<STYLE TYPE="text/css"> #textDiv {position:absolute; left:50; top:50;
width:300; height;200; clip:rect(0,300,200,0);} </STYLE>

<DIV ID="textDiv"> <IFRAME SRC="text.html" NAME="textFrame"
SCROLLING="No" WIDTH="300" HEIGHT="200" MARGINWIDTH=0 MARGINHEIGHT=0
FRAMEBORDER="No"></IFRAME> </DIV>
```

By combining these 2 techniques you can have a page which loads the contents automatically.

Changing the Source File

In Netscape, to change the source for the Layer you change the .src property:

```
document.layerName.src = "newfile.html"
```

In IE, you change the src of the frame as if it were a normal frame:

```
parent.frameName.document.location = "newfile.html"
```

For the last example we can make one unified function that will do both

the Netcape and IE code at once:

```
function load(page) {
        if (ns4) document.textLayer.src = page
        else if (ie4) parent.textFrame.document.location = page
}
```

Using this function we can change the page by writing:

```
load("newpage.html")
```

Advantages:

- the contents for the layer/div will immediately be available
- the content pages can contain other JavaScript functions as long as you put in a few extra commands once in a while

For example, to call a function in the main document you have to use parent.functionName() instead of just functionName(). This is because remember in IE, the contents are in another frame even though it doesn't look like it.

Disadvantages:

- the IFRAME has the same disadvantages as using normal frames, they are totally opaque and you cannot lay other layers on top of them
- for the same reason the IFRAME-layer cannot be transparent, and cannot be nicely manipulated (slid, clipped, etc.) because it causes flickering when IE tries to redraw it

Hopefully the next version of the browsers they'll include a new CSS property like source:URL(filename.html) which will solve these problems. But for a neat hack to get around using IFRAME in the normal manner you can use technique 2...

Technique 2: Using IFRAME as a Buffer

With IE's ability to replace content by using it's innerHTML property, you can transfer the content from an IFRAME to a regular DIV thereby avoiding many of the display problems that IFRAME has. The only real disadvantage in using this technique is that the HTML you see will not be the real HTML - it will be a "virtual" layer. This makes your content pages less flexible. Doing JavaScript in the content pages will be very complex - I'm not even going to go into it. I'd only recommend using this technique if your content pages are mostly static.

First off we need an IFRAME that will be the "buffer". You could do it in this manner:

```
<SCRIPT LANGUAGE="JavaScript">
if (ie4) document.write('<IFRAME STYLE="display:none"
NAME="bufferFrame"></IFRAME>')
</SCRIPT>
```

Then you place one DIV which the content file will be loaded into:

```
<DIV ID="contents"></DIV>
```

Now for the Javascript. Here's a function to accomplish the task:

```
function loadSource(id,nestref,url) {
        if (ns4) {
                var lyr = (nestref)? eval('document.'+nestref+'.document.'+id) :
document.layers[id]
                lyr.load(url,lyr.clip.width)
        }
        else if (ie4) {
                parent.bufferFrame.document.location = url
        }
    }
    function loadSourceFinish(id) {
        if (ie4) document.all[id].innerHTML =
parent.bufferFrame.document.body.innerHTML
    }
```

The main part of the function will load the DIV directly with the external file in Netscape. But in IE will it will load the buffer frame named bufferFrame with the external file. The next obstacle is that you need a way to determine when the external file is completely loaded, so that you can then transfer the contenst of the frame to the DIV. There is hack that will work in IE (see Inside DHTML), but it won't work in Netscape. I forsee that it will be necessary for this to be done in Netscape so I will resort to using BODY onLoad in the external file. You merely call the loadSourceFinish() function and pass what DIV needs to be updated:

```
<BODY onLoad="parent.loadSourceFinish(id)">
```

This is done in the external file. This is the external file I used in the example below.

```
<HTML>
<HEAD>
<TITLE>Content Page</TITLE>
</HEAD>

<BODY onLoad="parent.loadSourceFinish('contents')">

This is my text. This is my text. This is my text. This is my text. This
is my text. This is my text. This is my text. This is my text.

</BODY>
</HTML>
```

The DynLayer Load Method

The DynLayer Object implements it's own load method which is an alternative to using the above function.

Working With Forms

For Netscape, forms and layers don't work so well together. Again since Netsape layers are essentially a whole different document, a form that crosses over several layers cannot be accomplished. For example, the following will work fine in IE, but in Netscape it won't work:

```
<FORM>

<DIV ID="layer1">
<INPUT TYPE="Text" NAME="field1" SIZE="10">
</DIV>

<DIV ID="layer2">
<INPUT TYPE="Text" NAME="field2" SIZE="10">
</DIV>

</FORM>
```

The solution is that you have put a form into each layer:

```
<DIV ID="layer1">
<FORM NAME="form1">
<INPUT TYPE="Text" NAME="field1" SIZE="10">
</FORM>
</DIV>

<DIV ID="layer2">
<FORM NAME="form2">
<INPUT TYPE="Text" NAME="field2" SIZE="10">
</FORM>
</DIV>
```

But this poses a problem when you want to capture the values of the fields in JavaScript or if you want to send the information to a CGI program. What you end up having to do is "glue" the data together using JavaScript and then doing whatever want with that information.

Remember in Netscape you have to reference the form with whatever layer it's within:

```
document.layerName.document.formName.fieldName.value
```

But in IE you just reference it as if it weren't in a layer:

```
document.formName.fieldName.value
```

Using this idea you can create some code that will extract the data from each field:

```
if (ns4) {
      field1value = document.layer1.document.form1.field1.value
      field2value = document.layer2.document.form2.field2.value
}
if (ie4) {
```

```
        field1value = document.form1.field1.value
        field2value = document.form2.field2.value
    }
```

But what if you want to then send that information to a CGI program? CGI's can only accept values from one form. So what you can do is create yet another form with hidden fields:

```
    <FORM NAME="mainForm" ACTION="/cgi-bin/inputform.pl"  METHOD="POST">
    <INPUT TYPE="Hidden" NAME="field1">
    <INPUT TYPE="Hidden" NAME="field2">
    </FORM>

    <DIV ID="layer1">
    <FORM NAME="form1">
    <INPUT TYPE="Text" NAME="field1" SIZE="10">
    </FORM>
    </DIV>

    <DIV ID="layer2">
    <FORM NAME="form2">
    <INPUT TYPE="Text" NAME="field2" SIZE="10">
    </FORM>
    </DIV>
```

The hidden fields in mainForm can be assigned the values from the other forms. Then you can send that form to a CGI to interperet the data:

```
    function sendForm() {
        if (ns4) {
            document.mainForm.field1.value =
document.layer1.document.form1.field1.value
            document.mainForm.field2.value =
document.layer2.document.form2.field2.value
        }
        if (ie4) {
            document.mainForm.field1.value = document.form1.field1.value
            document.mainForm.field2.value = document.form2.field2.value
        }
        document.mainForm.submit()
    }
```

I've made a simple demo that should show how this can be used in a real application. It's a simple 6-element form that could be used in a feedback form of some sort. I've also created a Perl script that simply returns back what was sent to it in a generated HTML page.

forms1.html - has each element split up into different forms and a function to glue the data together and send it to the Perl script. This was just to make sure my JavaScript and Perl scripts were working properly.

forms2.html - has each form in different layers and then allows you to flip between them by simply hiding and showing the appropriate layers. Then when you get to the last field you can submit the form.

forms-dhtml.txt source code for the perl script I used.

Page Templates

Using JavaScript/CSS page templates makes it easier to create an entire web site that has consistent features that are "site-wide" such as toolbars and default styles.

By linking each of your pages to external stylesheets (.css files) and using external JavaScript files (.js files) to write out your layers you can assemble your pages on the fly and have a central location for changing parts of your pages throughout your website. This is similar to what Server-Side Includes (SSI) do but when you use JavaScript files you have a lot more control over what gets written to the browser. For example you can determine what browser is being used and change the look of the page accordingly, or you can do other things like center all the layers (as in the Generating Layers lesson).

Also page templates, if used properly, can render the use of frames almost totally obsolete. When you use frames in unison with layers, you are limited in that your layers cannot cross over the frame borders. But by dynamically writing layers throughout your site you can do anything that you can in a single layers page.

Your CSS can be linked from one source file by using the LINK tag:

```
<LINK REL=STYLESHEET HREF="filename.css" TYPE="text/css">
```

That file can contain any stylesheet information that needs to be implemented across any number of html files (each of which must contain that LINK tag).

And similarly you can assign the source file for your JavaScript by using the SRC attribute:

```
<SCRIPT LANGUAGE="JavaScript" SRC="filename.js"></SCRIPT>
```

When you're developing page templates, I'd suggest you develop them as normal (all in one file) and then once you've got everything working, cut and paste the pieces into separate files. The following page is setup so that it's easy to extract the styles and the JavaScript which writes out the standard links for the page:

```
<HTML>
<HEAD>
<TITLE>The Dynamic Duo - Page Templates Demo 1</TITLE>
<STYLE TYPE="text/css">
<!--
#title {position:absolute; left:100; top:10; width:300; font-size:18pt; font-
weight:bold;}
#links {position:absolute; left:10; top:40; width:100; font-size:12pt;}
#content {position:absolute; left:100; top:55; width:400; font-size:10pt;}

BODY {font-family:"Arial";}
-->
</STYLE>
</HEAD>

<BODY>
```

```
<DIV ID="title">This is the Title</DIV>

<SCRIPT LANGUAGE="JavaScript">
document.writeln('<DIV ID="links">');
document.writeln('<B>Links:</B>');
document.writeln('<BR><A HREF="#">Page 1</A>');
document.writeln('<BR><A HREF="#">Page 2</A>');
document.writeln('<BR><A HREF="#">Page 3</A>');
document.writeln('</DIV>');
</SCRIPT>

<DIV ID="content">
<P>This is the body content....
</DIV>

</BODY>
</HTML>
```

Once that works, you can then make the CSS and JavaScript separate files that you link to:

```
<HTML>
<HEAD>
<TITLE>The Dynamic Duo - Page Templates Demo 2 [External Files]</TITLE>
<LINK REL=STYLESHEET HREF="mystyles.css" TYPE="text/css">
</HEAD>

<BODY>

<DIV ID="title">This is the Title</DIV>

<SCRIPT LANGUAGE="JavaScript" SRC="mylinks.js"></SCRIPT>

<DIV ID="content">
<P>This is the body content...
</DIV>

</BODY>
</HTML>
```

An easy concept to understand. However your pages will certainly be more complicated than this, so be careful!

Introduction to Object Oriented DHTML

Object oriented programming (OOP) takes a little to get used to, but I assure you if you've understood everything up until now, you (probably) won't have any difficulty understanding this lesson. Using Object Oriented DHTML is the next progression and will allow you to build more sophisticated scripts in an organized way.

You want to use objects whenever you need to create more than one of something. By creating one generic object, you can use it to create any number of them without duplicating much of your code. This has a big impact on your coding because it means you don't have to write as much code, everything is organized nicely, and the JavaScript will execute faster.

You can view Netscape's JavaScript tutorial for a comprehensive explanation of how to create new objects. On this lesson I'll give a brief overview of OOP and show how to make a simple reusable object for working with layers. Everything I do in this tutorial from this point on will be object-based so it's a good idea to follow along.

OOP Overview

When you create objects, really what you are doing is collecting a lot of different variables into a nice neat package which you can then easily access at a later time. Each object is it's own package, and the variables in that package are referred to as properties. You can then write functions which manipulate only the variables (properties) of that object, these are referred to as methods.

Using objects are helpful because they are designed so that they can be cloned to make many different objects that will be used similarly. Without confusing you yet, say we had a generic object named "block". Then we decided we want to have several blocks (block1, block2, block3 etc.) that all had similar variables (height, width, depth etc.). Using object oriented programming, the variables in the blocks are all collected together in this format:

Object Name	block1	block2	block3
property 1	block1.width	block2.width	block3.width
property 2	block1.height	block2.height	block3.height
property 3	block1.depth	block2.depth	block3.depth

This is the basic set-up for all JavaScript objects:

```
function objectName(arguments) { this.propertyName = somevalue }
```

So to create this generic "block" object, the code would look something like this:

```
function block(width,height,depth) {
        this.width = width
        this.height = height
```

```
        this.depth = depth
}
```

Notice that any variable defined with "this" in front of it becomes a property of the object. Many of the properties will initially be assigned values based on those passed in the arguments, but you can also set other default properties to whatever you'd like, for example:

```
function block(width,height,depth) {
        this.width = width
        this.height = height
        this.depth = depth
        this.color = "red"
}
```

...would make the width, height, and depth change depending on how it's initialized, but all the blocks would have a color property of "red".

Once this generic object function is created, you can clone objects based on this code, this is referred to as creating an instance of an object. The format for creating instances of a self-defined object is as follows:

```
newObjectName = new objectName(arguments)
```

So to create the block objects named block1, block2, and block3 you'd write:

```
block1 = new block(10,20,30)
block2 = new block(5,20,10)
block3 = new
block(15,30,15)
```

After they're defined you can do whatever you'd like with the object, check it's properties, or change them. Properties of an object work exactly the same way as normal variables do.

Once you've got the object function itself created, you can extend the functionality of the object by creating methods for it. The format for creating a method is as follows:

```
function objectName(arguments) {
        this.propertyName = somevalue
        this.methodName = methodFunction
}

function methodFunction(arguments) { // write some code for the object }
```

The methodFunction() works like any other function, except when you call the function you add the name of the object you want to apply the function to:

```
newObjectName.methodFunction()
```

So using this idea, we can add a method to the block object that calculates the volume of the block and returns the value:

```
function block(width,height,depth) {
        this.width = width
        this.height = height
        this.depth = depth
        this.color = "red"
        this.volume = blockVolume
    }
    function blockVolume() {
        return this.width*this.height*this.depth
    }
```

To find the volume of block1 you write:

```
block1.volume()
```

newobjects1.html has this simple block object and allows you to check the values of the properties and the volume using alert's.

Objects and Layers

Using the concepts from previous lessons, we can apply object oriented programming to make working with layers a much easier task. Instead of always re-writing initialization code for pointer variables, you can create generic objects that do the same thing with just one line of code.

Recall again what pointer variables are doing:

```
if (ns4) layer1 = document.layer1Div
else if (ie4) layer1 = layer1Div.style
```

...which is synonymous with:

```
if (ns4) layer1 = document.layers["layer1Div"]
else if (ie4) layer1 = document.all["layer1Div"].style
```

In this case, layer1 is a reference variable which points to the CSS properties of the layer named "layer1Div". If you've been using pointer variables on your own, you'll know that to create many pointer variables is rather cumbersome. To solve that problem I'm going to make a generic object that creates a css property which does the same thing as pointer variables do:

```
function layerObj(id) {
    if (ns4) this.css = document.layers[id]
    else if (ie4) this.css = document.all[id].style
}
```

To use this layerObj object, you can now initialize your pointer variables with only:

```
layer1 = new layerObj("layer1Div")
```

Once the layer1 object is initialized you can access the css properties of "layer1Div" by using:

layer1.css.left layer1.css.top layer1.css.visibility etc....

So all that's really changed is the extra css between everything. We can further extend the layerObj object by automatically defining x and y properties which will represent the left and top properties of the layer:

```
function layerObj(id) {
        if (ns4) {
            this.css = document.layers[id]
            this.x = this.css.left
            this.y = this.css.top
        }
        else if (ie4) {
            this.css = document.all[id].style
            this.x = this.css.pixelLeft
            this.y = this.css.pixelTop
        }
    }
```

So again if we were to create a layer1 object:

```
layer1 = new layerObj("layer1Div")
```

The layer1 object would have these properties:

```
layer1.css
layer1.x
layer1.y
```

Note that we can now use x and y as our properties because this is our own object we're working with. I was using xpos and ypos before because Netscape already included those properties as part of it's own Layer object. Since we're creating a new object we can name the properties whatever we like.

Making Methods

We can extend the functionality of the layerObj object to make it easy to manipulate layers just as we can create separate individual functions. Recall in the Moving Layers lesson I made a generic functions that can be used to move a layer to any position on the screen:

```
function moveBy(obj,x,y) {
        obj.xpos += x
        obj.left = obj.xpos
        obj.ypos += y
        obj.top = obj.ypos
    }
```

```
	function moveTo(obj,x,y) {
		obj.xpos = x
		obj.left = obj.xpos
		obj.ypos = y
		obj.top = obj.ypos
	}
```

Those functions translated into methods of the layerObj object look like this:

```
  function layerObj(id) {
		if (ns4) {
			this.css = document.layers[id]
			this.x = this.css.left
			this.y = this.css.top
		}
		else if (ie4) {
			this.css = document.all[id].style
			this.x = this.css.pixelLeft
			this.y = this.css.pixelTop
		}
		this.moveBy = layerObjMoveBy
		this.moveTo = layerObjMoveTo
	}
 function layerObjMoveBy(x,y) {
		this.x += x
		this.css.left = this.x
		this.y += y
		this.css.top = this.y
	}
 function layerObjMoveTo(x,y) {
		this.x = x
		this.css.left = this.x
		this.y = y
		this.css.top = this.y
	}
```

Again if we were to create a layer1 object:

```
layer1 = new layerObj("layer1Div")
```

We can then move the layer by using either the moveBy() or moveTo() methods:

```
layer1.moveBy(-5,10) layer1.moveTo(100,100) etc.
```

Where to go from here

This tiny object would only be used in very specific instances where all you need to do is move the layer around in a simple manner. The way I've made this object doesn't account for nested layers, so if you need to use nested layers you'd have to change the code to include the extra parent layers. The above object is actually derivative of The Dynamic

Layer Object. It is a unified cross-browser object oriented solution for DHTML.

BrowserCheck Object

With the advent of new browsers I needed a more sohpisticated browser checking solution - the document.layers and document.all check that I've been using will not suffice when we need to work with Netscape 4.0, 5.0 and Internet Explorer 4.0 and 5.0. We need a way to individually check for them. So I've built the BrowserCheck Object. This object is including within the DynLayer, and as well as a separate js file if you want to use it when you're not using the DynLayer.

This object will automatically define an instance of itself as "is":

```
is = new BrowserCheck()
```

The is object has the following properties

is.b - (String) browser name, converted to "ns" if Netscape, "ie" if Internet Explorer
is.v - (integer) version number (2,3,4,5 etc.)
is.ns - (boolean) Netscape 4 or greater
is.ns4 - (boolean) Netscape 4
is.ns5 - (boolean) Netscape 5
is.ie - (boolean) Internet Explorer 4 or greater
is.ie4 - (boolean) Internet Explorer 4
is.ie5 - (boolean) Internet Explorer 5
is.min - (boolean) Netscape 4 or 5, or Internet Explorer 4 or 5

This combination of properties will serve almost all your needs with respect to DHTML - that's all this object is designed to do. You could extend this to check for operating systems and mime-plugins or whatever.

So now you no longer need the following lines on any pages:

```
ns4 = (document.layers)? true:false
ie4 = (document.all)? true:false
```

Instead add the browser.js file to your page:

```
<script language="JavaScript" src="../dynapi/browsercheck.js"></script>
```

Or, if you are already using the DynLayer you don't need to do anything. The DynLayer has the BrowserCheck Object in its source.

The Dynamic Layer Object API

The Dynamic Layer Object API (DynLayer) is a lightweight object that provides a highly flexible manner of working with layers. Not only does it have common properties and methods for manipulating layers, it is an object based API which opens up a new way of working with layers that far exceeds the traditional way of coding DHTML. I've found it to be the ideal foundation for nearly every application of DHTML including animation, applications, and gaming environments.

All of the next lessons in this tutorial will use the DynLayer as the basis for accomplishing some other task, so it is important that you understand how it works and how to use it.

Features of the DynLayer Object:

- an object-based API that is easy to implement and use
- targets layers in a similar manner that I've used pointers to target layers - to avoid the problems of the different object models between Netscape and IE
- automatic nested layer handling
- full support for working with layers in separate frames
- provides it's own properties and methods for changing the location of the layer - to avoid the position problems associated with Microsoft's proprietory way ofchanging the location
- has the handy hide() and show() methods to change the visibility
- exposes a common event model for layer-based events
- has built-in slide, clip, and write methods
- includes css() function to auto-generate CSS syntax
- easy to make extensions such as wipe, glide, background color, external source files etc.

Quick Summary of what the DynLayer Does

Instead of using the true commands for manipulating layers, you make a DynLayer object and tell it the name of the layer that it will be used for. The DynLayer object will have 3 main properties:

- doc - points to "document.layername.document" in Netscape 4, and "document" in IE and Netscape 5 (Mozilla)
- elm - points to "document.layername" in Netscape 4, "document.all['layername']" in IE, and "document.getElementById('layername')" in Netscape 5
- css - points to elm in Netscape 4, elm.style in IE and Netscape 5

This manner gives us a similar way to access properties and methods of layers no matter which browser is being used.

The DynLayer then has other properties such as (x,y) to hold various information needed to operate the DynLayer, and has methods such as hide() show() and slideBy() and slideTo() to manipulate the layer. Read the next sections of the DynLayer for specific usage of these methods.

Initialization of DynLayers

The DynLayer can be applied to any layer using this general format:

objectName = new DynLayer(id,nestref,iframe)

Where:

objectName Name of the object - how you will reference the DynLayer object.

id ID of the layer to which this DynLayer is being applied to, this cannot be the same as the objectName

nestref (now optional in most circumstance). Nested reference to that layer

iframe Name of the iframe that the layer is contained. This is used when you need to manipulate a layer inside an IFrame. Currently IFrame is only a feature of is.ie.

Simple Layer Example:

Let's say you have a very simple layer with this structure:

<STYLE> #mylayerDiv {position:absolute; left:30; top:50;} </STYLE>

<DIV ID="mylayerDiv"></DIV>

To initialize 'mylayerDiv', your javascript will be:

mylayer = new DynLayer('mylayerDiv')

Notice how I append the 'Div' extension on the ID of the layer. I do this is because the name of the object cannot be the same as the ID of the layer. It's just a nice way to keep your variables separate.

Nested Layer Example:

I updated the DynLayer so that in most circumstances working with nested layers is exactly the same as working with non-nested layers.

If you have nested layers like this:

<DIV ID="myparentDiv"> <DIV ID="mylayerDiv"></DIV> </DIV>

If you wanted to use the nestref parameter you could initialize both of them like this::

myparent = new DynLayer('myparentDiv')
mylayer = new DynLayer('mylayerDiv','myparentDiv')

However, nestref is now optional, so you if you don't send the nestref parameter for the nested layer it will still work:

myparent = new DynLayer('myparentDiv') mylayer = new

```
DynLayer('mylayerDiv')
```

Notice the name of the parent layer is passed for the nestref argument.

Nestref for Multiple Nesting:

Again, no difference here, simply send the ID of the layer to the DynLayer and it will take care of the nested referencing itself. But for argument's sake, lets say you had these layers:

```
<DIV ID="myparent1Div">
<DIV ID="myparent2Div"> <DIV
ID="mylayerDiv"></DIV>
</DIV> </DIV>
```

In this case if you need to use the nestref parameter you must pass the names of all layers in that hierarchy separated by '.document.':

```
mylayer = new
DynLayer('mylayerDiv','myparent1Div.document.myparent2Div')
```

The pattern continues no matter how many times it's nested.

Working with Layers in separate Frames:

The DynLayer can be used to work with layers that are withing separate Frames, or IFrames (for IE only). In this case sending a nestref parameter for nested layers is manditory.

Note: This has changed, you must now send the frame reference as an object, no longer as a string

```
mylayer = new DynLayer('mylayer',null,parent.myframe) // send the frame
object reference
```

Assigning DynLayers Using Arrays:

If you have a sequencial set of layers you could alternatively assign DynLayers to Arrays rather than just variable names.

```
<DIV ID="mylayer1"></DIV>
<DIV ID="mylayer2"></DIV>
<DIV ID="mylayer3"></DIV> <DIV ID="mylayer4"></DIV>
```

To initialize these you could do:

```
mylayer = new Array()
for (var i=0;i<4;i++) {
        mylayer[i] = new DynLayer("mylayer" + i)
}
```

DynLayerInit() Function

Note: This function is now mandatory. Even if you do not specifically call this function, the first time you assign a DynLayer, the DynLayerInit() function will automatically be called to assign whatever layers have a "Div" in their ID, as well as find all the nestref's for all layers in the page. So you don't necessarily ever have to call this function manually unless you don't plan on assigning any DynLayers yourself.

The DynLayerInit() function is used to initialize all your DynLayers at once automatically and is used by the DynLayer to take care of all nested heirarchy work for Netscape. The way it works is by sniffing through the names of all layers in the page. Any layers that contain an ID with a "Div" extension on it will be automtically assigned to a DynLayer. This function does not apply to layers inside Frames or even more advanced circumstances like external files or dynamically generated layers.

As noted, you only have to call the DynLayerInit() function manually if you do not have any layers that don't have a "Div" extension, and therefore won't need to manually define any layers at all. Just call the function in your default init() function:

```
function init() { DynLayerInit() }
```

As long as you follow my lead of appending a "Div" to the names of your layers it will do the same thing as defining your layers manually. For example say you had a layer named "blueDiv" like this:

```
<STYLE TYPE="text/css">
#blueDiv {position:absolute; left:50; top:50;}
</STYLE>
<DIV ID="blueDiv"></DIV>
```

The DynLayerInit() function will automatically execute the code to initialize it:

```
blue = new DynLayer("blueDiv")
```

So any layers that have a "Div" extension never have to be initialized manually. This includes nested layers. Note though, layers that are contained in external files, different frames must be manually assigned.

Also note, the names of your layers may be something other than with the "Div" extension. However, DynLayerInit() will not automatically define these, although it will find it's nestref value so that you don't have to pass it it.

DynLayer Properties

Once you have applied the DynLayer to a layer, you have a unified way of referencing the layer for both Netscape and IE. From that point on you can use the DynLayer to retrieve and change the properties of the layer using a single command.

Core Properties

In most cases it is only necessary to have movement control of the layer. The core properties of the DynLayer provide the foundation for controlling the location of a layer.

css - points to the CSS properties of the layer
elm - points to the actual HTML element
id - retrieves the ID of the layer/element
x - stores the "left" location
y - stores the "top" location
w - captures the initial width
h - captures the initial height
doc - points to the document object for the layer
obj - a string property of the object which points to itself The css Property:

The css Property:

The css property is the way you directly reference the CSS properties of the layer. It is the equivalent to how I've used pointers in previous lessons. For Netscape it points to document.layer[id], and for IE it points to document.all[id].style

When you need to retrieve or change any of the CSS properties you reference the property in this manner:

objectName.css.propertyName

For example to retrieve the z-index of a "mylayer" DynLayer you'd use:

mylayer.css.zIndex

This can be done for any of the CSS properties. However in practice it is rarely necessary to call the css property manually because the DynLayer takes care of most of them: left, top, visibility, and clip value. The only property that you'd ever really need to access using the css property is zIndex.

The elm Property:

This points to the actual HTML element object. For Netscape it is equivalent to the css property. But for IE it points to document.all[id] rather than document.all[id].style. For basic DHTML animation and such this property isn't necessary, but there are situations where this is needed (check the Using Layer-based Events and Scroll Concepts lesson for an example).

The x and y Properties:

The x and y properties always contain the current location of the layer. They are equivalent to how I've been using the xpos and ypos properties in previous lessons. Using these separate properties is a personal preference of mine because I fell it is cleaner (as well as more efficient) to access the location of a layer as a property.

To retrieve the current location of the layer you use either:

objectName.x or objectName.y

As before, you have to ensure that the x and y properties are in synch with the actual left and top properties by copying the values:

objectName.x += 10 objectName.left = objectName.x

But you often never have to do that because I've included the moveTo() or moveBy() methods which change the location of the layer for you.

Instead of having the x and y properties you could optionally write your own methods like getLeft() or getTop() for extracting the current location. But you can do that on your own if you want to.

The w and h Properties:

Just as you can retrieve the location of the layer using x and y, you can retrieve the width and height of the layer using w and h:

objectName.w or objectName.h

The doc Property:

The doc property can be used when you want to access other elements inside the layer such as images and forms. In Netscape it points to document.layers[id].document, but in IE it points just to the main document. This is necessary because Netscape handles contents in layers as separate documents.

Element inside the layer should be called by:

mylayer.doc.elementName

// changing images mylayer.doc.images["myImage"].src = newimage.src

// get form value mylayer.doc.myform.myformfield.value

Read the Changing Images and Working With Forms lessons for more thorough explanations on those topics. There is also the image() method extenson.

The obj Property

The obj property is a string reference to the DynLayer. This property is necessary when methods use setTimeouts, setInterval, or eval's to call itself. Both of those statements only accept strings. For example you cannot have a setTimeout inside a method when it is set up like this:

```
setTimeout(this + ".methodName()",100)
```

Instead you have to use the obj property:

```
setTimeout(this.obj + ".methodName()",100)
```

The obj property is used by addon methods such as the slide and wipe methods, as well as other objects that use the DynLayer, and all my widget objects. It's extremely useful.

Using Layer-based Events

Pre-requisite: You should read the Document Mouse Events lesson before reading this one.

Using the elm property you can define events for your layer such as onMouseOver, onMouseOut, onMouseDown, onMouseUp, onMouseMove, and onClick.

In Netscape you can't mark up the event handlers like you can with an anchor:

```
<DIV ID="divName" onMouseDown="/*your code*/"></DIV>
```

However, you can define handlers directly using JavaScript. For Netscape you use:

```
document.layer[id].captureEvents(Event.MOUSEDOWN)
document.layer[id].onmouseover = downHandler
```

For IE:

```
document.all[id].onmouseover = downHandler
```

For a cross-browser solution you can define the handlers using the DynLayer elm property (which points to the actual element rather than the CSS properties):

```
objectName.elm.onmousedown = layerDownHandler
objectName.elm.onmouseup = layerUpHandler
objectName.elm.onmouseover = layerOverHandler
objectName.elm.onmouseout = layerOutHandler
objectName.elm.onmousemove = layerMoveHandler
objectName.elm.onclick = layerClickHandler
if (is.ns) objectName.elm.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP |
Event.MOUSEMOVE)
```

Make sure you define the mouse events using all lowercase. The handler names can be whatever you choose.

For Netscape you have to manually capture the events that you want to use. You can see in the captureEvents line how to set them up. If you don't need one of them just remove it from the command. You do not need to capture the mouseOver and mouseOut events as it appears they are captured by default.

Another way to define your handlers is to use the new Function() command. Doing it this way you can pass parameters quite easily.

```
objectName.elm.onmousedown = new Function("layerDownHandler('my
string!')")
if (is.ns) objectName.elm.captureEvents(Event.MOUSEDOWN)

function layerDownHandler(string) {
        status = string
}
```

There is yet another way to pass parameters, you can temporarily define sub-properties to the dynlayer event property, and retrieve them in the handler function:

```
objectName.elm.string = "my string!"
objectName.elm.onmousedown = layerDownHandler
if (is.ns) objectName.elm.captureEvents(Event.MOUSEDOWN)

function layerDownHandler() {
        status = this.string
}
```

Using Mouse Coordinates

In the handler functions you can retrieve the location of the mouse by using the following commands:

```
var x = (is.ns)? e.layerX:event.offsetX
var y = (is.ns)? e.layerY:event.offsetY
```

The x and y variables can then be used to do whatever you wish. For example, here's some code that will display the location of the mouse while over the layer:

```
function init() {
            mylayer = new DynLayer("mylayerDiv")
            mylayer.elm.onmousemove = layerMoveHandler
            if (is.ns) mylayer.elm.captureEvents(Event.MOUSEMOVE)
      }

function layerMoveHandler(e) {
            var x = (is.ns)? e.layerX:event.offsetX
            var y = (is.ns)? e.layerY:event.offsetY
            status = x+","+y
```

```
}
```

Core DynLayer Methods

The core methods (moveTo(), moveBy(), show(), and hide()) are by default included in the DynLayer. They are always available because they are used quite often.

The moveTo() Method:

The moveTo() method moves the layer to a specific coordinate:

objectName.moveTo(x,y)

If you need to only change one of x or y why you can send null for the value:

objectName.moveTo(x,null) or objectName.moveTo(null,y)

Examples:

mylayer.moveTo(100,50) mylayer.moveTo(100,null) mylayer.moveTo(null,50)

The moveBy() Method:

The moveBy() method will shift the location of a layer by a specified number of pixels:

objectName.moveBy(x,y)

Example:

mylayer.moveBy(5,0)

The show() and hide() Methods:

For changing the visibility I've included the methods show() and hide(). Their are no parameters to pass so their usage is simple:

objectName.show() objectName.hide()

In the standard DynLayer I haven't included any way to retrieve the visibility, only because I've found there's very few instances where you need to find the visibility (usually you already know if it's visible or not). But you could of course extend the DynLayer to keep track of that if need be.

Slide Methods

The slide methods provide an easy solution for creating simple straight-line animations. The slide methods are now automatically assigned to all DynLayers, so you can use them at any time.

The slideTo() Method:

The slideTo() method will slide the DynLayer to a specific coordinate.

objectName.slideTo(endx,endy,inc,speed,fn)

Where:
endx - the final x coordinate
endy - the final y coordinate
inc - the incrementation amount (the number pixel units to move each repetition)
speed - the speed of repetition in milliseconds
fn - (optional) the function or statement to be executed when the slide is complete

If you want the DynLayer to slide in a horizontal line pass null for the endy value. And if you want the DynLayer to slide in a vertical line pass null for the endx value.

Examples:

To slide the DynLayer to coordinate (100,50), in increments of 10 pixel units, at 20 milliseconds per repetition:

mylayer.slideTo(100,50,10,20)

To slide the DynLayer horizontally to the x-coordinate 80:

mylayer.slideTo(80,null,5,30)

To pop up alert to notify when a slide is complete:

mylayer.slideTo(100,50,10,20,'alert("The slide is complete")')

When using the fn property from a hyperlink you must do a trick with the quotes:

<A HREF="javascript:mylayer.slideTo(100,50,10,20,'alert(\'The slide is complete\')')"></A>

The slideBy() Method:

The slideBy() method will slide the DynLayer to another coordinate by defining the amount of pixels to shift from it's current location (similar to moveBy() but animated). The usage is very similar to slideTo():

objectName.slideBy(distx,disty,inc,speed,fn)

Where:
distx - the amount of pixels to shift horizontally
disty - the amount of pixels to shift vertically
inc - the incrementation amount (the number pixel units to move each repetition)
speed - the speed of repetition in milliseconds
fn - (optional) the function or statement to be executed when the slide is complete

If you want the DynLayer to slide in a horizontal line pass 0 for the endy value. And if you want the DynLayer to slide in a vertical line pass 0 for the endx value.

Examples:

To slide the DynLayer on a diagonal 40 pixels left and 60 pixels down:

```
mylayer.slideBy(-40,60,5,20)
```

To slide the DynLayer 50 pixels to the right:

```
mylayer.slideBy(50,0,5,20)
```

Making Sequences:

I left the fn property so that you always have a way of determining when the slide is complete. By taking advantage of this feature you can link a series of slide()'s together to make a sequence of animations. Here's an easy way to accomplish a sequence:

```
seq1 = 0
function sequence1() {
            seq1++
            if (seq1==1) {
                  // commands for first part of sequence
                  // link the slide back to this function to keep it going
                  mylayer.slideBy(50,0,10,20,'sequence1()')
            }
            else if (seq1==2) {
                  // commands for seconds part of sequence
            }
            else seq1 = 0 // reset to 0 so you can play the sequence again
      }
```

onSlide Handlers

I've added 2 event handlers to the Slide Methods:

onSlide - called in each and every step in the slide
onSlideEnd - called when the slide has finished (just like the "fn")

I have not put these handlers to large use, but it seems to work pretty well, and are perhaps better to use than the "fn" parameter in the slideBy() and slideTo() methods.

By default these handlers do nothing, but all you have to do is reset

them to some function after calling the slideInit() method:

```
mylayer.slideInit()
mylayer.onSlide = mylayerOnSlide // some function that runs each step in the slide
mylayer.onSlideEnd = mylayerOnSlideEnd // some function that runs when completed the slide
```

Clip Methods

The clip methods give you a simple way to clip your DynLayers in the same manner that I used the clip functions in the Clipping Layers lesson.

Note: although the clip methods are now automatically assigned to all DynLayers, you may still have to call the clipInit() method to initially clip the layer so that IE will have base clip values to work from. I will be looking into this further to hopefully remove this necessity.

Initialize The Clip Methods:

There are 2 different situations when applying the clip methods.

Situation 1: Clipped to the default values

This occurs when you have either a) have defined no clip values in your CSS, or b) the values you have defined in your CSS are equal to that of the width and height of the layer. In either case, to use the clip methods you must define the width and the height of the layer in your CSS. To initialize the clip methods in situation 1, you can use:

```
objectName.clipInit()
```

Situation 2: Clipped to arbitrary values

If you have clipped your layers other than that of the default values you must initialize the clip methods in a different manner. For example if your layer has a width of 100 and a height of 50, but you have clipped your layer to (-10,-10,60,110), then you must pass those values to the clipInit() method:

```
objectName.clipInit(clipTop,clipRight,clipBottom,clipLeft)
```

Example:

```
mylayer.clipInit(-10,-10,60,110)
```

This is necessary because IE cannot initaially determine the clip values, the clipInit() function will re-clip the layer to those values so that they can be determined thereafter.

Once you have initialized the clip methods, the DynLayer adds 3 additional methods by which you can retrieve or change the clip values.

The clipValues() Method:

The clipValues() method is used to retrieve the current clip value for any of the 4 edges.

objectName.clipValues(which)

For the which argument, you pass the letter in quotes signaling which of the edges you want to find the value of. So there are four different ways of calling the method:

objectName.clipValues('l') // clip left value
objectName.clipValues('r') // clip right value
objectName.clipValues('t') // clip top value
objectName.clipValues('b') // clip bottom value

The clipTo() Method:

The clipTo() method will clip the edges of the layer to the specified values:

objectName.clipTo(t,r,b,l)

Where: t - new clip top value r - new clip right value b - new clip bottom value l - new clip left value

For any of the values which you do not want to change pass null for the value.

Examples:

mylayer.clipTo(0,25,50,0) mylayer.clipTo(null,50,null,null)

The clipBy() Method:

The clipBy() method clips the edges based on their current value (as moveBy() is to moveTo()). The usage is the same as clipTo():

objectName.clipBy(t,r,b,l)

IE 5.0 Notes:

Unfortunately in IE 5.0, there is no way for the DynLayer to retrieve the initial clip values. So therefore if you want to do any complex clipping (like in the exmple below) you must manually send the clip values when calling clipInit().

Warning: IE works differently with respect to clipping. When you clip outside of it's original boundaries (0,width,height,0) the background will not show. The layer will still clip however (inwards), better to view with both browser to see what I'm talking about. If you need to extend the width/height of a layer when it has been clipped you must also change the css.width and css.height values of the layer as well

(only for IE because Netscape cannot change the true dimensions on the fly).

Write Method

I only recently decided to put the Write Method into my "standard" dynlayer (although that's not a good name for it). Generally with the DynLayer you only need to include the stuff that you want, and I wanted this method in there because about 3 or 4 other lessons use this.

The DynLayer Write method allows you to re-write the contents of the layers by using the document.write() command for Netscape, and the innerHTML property for IE. Please read the Layer Writing lesson for a full explanation of what's really going on when writing a layer.

Using the write method is very simple:

```
mylayer = new DynLayer(id,nestref,iframe)
mylayer.write(html)
```

Example:

```
mylayer = new DynLayer('mylayerDiv')
mylayer.write('my new content goes here')
```

DynLayer Functions

There are 2 other functions I need to explain:

The DynLayerTest() Function

The DynLayerTest() function was added only as a debugging function. I really should have included this sooner as it really helps you when you get into creating DHTML Objects. This function will automatically double check your initialization (the id and the nestref parameters) to make sure the layer in fact does exist before defining itself. It does not check when you use IFrame, you're on your own with that.

If you happen to get the initialization wrong, the DynLayerTest() function will show an alert() of what layer in the hierarchy is the problem (<-- this layer cannot be found). This should help you figure out what's wrong with your code. This function is redundant for layers that are initialized by the DynLayerInit() function.

The css() Function

I'm including the css() function in the DynLayer because many of the later lessons in this tutorial are using it. This is more a personal preference than a necessity. Read the Generating Layers lesson for an explanation of this function.

How To Extend The DynLayer

There are 4 different ways to extend the DynLayer

DynLayer Add-on Methods
DynLayer Add-on Objects
Objects Which Encapsulate The DynLayer
Objects Which Internally Use The DynLayer

DynLayer Add-on Methods

It is quite easy to add you own methods to the DynLayer. Just create your own function:

```
function DynLayerMyNewMethod() {
        // code for this method
}
```

This method is not available to the DynLayer until you "attach" it. There are 3 ways to do this

1. Use the prototype command (recommended) This way your method will be available to all DynLayers that you define

```
DynLayer.prototype.myNewMethod = DynLayerMyNewMethod
```

You can either make your own .js file and include both the function and the prototype call in that function, or include these in the dynlayer.js source file itself.

2. Include your method in the constructor (not recommended): This will do the same as a prototype but all methods of the DynLayer now use prototyping

```
function DynLayer(id,nestref,frame) {
        ... code in constructor
        this.myNewMethod = DynLayerMyNewMethod
}
```

3. Assign the method explicitly to a specific DynLayer

This way your method will only be available to a specific DynLayer. In some instances this may be preferrable.

```
mylayer.myNewMethod = DynLayerNewMethod
```

DynLayer Add-on Objects

If you require an addition to the DynLayer which contains it's own set of properties and several methods, you may want to make it it's own object and append it to the DynLayer. What I suggest you do is pass the new object information so that it is still able to update the DynLayer. Do do this the object will require the name of the DynLayer, as well as

the name of the add-object:

```
objectName = new DynLayer("objectNameDiv")
      objectName.myobject = new MyObject("objectName","myobject")

      function MyObject(dynlayer,name) {
            this.dynlayer = dynlayer
            this.name = name
            this.value = eval(this.dynlayer+'.x') + 100  // use eval's to capture data
                from the Dynlayer
            this.method = MyObjectMethod
            this.repeat = MyObjectRepeatMethod  // repeats MyObjectMethod using
                setTimeout
      }
      function MyObjectMethod() {
            eval(this.dynlayer+'.moveBy(10,10)')  // use eval's to assemble the name
                of the DynLayer
      }
      function MyObjectRepeat() {
            setTimeout(this.dynlayer+'.'+this.name+'.method()',50)  // use eval's to
                assemble the name of the object's method
      }
```

Then to use the add-on object you use this general format:

```
objectName.myobject.method() or
objectName.myobject.repeat() etc.
```

This tactic is used by the Geometric Objects, and the Path Object.

Objects Which Internally Use The DynLayer

If you want one object to control multiple layers, your best bet is to assign properties which are in fact DynLayers.

Option 1: Send the object the names of the layers, and let the object define the DynLayers

```
myobject = new MyObject('layer1Div','layer2Div')

function MyObject(lyr1,lyr2) {
       this.lyr1 = new DynLayer(lyr1)
       this.lyr2 = new DynLayer(lyr2)
}
```

This way, the main object (MyObject) can control both those layers by using the properties and methods of those DynLayers. For example you could create a method by which it slides both layers in unison:

```
myobject = new MyObject('layer1Div,'layer2Div')

function MyObject(lyr1,lyr2) {
       this.lyr1 = new DynLayer(lyr1)
       this.lyr1.slideInit()
```

```
        this.lyr2 = new DynLayer(lyr2)
        this.lyr2.slideInit()
        this.slideBoth = MyObjectSlideBoth
}
function MyObjectSlideBoth() {
        this.lyr1.slideBy(-100,0,5,50)
        this.lyr2.slideBy(100,0,5,50)
}
```

This tactic is used by all of the widgets/components, however usually what I do is generate layer names automatically, but it's still the same basic idea.

Option 2: Pre-define your DynLayers and send the object the names of the DynLayers

```
mylayer = new DynLayer("mylayerDiv")
myobject = new MyObject(mylayer)

function MyObject(dynlayer) {
        this.dynlayer = dynlayer // do something with this.dynlayer
}
```

This tactic is used by the Drag Object.

Objects Which Encapsulate The DynLayer

Note: As of the June 23 update to the DynLayer you must also set the prototype of your object to the Dynlayer's prototype in order to attach the methods.

Perhaps the most powerful way of extending the DynLayer is to , is to make an object encapsulate the DynLayer, in other words to import all the functionality of the DynLayer into that object.

Be aware, this is not the same thing as the above section. The above section makes the DynLayer a property of an object. Encapsulation means that this object actually becomes a DynLayer that has it's own set of properties and methods.

To encapsulate the DynLayer, you assign the DynLayer as a method of the object, and immediately call that method, and at the end make your object have the same prototype as the DynLayer. What that does is attach all the methods of the DynLayer to your Object.

```
myobject = new MyObject('myObjectDiv',null)

function MyObject(id,nestref) {
        this.dynlayer = DynLayer
        this.dynlayer(id,nestref)
}

MyObject.prototype = DynLayer.prototype
```

What this does is assigns all the properties and methods of the DynLayer to this object. It is in fact a DynLayer itself because you work with it in the same manner...

myobject.hide() myobject.moveBy(10,10) etc.

So what advantage does this have? Well this is the ultimate way of extending the DynLayer because you can add much more functionality to this object. This technique is the ideal way to make a back-end to a DHTML game, where you need many types of objects that do different tasks, yet they all need to control layers like the Dynlayer does.

DynLayer Extensions [Common]

These are commonly used additions that you may want to use. Note: these examples are shown for a DynLayer that is named 'mylayer' but they will work for all DynLayers of any name.

When you want to use these functions all you have to do is include the dynlayer-common.js file after the dynlayer file:

```
<SCRIPT LANGUAGE="JavaScript" SRC="dynlayer.js"></SCRIPT>
<SCRIPT LANGUAGE="JavaScript" SRC="dynlayer-common.js"></SCRIPT>
```

External File Load - load()

This method is based on the External Source Files lesson.

Usage of the load() method:

```
mylayer.load('myfile.html',fn)
```

The fn parameter is optional, it's used to execute some other function or statment when the external file has been fully loaded into the page.

In the main HTML document you must have an hidden IFrame named "bufferFrame", this is used to copy the contents of the external file into the layer.

```
<IFRAME STYLE="display:none" NAME="bufferFrame"></IFRAME>
```

The external html file must call the DynLayer's loadFinish() method. Since in IE, the main document is in a different frame, we must call it as "parent". Fortunately this is simultaneously compatible for Netscape because in Netscape it is all the same document, and therefore in that case "parent" is synonymous with "document".

```
<BODY onLoad="parent.mylayer.loadFinish()">
```

Warnings: This method will not work "as-is" if these these files are all to be contained within another frameset. In that case you'd need to send an additional parameter for the name of the frame instead of "parent".

Nor will this work if you want to load multiple files simultaneously into separate layers. This function assumes there's only one IFrame, and hence only one file in a buffer-zone. If you wanted multiple files to be buffered like this you'd have to have separate IFrames, and yet another parameter to determine which frame to take the contents from.

Background Color - setbg()

Simply sets the background color of the layer. Watch out though, you usually have to have the layer clipped, and you'll sometimes run into problems with text that's contained within the layer. I'll leave you to encounter all the "fun" with this function :)

Usage of the setbg() Method:

```
mylayer.setbg('#ff0000')
```

Change Image - img()

This one-line method can be used instead of the changeImage() function so that you don't have to worry about nested references:

Usage of the img() Method:

```
myImgObject = new Image() myImgObject.src = 'myimg-new.gif'

mylayer.img('myImg','myImgObject')

// image must have a NAME assigned, index values won't work between both browsers
<div id="mylayerDiv"><img src="myimg.gif" name="myImg"></div>
```

Get Relative X Location - getRelativeX()

This function can be used to find the actual left location of the layer (relative to the document). This only has to be used when you've positioned your layer relatively

In order to set your CSS to make a relatively positioned layer you can use either of the following:

```
#mylayer {position:absolute;} // no left or top defined

css('mylayerDiv',null,null) // in css() function null,null for x,y means relative positioning
```

Then to find the actual left location use the getRelativeX() method:

```
var x =  mylayer.getRelativeX()
```

Get Relative Y Location - getRelativeY()

Same as the above but for the top location:

var y = mylayer.getRelativeY()

Get Content Height - getContentHeight()

When you don't specify a height in your CSS, you can still obtain what the actual height of the contents of that layer is by using the getContentHeight() method. Note this uses the same tactic as shown in the Scrolling Concepts lesson, however that lesson does the true call for this value explicitly.

var h = mylayer.getContentHeight()

Get Content Width - getContentWidth()

Same as above but for the width:

var w = mylayer.getContentWidth()

Wipe Methods

The wipe methods are animated versions of the clip methods (as slide is to the move methods).

When you want to use these functions all you have to do is include the dynlayer-wipe.js file after the dynlayer file:

<SCRIPT LANGUAGE="JavaScript" SRC="dynlayer.js"></SCRIPT> <SCRIPT LANGUAGE="JavaScript" SRC="dynlayer-wipe.js"></SCRIPT>

A change in IE 5.0 causes clipping to be a bit of pain. What I recommend is before you do any wipes, reclip your layer using the clipTo() function - just use your initial CSS clip values.

objectName.clipTo(t,r,b,l) // use your CSS values // then do your wipe

The wipeTo() Method:

The wipeTo() method will wipe (clip incrementally) the DynLayer's edges from their current value to specific new value. It can do this for any single edge, or multiple edges at the same time.

objectName.wipeBy(endt,endr,endb,endl,num,speed,fn)

Where:
endt - final clip top value
endr - final clip right value
endb - final clip bottom value
endl - final clip left value num - the total number of steps in the wipe sequence
speed - speed of repetition in milliseconds
fn - (optional) function or statement to execute when the wipe is complete

For any of the edges which you do not wish to be clipped, pass null for

it's value.

Examples:

To wipe the DynLayer's top edge to 0, right to 100, bottom to 100, and left to 0 (making a square box 100x100), in 10 steps, at 30 milliseconds per step:

mylayer.wipeTo(0,100,100,0,10,30)

To wipe only the right edge to 100:

mylayer.wipeTo(null,100,null,null,10,30)

The wipeBy() Method:

Again the wipeBy() is the same as the wipeTo() except the edges are shifted a given number of pixels:

objectName.wipeBy(distt,distr,distb,distl,num,speed,fn)

Where:
distt - clip top increment
distr - clip right increment
distb - clip bottom increment
distl - clip left increment
num - the total number of steps in the wipe sequence
speed - speed of repetition in milliseconds
fn - (optional) function or statement to execute when the wipe is complete

For any of the edges that you do not wish to be clipped pass 0 for it's value.

Examples:

Wipe all edges "outward" by 20 pixels:

mylayer.clipBy(-20,20,20,-20,5,30)

Wipe all edges "inward" by 20 pixels:

mylayer.clipBy(20,-20,-20,20,5,30)

Wipe the right edge outward by 100 pixels:

mylayer.clipBy(0,100,0,0,5,30)

When working with the wipe methods you have to keep your orientation correct. Remember how positive and negative values will effect each of the edges:

Edge	Positive Increment	Negative Increment
left	subtracts from the edge	adds more to the edge

right	adds more to the edge	subtracts from the edge
top	subtracts from the edge	adds more to the edge
bottom	adds more to the edge	subtracts from the edge

Glide Methods

The Glide methods are almost the same as the Slide Methods except they use a different formula for moving the layer. The Slide methods are simple straight-line animations, whereas the Glide methods use trigonometric math to create a subtle acceleration or deceleration effect. The result is some very slick looking animations.

As with the Wipe methods, I've made the Glide library a separate javascript file, dynlayer-glide.js. You must call include this file in any code that uses the glide methods:

```
<SCRIPT LANGUAGE="JavaScript" SRC="dynlayer.js"></SCRIPT>
<SCRIPT LANGUAGE="JavaScript" SRC="dynlayer-glide.js"></SCRIPT>
```

The glideTo() Method:

Glides the layer to a specific co-ordinate. The parameters are almost the same as in the slideTo() method:

```
objectName.glideTo(startSpeed,endSpeed,endx,endy,angleinc,speed,fn)
```

Where
startSpeed - "slow" to begin slowly (acceleration), "fast" to begin fast (deceleration)
endSpeed - "slow" to end slowly (decelaration), "fast" to end fast (acceleration)
endx - final x-coordinate
endy - final y-coordinate
angleinc - the angle incrementation (read below)
speed - speed of repetition in milliseconds
fn - (optional) function or statement to execute when complete

The angleinc parameter is probably the only one which isn't obvious. The glide methods use a Sine wave as the basis for the acceleration, and the angleinc simply determines how many degrees to jump each time. The bigger the angleinc, the bigger the jumps it will make. So it is similar to the inc value in the Slide methods - usually a value from 5 to 10 is good to use.

Example: glides to (50,50), starting slow, ending slow, at 10 degrees, and 20 milliseconds per interval.

```
mylayer.glideTo("slow","slow",50,50,10,20)
```

The glideBy() Method:

Same as all the others, glideBy() shifts the location by a given number of coordinates:

objectName.glideBy(startSpeed,endSpeed,distx,disty,angleinc,speed,fn)

Where distx and disty, are now the amount it will shift by.

Geometric Objects

Note: I'm told there's a few bugs in these, so use at your own risk :)

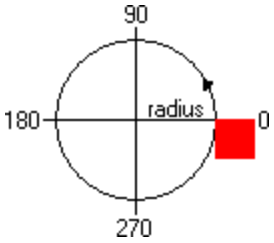
The Geometric Objects provide a solution for doing animation along a path of a geometric shape. They could be used in demos and games if need be. They are built as add-on objects to the DynLayer. For example, any DynLayer that you have can also have a geometric object added on to it.

Circle Object

The Circle Object will slide a layer in a perfect circle.

Initialization:

The Circle Object is an addon object to the DynLayer. You must make a new property onto the DynLayer and make that the Circle Object:



objectName.circle = new Circle("objectName","circle")

Example:

```
mylayer = new DynLayer("mylayerDiv")
mylayer.circle = new Circle("mylayer","circle")
```

You must pass the name of the DynLayer and the name of the new circle object (which would usually be "circle") to the Circle Object. These 2 pieces of information are needed in order for the circle object to access the DynLayer.

The play() Method:

The play() method begins the slide along a circular path. You must pass information to the play() method that define the shape and properties of the circle:

objectName.circle.play(radius,angleinc,angle,endangle,speed,fn)

Where:

radius - radius of circle
angleinc - angle incrementation
angle - starting angle
endangle - ending angle
speed - speed of repetition
fn - (optional) function or statement to execute when complete

Notes: to rotate counter-clockwise use a negative angleinc value, for clockwise use a positive value If you want the circle to keep looping then pass null for the endangle value.

Examples:

A circle, radius 100, increment 5 degrees (clockwise), starting at 0 degrees, ending at 90 degrees, at 20 milliseconds per repetion:

mylayer.circle.play(100,5,0,90,20)

The same except it loops:

mylayer.circle.play(100,5,0,null,20)

The pause() and stop() Methods:

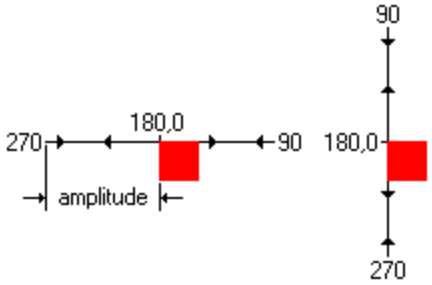
The pause() method will pause the current circular path. The next time a play() method is called it will continue along the same circular path:

objectName.circle.pause()

The stop() method will stop the current circular motion. The next time a play() method is called it will slide along a new circular path.

objectName.circle.stop()

Hover Object



The Hover Object will hover a layer in a straight-line.

Initialization:

objectName.hover = new Hover("objectName","hover")

Example:

```
mylayer = new DynLayer("mylayerDiv")
mylayer.hover = new Hover("mylayer","hover")
```

The play() Method:

The play() method begins the hover motion:

```
objectName.hover.play(amplitude,angleinc,angle,cycles,orientation,speed,
fn)
```

Where:
amplitude - the amplitude of the hover motion
angleinc - angle incremention
angle - starting angle
cycles - the number of cycles to move before stopping
orientation - 'v' for vertical, 'h' for horizontal
speed - speed of repetition
fn - (optional) function or statement to execute when complete

Notes: If you want the hover to keep looping then pass null for the cycles value.

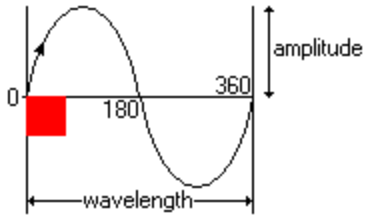
Examples:

```
mylayer.hover.play(60,8,0,1,'v',10)
```

The pause() and stop() Methods:

Work the same as in the circle() method.

Sine Wave Object



The SineWave Object will slide the layer along a sine wave path.

Initialization:

```
objectName.sinewave = new SineWave("objectName","sinewave")
```

Example:

```
mylayer = new DynLayer("mylayerDiv")
mylayer.sinewave = new SineWave("mylayer","sinewave")
```

The play() Method:

The play() method begins the sine wave:

objectName.sinewave.play(amplitude,wavelength,angleinc,angle,cycles,
direction,speed,fn)

Where:
amplitude - the amplitude of the wave
wavelength - length of one wave
angleinc - angle incremention
angle - starting angle
cycles - the number of cycles
direction - use 1 for right, -1 for left
speed - speed of repetition
fn - (optional) function or statement to execute when complete

Notes:
If you want the sinewave to keep looping then pass null for the cycles value.

Examples:

mylayer.sinewave.play(60,200,15,0,2,1,20)

The pause() and stop() Methods:

Work the same as in the circle() method.

Parabola Object

The Parabola Object will slide the layer along a parabolic path.

Initialization:

objectName.parabola = new Parabola("objectName","parabola")

Example:

mylayer = new DynLayer("mylayerDiv")
mylayer.parabola = new Parabola("mylayer","parabola")

The play() Method:

The play() method begins the parabola:

objectName.parabola.play(type,distx,disty,xinc,speed,fn)

Where:
type - 1 is a dropping parabola, 2 is an arcing parabola
distx - horizontal distance
disty - vertical distance
xinc - the number of pixels to move horizontally each repetition
speed - speed of repetition
fn - (optional) function or statement to execute when complete

Examples:

mylayer.parabola.play(1,200,200,5,20)

The stop() Methods:

Work the same as in the circle() method. I didn't bother with a pause method cause I can't see it being useful.

objectName.sinewave.stop()

Gif Animation

When it comes to making dynamic web pages and animations, using animated GIF's would seem to help things. Unfortunately, animated GIF's have no built in controls - you can't start, stop, or pause the animation on command. Although it is possible to mix animated GIF's and non-animated GIF's to mimic the effect, it really doesn't work that well. This makes them unsuitable when trying to do anything complex with them. That is why I made my own Gif Animation Object (GifAnim). It is a piece of code which gives you the kind of control that is necessary for any type of gif animation sequence you can think of.

Although the Gif Animation object is structured similarly to the Dynamic Layer Object, it is an independent object with it's own set of methods. Most likely however, you'll want to mix the 2 objects when creating an animation sequence. The 2 are totally compatible so don't worry about that.

Preloading an Image Series

The GifAnim Object requires that you have a series of preloaded image objects already created that are named something like image0, image1, image2 etc. where 0 is the first frame of the animation, 1 is the second and so on.

I'm going to use the following images for my example:

num0.gif
num1.gif
num2.gif
num3.gif
num4.gif
num5.gif

To preload this Image Series I can use the preload function from the previous lesson.

```
function preload(imgObj,imgSrc) {
            eval(imgObj+' = new Image()')
            eval(imgObj+'.src = "'+imgSrc+'"')
      }
preload('num0','num0.gif')
preload('num1','num1.gif')
```

```
preload('num2','num2.gif')
preload('num3','num3.gif')
preload('num4','num4.gif')
preload('num5','num5.gif')
```

But notice my preload() call is repetitive - I can do the same thing a loop:

```
for (var i=0;i<=5;i++) preload('num'+i,'num'+i+'.gif')
```

That way it can be used for any number of images by changing the for loop arguments.

Whatever is the consistant name in the series is the Image Series name - in my case it would be "num" because each image object name starts with num.

Remember you initially have to show one of the images and define the image name, this is what I'll be using:

```
<DIV ID="numDiv"> <IMG NAME="numImg" SRC="num0.gif" WIDTH=50 HEIGHT=50 BORDER=0> </DIV>
```

Initializing GifAnim Objects

To initialize a GifAnim object, you need to define 5 things:

layer - the layer the image is inside
imgName - the name of the image (in the IMG tag)
imgSeries - the name of Image Series
end - the number of the last frame in the animation
speed - the speed of repetition in milliseconds
startFrame - (optional) the number of which frame to start on (only needed if it starts on a frame other than 0)

The general format for initialization is:

```
objectName = new GifAnim(layer,imgName,imgSeries,end,speed,startFrame)
```

To define my GifAnim object I'll using the object name numImgAnim, so to initialize that animation I'd use:

```
numImgAnim = new GifAnim('numDiv','numImg','num',5,200)
```

If for some reason I wanted the animation to begin on the 3rd frame (image num2.gif) I'd instead use:

```
numImgAnim = new GifAnim('numDiv','numImg','num',5,2)
```

And remember the first image displayed would have to be num2.gif.

Using the GifAnim Methods

play() Method:

The play() method begins the animation. It has the following arguments (in bold is the default):

loop - true/false - whether the animation is to loop or only play once reset - true/false - whether the animation is to reset to the first frame when complete fn - function or statement to execute when complete

These arguments are optional, if you don't specify any of them:

objectName.play()

it doesn't loop, doesn't reset, and does nothing when complete. But you can assign the ones you want:

objectName.play(true) objectName.play(true,false) objectName.play(false,true,'alert(\'done\')') // alert(\'done\') can be replaced with anything

stop() Method:

The stop() method simply stops, or more accurately pauses, the animation:

objectName.stop()

Depending on the arguments in the play() method it will do different things when it is stopped. Like if the reset argument in the play() method is true, it will return to the first frame. If reset was false, the next time you start the animation it will continue where it left off. And if fn was defined it'll evaluate it. If it was a looping animation, the fn will only evaluate after it is stopped.

goToFrame() Method:

The goToFrame() method brings the animation to whichever frame you want.

objectName.goToFrame(index)

Where index is the index number of the image Series - 0 is the first image in the series, 1 is the second and so on.

There is another method, run(), which is the logic behind the GifAnim object, but you should never have to call that method because it doesn't do any error checking to make sure that multiple instances of an animation get executed.

Path Animation [The Path Object]

Path Animation Concepts:

From a programming point of view, doing animation using a path or timeline is sort of like cheating. All you do is define a path for the

layer to follow, and it just loops through each point by just moving it there rather than calculating where to move it. This has some big advantages because it takes little CPU time (so they go really fast) and you can move the layer in any way you can think of. The disadvantage though, is that you cannot easily change the path like you can a calculative animation. If you need to change it, you have to go back and change all the points in your path. However, with that said, using a path is still a decent way of making animation.

The basic technique for path animation is easy to understand. Once you have all the co-ordinates of where to move the layer, you just need a function to loop through them and move the layer to those points. For this purpose I've written a Path Object which will take care of this for you. You just have to add the object to your DynLayers and use the appropriate methods.

The only problem then is to get your coordinates for your path. There are several commercial products that will do this for you such as Macromedia Dreamweaver or mBed Interactor. These pieces of software will auto-generate all code for you and require their own gigantic library files. And if you want to edit the JavaScript code by yourself you'll have to invest quite a bit of time to fiqure out exactly what it going on. If all you want to do is make a simple path animation in a small demo, then paying a couple of hundred dollars for it seems a little extreme. This is why I've designed my own little tool, called DuoPath, which a) doesn't cost anything, b) makes it easy to get all the co-ordinates, and c) lets you do all the coding so you understand what's going on and can easily change it to your needs. The next lesson will show how to use DuoPath, so I'll just continue on assuming that we've already created our path and have all the co-ordinates ready to go.

Here's all the coordinates for a path that I made with DuoPath:

x-coords:
101,105,113,122,131,140,149,156,157,156,155,157,163,172,181,188,196,203,
208,215,221,227,234,243,252,261,270,278,287,296,303,310,315,319,321,322,322,322
,322,323,328,335,345,355,368,377,385,390,392,393,394,391,387,381,377,376,376,37
8,382,386,391,398,406,414,424,434,442,453,461,468,471,474,475,476,476,
475,474,473,472,472,476,481,488,498,508,515,523,529,536,539,542,542,541,539,535
,529,523,517,514,513,513,520,530,540,552,564,574,579,581,580,576,567,555,540,52
1,501,479,459,441,422,404,384,366,349,330,310,291,272,254,238,222,206,188,172,1
57,141,128,113,102,92,82,72,63,55,48,42,36,32,29,27,29,32,39,
48,59,67,74,81,87,91,97

y-coords:
285,271,255,242,230,225,225,233,242,253,265,275,281,284,280,269,256,239,224208,
194,181,168,154,143,135,129,126,124,125,127,130,135,140,149,161,174,192,208,229
,250,269,281,286,284,277,269,259,248,237,222,202,183,164,147,133,120,108,99,92,
86,81,78,76,77,82,90,100,116,136,157,177,200,225,249,270,289,306,323,341,354,36
5,371,376,376,374,368,361,351,340,324,309,295,281,269,257,245,231,218,202,185,1
70,159,153,148,142,133,120,106,92,80,69,59,50,44,40,37,35,34,34,34,36,37,39,42,46
,50,54,60,67,76,84,95,105,116,128,140,153,166,176,187,200,212,224,237,250,264,27
9,295,310,326,341,351,357,355,350,342,331, 320,310,299

As you see there's about a million of them. But it doesn't really matter how many points there are because remember path animation takes no CPU time, so making a lot of points is no problem. Each set of co-ordinates represent one point in the path. The first x value and the first y value make up the first point and so on. By assinging these numbers to an array we will be able to access any single set by using their indexes:

```
path1x = new
Array(101,105,113,122,131,140,149,156,157,156,155,157,163,172,181,188,
196,203,208,215,221,227,234,243,252,261,270,278,287,296,303,310,315,319,321,322
,322,322,322,323,328,335,345,355,368,377,385,390,392,393,394,391,387,381,377,37
6,376,378,382,386,391,398,406,414,424,434,442,453,461,468,471,474,475,476,476,4
75,474,473,472,472,476,481,488,498,508,515,523,529,536,539,542,542,
541,539,535,529,523,517,514,513,513,520,530,540,552,564,574,579,581,580,576,567
,555,540,521,501,479,459,441,422,404,384,366,349,330,310,291,272,254,238,222,20
6,188,172,157,141,128,113,102,92,82,72,63,55,48,42,36,32,29,27,29,32,
39,48,59,67,74,81,87,91,97)

path1y = new
Array(285,271,255,242,230,225,225,233,242,253,265,275,281,284,280,269,
256,239,224,208,194,181,168,154,143,135,129,126,124,125,127,130,135,140,149,161
,174,192,208,229,250,269,281,286,284,277,269,259,248,237,222,202,183,164147,133
,120,108,99,92,86,81,78,76,77,82,90,100,116,136,157,177,200,225,249,270,289,306,
323,341,354,365,371,376,376,374,368,361,351,340,324,309,295,281,
269,257,245,231,218,202,185,170,159,153,148,142,133,120,106,92,80,69,59,50,44,4
0,37,35,34,34,34,36,37,39,42,46,50,54,60,67,76,84,95,105,116,128,140,153,166,176,
187,200,212,224,237,250,264,279,295,310,326,341,351,357,355,350,342,
331,320,310,299)
```

Now if we wanted to know the 10th x value and the 10th y value, you check value of path1x[9] and path1y[9]. The index number is always 1 minus the point we check because arrays start at zero. So path1x[9]=156 and path1y[9]=253.

With this knowledge in mind, you can apply these coordinates to the Path Object which will allow you to turn your coordinates into a path animation.

The Path Object

Initialization:

The Path Object is an addon object to the DynLayer. It works similarly to the way my Geometric Objects work. It's probably best to include the code for each of the DynLayer and the Path Object as js files:

```
<SCRIPT LANGUAGE="JavaScript" SRC="dynlayer.js"></SCRIPT> <SCRIPT
LANGUAGE="JavaScript" SRC="path.js"></SCRIPT>
```

Once you've created a DynLayer, you create the Path Object on top of it. You have to pass the Path Object the name of the DynLayer, and the name of the Path Object along with the path information. This is necessary so that the Path Object can manipulate the DynLayer.

objectName.pathName = new Path(dynlayer,name,arrayX,arrayY)

Where:
dynlayer - name of the DynLayer
name - name of the path (in quotes)
arrayX - array of the x-coordinates
arrayY - array of the y-coordinates

Example:

```
mylayer = new DynLayer("mylayerDiv")
mylayer.path1 = new Path(mylayer,'path1',
new Array(0,10,20,30),
new Array(0,10,20,30))
```

Optionally you could predifine the arrays and just send their names for the arrayX and arrayY values. Once the Path Object is initialized, the parameters become properties of the DynLayer/Path Object and can be changed at any time if need be (ie. mylayer.path1.arrayX = new Array(5,10,15,20)).

The play() Method:

Begins the animation.

objectName.pathName.play(loop,speed,fn)

Where:
loop - boolean value determining whether to loop the animation
speed - speed of repetition in milliseconds
fn - function or statement to execute when complete

All the parameters of the play() method are optional, if you do not specify them it will default to play(false,30,null)

Example:

mylayer.path1.play(true,50)

The pause() and stop() Methods:

Self-explanitory:

objectName.pathName.pause() or
objectName.pathName.stop()

Using DuoPath 1.12

DuoPath is a freeware JavaScript application for making DHTML path animation for Netscape 4.0 and Internet Explorer 4.0. DuoPath is intended to be used along with the technique shown in the Path Animation

lesson - it'll explain the concepts of how DuoPath can be used for your own purposes.

First of all, launch DuoPath and follow this lesson and switch between the windows to understand what I'm talking about.

Launch DuoPath 1.12 Warning: DuoPath will take a little while to initialize so don't be alarmed if your browser seems to do nothing for 10 seconds or so - just be patient and it should load up eventually. Although DuoPath does work in Internet Explorer 4.0, I highly recommend using Netscape 4.0 because it's much faster.

DuoPath is included along with the tutorial when you download it.

Overview of New Features

Version 1.12: Oct 15. 1999

Again updated the output, now requires you to manually include the dynlayer.js and path.js files from the DynAPI

Version 1.11:

Updated the output of DuoPath to coincide with my updated DynLayer and Path Objects.

Version 1.1:

DuoPath 1.1 is basically the same as 1.0 but with a few enhacements:

- Curve Mode - A very powerful set of tools to create smooth curved paths. All you do is set a few control points and DuoPath will draw in a smooth curve based on those points.
- Full Drag and Drop - The "move" tools are now full drag and drop instead of double click as in version 1.0
- Updated Output - When you generate the HTML DuoPath will now use the updated path scripts that are documented in the Path Animation lesson.

How to Use DuoPath

I think DuoPath is simple enough to use that you'll understand how to use it almost immediately. The basic idea is you click around the screen plotting new points for your path. Once you're done you can generate the HTML to create a simple demo using your path. I'll do a quick summary of how to use the features in case something seems a little odd.

Edit Mode: - controls the points in your path

New - inserts a new point to the path
Move - moves a single point to a new location
Move All - moves all the points to a new location
Erase - erases a single point anywhere in the path
Erase Last - erases the last point in the path

Insert - inserts a single point in the middle of the path
Line - creates a straight line of points betwen 2 locations
Circle - creates a circle/oval starting at the last point
Parabola - crates a parabola starting at the last point
Curve - goes into curve mode
Info - retrieves the number, and x and y location of a point

Notes:

Inserting a point is not the same as making a new point. A new point is appended to the end of the path. Inserting a point puts another point between 2 points already in the path. You first have to click the point that comes after the point you are inserting. This will attach a point to the cursor, then you click again to drop that point into the path.

For the line mode, first click where you want the line to begin. Then click again for where you want the line to end. A dialog will pop up and ask how many pixels apart the points will be spaced, then it'll insert them for you.

If you're not going to take my advice and use Netscape, then you will have to be aware that in IE when you click on the scrollbars it will insert a point if you are in "New" or "Line" mode. I guess IE thinks that the scrollbars are part of the document for some reason. To avoid putting points where you don't want them just go into one of the other modes before scrolling.

Curve Mode:

Curve mode allows you to draw a perfectly smooth curve of points based on just a few control points which you must define.

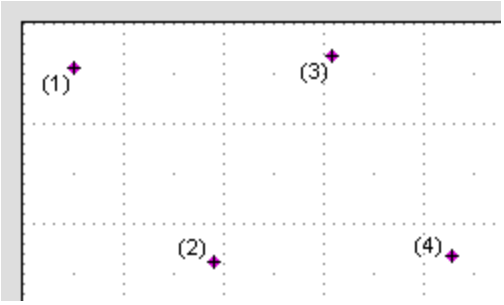
The curve tools:

new point - new curve point
move point - move curve point
erase point - erase curve point
move curve - move all curve points
Path Points - sets the number of path points for each curve point (the distribution)
erase curve - erases the whole curve
new curve - finalizes the current curve and starts a new one

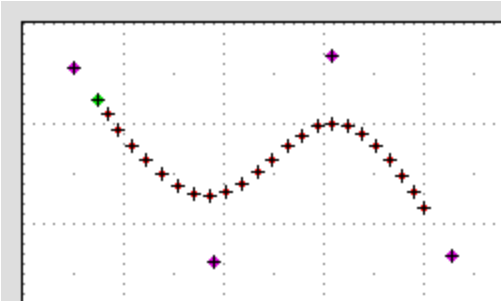
You work with curve points just as you would the normal path points. However, once you put down 4 curve points DuoPath will draw in a series of path points based on where your curve points are. As you continue to add more curve points it will redraw the curve accordingly. The curve points (in purple) are used to influence how the curve bends. You can move the curve points and reshape the curve to your liking.

The curve tools should be self-explanitory except for "Path Points" - that's to determine how many actual path points get put into your curve. A higher number means there will be more points in the curve and therefore the animation will move smoother and slower.

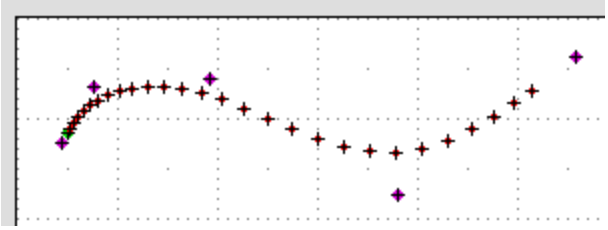
Here's a "before" shot of some points labeled in the order they were clicked:



The first 3 points won't do anything, but once the forth one is down it'll draw the curve for you:



By positioning the curve points closer or farther apart you can create an acceleration effect. The picture below shows a path that will start off slow and speed up as it swooshes away:



Special Buttons

Preview - allows you to preview the animation to make sure if the locations of the points are correct and the speed is suitable
Generate - generates an HTML page with all the coding already done for you
Load Source - loads an HTML page into the workspace (Netscape only)
Load Path - allows you to insert a path that you created in the past
New Path - erases all the points

Inputing Points:

This is a way to work on a path that you created earlier, and want to edit some more. 2 dialog boxes will appear, the first asking for the x values, and the second for the y values. You have to manually cut and paste the numbers from your previous path into these dialogs. To

demonstrate this feature just copy the first line of numbers below (including the commas) by putting your cursor in front of the first number. Then hold shift and press the END key. Then copy the numbers (Ctrl-C, depending on your platform). Then switch to DuoPath and in the Input Points dialog paste the numbers in (Ctrl-V). Then hit "Okay" and do the same for the y values (the second row of numbers). DuoPath will then draw out all the points and you can continue editing them.

Preview

The preview mode is pretty cool. You can instantly see what your animation is going to look like by playing the animation right inside DuoPath. You can change characteristics of the animation - whether it loops or not, or change the speed of repetition.

Generate HTML

With a simple click and a few options to set you can have all the path animation code generated for you on the fly. It has options for setting the speed, the name of the object and so on. The code is basically the same as shown in the Path Animation lesson but can also contain links for playing, pausing, and stopping the animation - just as in DuoPath's preview mode. Remember that by no means you have to use this code, you can still just take the co-ordinates themselves and do whatever else you want with them.

Other Tips:

Don't hesitate to make tons and tons of points. It makes the animation smoother and DuoPath is designed so that there's no limit to how many points you can have. Every 100 points you lay down it'll refresh itself to hold more - don't worry, all your points will remain in tact.

Also if DuoPath ever messes up - like a error occurs or something - you can at anytime reload the workarea frame and it should correct the problem.

DynAPI Mouse Events

Just as you can capture keyboard events, you can capture mouse events like onMouseDown, onMouseUp, and onMouseMove - there are other if you want to read up on them but these are the most important ones. For each of these events, you can obtain the location of the mouse and use those coordinates to move a layer.

The first part will explain the basics of mouse events in a cross-browser setting, the second part will explain the mouse event functions used for elements in the DynAPI.

Overview of Document Mouse Events:

Each of onMouseDown, onMouseUp, and onMouseMove are initialized the same way. Here's the set up I like to use:

```
function init() {
            document.onmousedown = mouseDown
            document.onmousemove = mouseMove
            document.onmouseup = mouseUp
            if (is.ns) document.captureEvents(Event.MOUSEDOWN |
Event.MOUSEMOVE | Event.MOUSEUP)
      }

      function mouseDown(e) {
      }
      function mouseMove(e) {
      }
      function mouseUp(e) {
      }
```

The function names can be whatever you want, but to keep it consitant I will always use mouseDown(e), mouseMove(e), and mouseUp(e). Netscape 4 has a different event model than IE which requires you to "capture" events before they can be used. In IE they are always captured.

Getting the Mouse Coordinates

The "e's" in each function represent the built-in Event object for Netscape. It is how Netscape obtainss the location of the mouse:

```
function mouseDown(e) {
        if (is.ns) {
         var x = e.pageX var y = e.pageY
        }
}
```

IE will ignore the e's because it uses a slightly different system for capturing mouse events. IE has an "window.event" object which handles all events. The window.event object contains the properties x and y which represent the location of where the mouse event occured:

```
function mouseDown(e) {
        if (is.ns) {
         var x = e.pageX
         var y = e.pageY
        }
        If (is.ie) {
        var x = event.x
        var y = event.y
        }
}
```

Those values reflect where in IE's browser window the mouse event occured - it does not necessarily reflect exactly where on the document has been clicked. If you scroll down the window.event.y value isn't in synch with the document, so we have to account for that discrepency ourselves. You add the amount that the document has been scrolled by using document.body.scrollTop.

```
var x = event.x+document.body.scrollLeft
var y = event.y+document.body.scrollTop
```

What I usually like to do is compact the x-y capturing with these 2 lines:

```
var x = (is.ns)? e.pageX : event.x+document.body.scrollLeft
var y = (is.ns)? e.pageY : event.y+document.body.scrollTop
```

Now when any of the events occur we can work with the x and y variables (the current location of the mouse) and do some neat things with them.

An improvement I like to do is only allow left mouse button clicks. In Netscape you check for e.which, and IE you check for event.button:

```
function mouseDown(e) {
            if ((is.ns && e.which!=1) || (is.ie && event.button!=1)) return true
            var x = (is.ns)? e.pageX : event.x+document.body.scrollLeft
            var y = (is.ns)? e.pageY : event.y+document.body.scrollTop
      }
```

I like to do this because the default action when right clicking your mouse is to pop open the command menu (Back, Forward etc). This interferes when coding mouse events, so I don't do anything when you right click.

Return Values:

The return values in the mouse events are very important for Netscape. When you click the mouse somewhere on a page, the document recieves the event first (and thus the mouseDown event is triggered). Even if you are clicking on a hyperlink the mouseDown event will be called. By returning true in the event handler you allow other elements on the page use that event. If you return false you stop the page from doing anything else with that event. So for example if you were to always return false in your mouseDown handler:

```
function mouseDown(e) {
       return false
}
```

Than you would never be able to click on a hyperlink - which is obviously not desired. Always returning true isn't the answer either because sometimes elements on the page will interfere with what you want to do with the mouse. Also, on MacIntosh's there is no left mouse button. So to open the command menu you must hold down the mouse for about a second before it pops up. This causes major problems. But by returning false in the mouseDown handler it will stop this from occuring.

The mouseMove event is a bit of a pain as well. On the document, the mouseMove event is what allows you to select/highlight text on the page (for copy/paste reasons). By returning false in the mouseMove event you

disable Netscape from being able to highlight the text.

So you must carefully place a return false only when you are specifically making use of the event, all other times you return true. Notice I return true when checking if the left mouse button was not clicked (I let the document use the mouseDown event).

In the end, these are the mouse functions that work pretty good

```
function init() {
        document.onmousedown = mouseDown
        document.onmousemove = mouseMove
        document.onmouseup = mouseUp
        if (is.ns4) document.captureEvents(Event.MOUSEDOWN |
Event.MOUSEMOVE | Event.MOUSEUP)
    }

    function mouseDown(e) {
        if ((is.ns && e.which!=1) || (is.ie && event.button!=1)) return true
        var x = (is.ns)? e.pageX : event.x+document.body.scrollLeft
        var y = (is.ns)? e.pageY : event.y+document.body.scrollTop
        // your code goes here
        return true
    }
    function mouseMove(e) {
        var x = (is.ns)? e.pageX : event.x+document.body.scrollLeft
        var y = (is.ns)? e.pageY : event.y+document.body.scrollTop
        // your code goes here
        return true
    }
    function mouseUp(e) {
        var x = (is.ns)? e.pageX : event.x+document.body.scrollLeft
        var y = (is.ns)? e.pageY : event.y+document.body.scrollTop
        // your code goes here
        return true
    }
```

DynAPI Mouse Events

To replace the need to manually insert the above functions, I've created a default JavaScript file that I'll be using in demos that require document mouse events. The particular sections are the Drag object and the Scroll2 object.

To use the DynAPI mouse events include the mouseevents.js file after the dynlayer.js and BEFORE the drag.js and scroll2.js if you will be using them:

```
<script language="JavaScript" src="../dynlayer.js"></script>
<script language="JavaScript" src="../mouseevents.js"></script>
<script language="JavaScript" src="../drag.js"></script>  // if needed
<script language="JavaScript" src="../scroll2.js"></script>  // if needed
```

These files will take care of retrieving the co-ordinates of the mouse, and the handling required to operate the Scroll2 and the Drag object. To initialize these you'll call the initMouseEvents() function in the init():

```
function init() {
	initMouseEvents()
}
```

Instead of having your own mouseDown, mouseMove, and mouseUp functions, you can now have simpler means of obtaining mouse coordinates by incorporating the DynMouseDown, DynMouseMove, and DynMouseUp functions:

```
function DynMouseDown(x,y) { // your code for mousedown return true }

function DynMouseMove(x,y) { // your code for mousemove return true }

function DynMouseUp(x,y) { // your code for mouseup return true }
```

These functions are optional. If the mouse events are only to be used by a Scroll2 or the Drag, then they don't even have to be included in your source code because the above empty default functions are already in the MouseEvents code.

Drag and Drop Concepts

Drag and drop scripts are entirely based around the mouse events. I will be using the DynAPI Mouse Event code, so if you haven't already, read that lesson to know what I'm talking about. This lesson will show step by step how to make one layer draggable. The next lesson will show how to make any number of layers draggable with a generic Drag Object.

A drag and drop sequence is handled like so:

mouseDown - check if the mouse has click on a layer, activate the mouseMove
mouseMove - move the layer to cooincide with the location of the mouse
mouseDown - stop the mouseMove thus ending the drag sequence

Setting up the Layer:

I will be using a 50x50px layer named "square" and defining a DynLayer to it named "dragObject"

```
function init() {
	dragObject = new DynLayer("square")
	dragObject.dragActive = false
	initMouseEvents()
}
```

Notice I have tacked on a dragActive property to the DynLayer. This boolean property will represent whether the layer is currently being dragged.

The DynMouseDown Handler

The first stage in the drag sequence is to check if you have clicked on the layer or not. To do this you simply compare the x-y coordinate of the mouse to the edges of the layer:

```
if (x>=dragObject.x && x<=dragObject.x+dragObject.w && y>=dragObject.y
&& y<=dragObject.y+dragObject.h)
```

If we have indeed clicked on a layer we will begin the actual dragging of the layer. All that's needed in the mouseDown is to set the dragActive flag to true:

```
if (x>=dragObject.x && x<=dragObject.x+dragObject.w && y>=dragObject.y
&& y<=dragObject.y+dragObject.h) {
        dragObject.dragActive = true
        return false
}
```

Notice that I have placed a return false in the block if we have clicked on a layer. This stops Netscape from using the mouseDown event for anything else (including a MacIntosh mouse-hold).

The full DynMouseDown handler looks like this:

```
function DynMouseDown(x,y) {
            if (x>=dragObject.x && x<=dragObject.x+dragObject.w &&
y>=dragObject.y && y<=dragObject.y+dragObject.h) {
                  dragObject.dragActive = true
                  return false
            }
            else return true
      }
```

The DynMouseMove Handler

The mouseDown handler on it's own won't do anything to the layer, but by setting the dragActive flag to true we have a way of turnging the mouseMove action on and off as we please. The mouseMove event simply checks if the dragActive flag is true, and if so moves the layer to the coordinates of the mouse:

```
function DynMouseMove(x,y) {
        if (dragObject.dragActive) {
                dragObject.moveTo(x,y) return false
                }
        else return true
}
```

As soon as dragActive is set to true the mouseMove function will begin moving the layer. Again note the placement of the return false, it is important. While we are dragging the layer around we do not want Netscape to use the mouseMove event for anything else (such as selecting

text).

The DynMouseUp Handler

To end the drag sequence all you need to do is set the dragActive flag back to false. This stops the mouseMove function from moving the layer:

```
function DynMouseUp(x,y) {
        dragObject.dragActive = false
        return true
}
```

In this case no return false is necessary. It doesn't matter if Netscape handles the mouseUp event anymore because we have already stopped the drag sequence.

Accounting for the Offset Values

You'll notice in that example that if the layer is moved directly to the location of the layer it doesn't look right. The layer pops to the corner regardless of where it was click on. We can account for this situation and correct it by capturing the difference between the location of the layer, and the coordinate of the mouse (the offset values). This is done in the mouseDown function:

```
if (x>=dragObject.x && x<=dragObject.x+dragObject.w && y>=dragObject.y &&
y<=dragObject.y+dragObject.h) {
            dragObject.dragOffsetX = x-dragObject.x
            dragObject.dragOffsetY = y-dragObject.y
            dragObject.dragActive = true
            return false
      }
```

By tacking on the dragOffsetX and dragOffsetY properties we have captured the offset values and can utilize them in the mouseMove handler to move the layer accordingly:

```
if (dragObject.dragActive) {
            dragObject.moveTo(x-dragObject.dragOffsetX,y-dragObject.dragOffsetY)
            return false
      }
```

Drag Object

Revision:

I pulled the drag mouse events into a "mouseevents.js" file, and created onDragStart(), onDragMove(), and onDragEnd() event handlers. built a "dropping" mechanism to track when you've dropped a layer on top of another layer (a drop target)

The Drag Object is a unified piece of code which allows you to

selectively make Dynamic Layers draggable with a minimal amount of coding. All that's needed is to set up the drag.js file, initialize your DynLayers, and then add them to the drag object.

Setting Up The Script

The Drag object is based around the DynLayer and the DynAPI Mouse Events. You simply add DynLayers to the Drag Object to make them draggable, and remove them from the drag object to make them static again.

The required DynAPI scripts are:

```
<script language="JavaScript" src="../dynlayer.js"></script>
<script language="JavaScript" src="../mouseevents.js"></script>
<script language="JavaScript" src="../drag.js"></script>
```

Make sure to have the drag.js after the mouseevents.js

The Drag Object code will automatically intialize a generic "drag" object which is the default (ie. you don't have to insert this code):

```
drag = new Drag()
```

However, being that this is an object you could create multiple drag objects to define different groups of draggable layers.

Mouse Event Handling

The mouse event handling for the Drag Object is now taken care of by the new DynAPI Mouse Events code. All you have to do is include the mouseevents.js file and call the initMouseEvents() function:

```
function init() {
        // initialize DynLayers here
        initMouseEvents()
}
```

The mouse handling only takes care of the default "drag" object. If you have other Drag objects you'll have to include a call to yourdrag.mouseDown(x,y), yourdrag.mouseMove(x,y), and yourdrag.mouseUp(x,y) into each of the DynMouseDown() Move() and Up() functions.

The Drag Object's add() method is what you use to make your layers draggable. The usage is pretty simple:

```
drag.add(dynlayer1, dynlayer2, etc...)
```

Where dynlayer1, dynlayer2, etc... is the names of your DynLayers. The method will accept any number of DynLayers in a row, or you can add them separately. As soon as they're added they will become draggable, this can be done at any time after the page has been loaded. The following init() function will make each of the DynLayers draggable as soon as the

page is finished loading:

```
function init() {
        // initialize DynLayers
        blue = new DynLayer("blueDiv")
        red = new DynLayer("redDiv")
        green = new DynLayer("greenDiv")
        purple = new DynLayer("purpleDiv")

        // add the draggable layers to the drag object
        drag.add(bluered,green,purple)

        initMouseEvents()
    }

initMouseEvents()

}
```

That's all that's necessary to get your drag and drop layers working.

Extra Functionality

remove() Method:

You can remove a DynLayer from the Drag Object, and therefore making it no-longer draggable, by using the remove() method. It's syntax is the same as the add() method:

drag.remove(dynlayer1, dynlayer2, etc...)

resort Property:

The resort property determines whether the layer that is being dragged will be layered on top of all the other layers. By default, when you click a draggable layer the Drag Object will make the z-index of that layer higher than all the rest. This may not be what you want, so you can turn it off by calling:

drag.resort = false

setGrab() Method:

The setGrab() method allows for only a portion of the layer to be "grabbable", this allows for draggable toolbars, or window-like layers (see DynWindow Object).

drag.setGrab(dynlayer,top,right,bottom,left)

This method is entirely optional, if you don't call it the entire layer will be "grabbable"

checkWithin(x,y,left,right,top,bottom) Function:

The checkWithin() function can be used to check if a particular coordinate is with a certain boundary. This function is used by the Drag Object to determine when a layer has been clicked on. But it can also be used for other things such as determining if you've dropped the object onto a particular area of the page.

To use the checkWithin() function you need a test coordinate (x and y) and you compare that to 4 other values (left,right,top,bottom) which represent a square portion of the page:

checkWithin(x,y,left,right,top,bottom)

checkWithinLayer(x,y,lyr) Function:

Same as checkWithin() except it checks if x,y is within the drag boundaries of the DynLayer (lyr). This can only be used for DynLayers that are part of the Drag Object either as a drag layer or a drop target layer.

Drag Events

If you want to "do stuff" before you drag a layer, while you're dragging a layer, or when you're finished dragging the layer, you'll need to implement the onDragStart, onDragMove, and onDragEnd event handlers respectively.

```
drag.onDragStart = startHandler
drag.onDragMove = moveHandler
drag.onDragEnd = endEndler
```

In your handler functions you can use any of the properties of the drag object to manipulate

Drop Targets

Making drop targets allow you to easily determine if your dragging layer has been dropped on top of another. For example if you had a shopping cart and wanted to drop an item onto the basket you'd make the layer with the basket a drop target.

You do that with the addTarget() method:

```
drag.addTarget(target1,target2,target3) // target1,2,3 are DynLayers
```

Then to do something when a drag layer has been dropped on the target, you must implement a onDrop event handler:

```
drag.onDrop = dropHandler
```

If you have multiple targets, you can determine which target was dropped on with the drag.targetHit property:

```
function dropHandler() {
```

```
        if (this.targetHit == target1)
        alert("you hit target 1")
}
```

Creating and Destroying Layers

In both the browsers there is a way to add new layers to the page on the fly (after the page is fully loaded). Again this is a situation where the solution is completely different between the browsers. However there is one crippling obstacle that I haven't been able to get around which makes it impossible to use this technique to it's full potential.

Netscape:

Creating a New Layer in Netscape:

In Netscape, to add a new layer to the page you simply create a new Layer object:

```
document.layers[id] = new Layer(width)
```

For a nested layer you must call the new Layer command like this:

```
document.layers[parentLayer].document.layers[childLayer] = new
Layer(width, document.layers[parentLayer])
```

Thanks to Bill Sager for showing me how to do that.

After the layer is created you can then assign the properties and add the content to the layer using JavaScript:

```
document.layers[id] = new Layer(400) document.layers[id].left = 40
document.layers[id].top = 100 document.layers[id].height = 300
document.layers[id].bgColor = "blue" document.layers[id].visibility =
     "show" document.layers[id].document.open()
document.layers[id].document.write("This is my content")
document.layers[id].document.close() etc.
```

Once all this is done, you can use the layer as normal.

Removing a Layer in Netscape:

Unfortunately there is no way that I know to truely delete a Layer in Netscape. So the closest thing we can do is simply hide the layer.

```
document.layers[id].visibility = "hide"
```

Want a challenge?

It is theorized that a solution could be created which keeps track of which layers have been deleted (keep track of their indexes), and then when you create new layers re-assign the indexes of deleted ones. So if you want a challenge that no one has been able to solve yet here's your chance! Please notify me of any solutions, even partial experimental

ones.

Internet Explorer

Creating a New Layer IE:

Internet Explorer's ability to work with HTML as if it were a string allows you to add more layers as you please. I recommend this be done using IE's insertAdjacentHTML(). If you use the innerHTML property it will cause some unexpected results.

To add another layer (or any other HTML for that matter) to the body of the document, you call the method in this manner:

document.body.insertAdjacentHTML(string)

Where string is a string of text or HTML that needs to be appended to the end of the page. So to create a layer you can pass a DIV tag with the style assign using the old inline technique if you prefer (remember IE doesn't have problems with inline styles):

document.body.insertAdjacentHTML(' <DIV ID="layerName" STYLE="position:aboslute; left:40; top:100;"> This is my content </DIV>')

To create a nested layer you can apply the insertAdjacentHTML() method to the parent layer just as you do the body of the document:

document.all[id].insertAdjacentHTML(string)

Removing a Layer in IE:

Initially I had though that the only way to delete a layer in IE was to do string manipulation to the document.body.innerHTML property of the page. However this creates some severe problems as "phantom" HTML elements get introduced. Fortunately, as a few other JavaScripters (Erik Arvidsson and Thomas Brattli) mentioned, there indeed is a pretty easy way to delete a layer in IE. You can use a combination of innerHTML and outterHTML on that particular layer only. It does work, and does not cause the problem seen when using document.body.innerHTML.

To remove a layer you can do these 2 commands:

document.all[id].innerHTML = "" document.all[id].outerHTML = ""

The createLayer() and destroyLayer() Functions

createLayer(id,nestref,left,top,width,height,content,bgColor,visibility, zIndex)

destroyLayer(id,nestref)

The usage should be obvious - id and nestref they are the same as for a Dynlayer. The left, top, and width properties are required, the rest are

optional. After you create the layer you could assign DynLayers to them and work with them that way.

CGI Communication

Although DHTML in the version 4 browsers is not specifically geared to interact with a server-side process, using some tricks it can be accomplished. This is particularly useful for building true applications using DHTML. You could create a wide range of applications such as some really nifty shopping carts, or a nice DHTML interface to a database, or possibly even a chat room if you're inclined to put in the time to write one. Following the guidelines I will explain here these things are all theoretically possible.

There are few ways to accomplish the task:

Hidden Frame

It's been long known that if you hide a frame you can target your form values to that frame, and the other page will stay loaded. You could submit a form, and have the CGI submit back a page containing JavaScript code that updates the other frame by rewriting layers or whatever. This is the easy way to do it, it's pretty straight-forward, so I won't cover it for now.

Java Client-Server Interaction (Servlet)

This would probably work very well. However you have to be using a server that supports Servlets and it would require a fair amount of work to accomplish. I may play around with creating a servlet and seeing if it indeed would work. The process would be like this:

JavaScript
Use LiveConnect execute functions in the applet

Applet
applet communicates directly with the server-side java

Servlet
servlet processes request (writes files, calls database, whatever) and then communicates back to the applet

Applet
applet sends the JavaScript the results

JavaScript
display the results from the process by writing layers

Submitting Forms to Layers

The basic technique to use to submit a form to a layer is really just a derivative of the load external files technique. You load in external

files that are generated by a Perl script or equivalent server-side process like ASP, PHP, or any number of web database languages like ColdFusion, Domino, Oracle etc. The fun part is, you can't send any information to the CGI by actually submitting the form (the way CGI scripts usually work). The whole point of developing a DHTML interface is to make the page static, but keep the information contained in it dynamic and updatable. This means you cannot ever change the location of the page and you must load files (containing new information generated by the CGI) into layers contained within the static page.

To communicate with a server using a regular CGI process there is only one solution: query strings!!!.

Your CGI script must be able to accept query strings instead of regular form parsed values. We will use query strings to pass all the relevant data to the CGI - I will be using a simple Perl script to accomplish this task.

Gathering the Data

The easiest way to gather your data is through HTML Forms. But you must realize that this is not the only way to gather data. You could create your own GUI elements to switch widget-like images on and off or any other crazy ideas you have. For simplicity in this example I'll just use a simple form that asks what your favorite operating system is:

```
<form name="myform">
<p>My Favourite Operating System is:
<p><input type="Radio" name="os" value="win9x">Windows 95/98
<br><input type="Radio" name="os" value="winnt">Windows NT 3.5/4.0
<br><input type="Radio" name="os" value="mac">MacOS 7/8
<br><input type="Radio" name="os" value="linux">Linux
<br><input type="Radio" name="os" value="solaris">Solaris
<br><input type="Radio" name="os" value="freebsd">FreeBSD
<br><input type="Radio" name="os" value="beos">BeOS
<br><input type="Radio" name="os" value="handheld">Handheld (PalmOS/WinCE)
<br><input type="Radio" name="os" value="otherunix">Other Unix-based
<p>
<input type="button" value="Submit" onClick="submitForm()">
</form>
```

Notice there is not ACTION associated with the Form tag, that's a no-no. You must create a JavaScript function of some sort to collect your data into individual variables. In my case, I just need to know which operating system was selected. So I created a submitForm() function to get that value when the Submit button is clicked:

```
function submitForm() {
        for (var i=0;i<document.myform.os.length;i++) {
            if (document.myform.os[i].checked) {
                var os = document.myform.os[i].value
                break
            }
        }
```

```
        alert(os)
    }
```

Getting The CGI Process To Work

Before we go head first into writing a big Perl script, we better do a little test to make sure this will in fact work. What I did was create a "results" layer which will contain the Perl-generated external page. To load the page I'll use the DynLayer load() method. Because the load() method uses an IFrame called "bufferFrame" we must include that as well. If you didn't read the DynLayer load() section yet, I recommend you do so now to understand what I'm doing.

The CSS (auto-generated via the css() function):

```
writeCSS ( css('resultsText',250,30)+
css('resultsDiv',250,50,200,100,'#c0c0c0') )
```

The Div's:

```
<iframe style="display:none" name="bufferFrame"></iframe>

<div id="resultsText"><b>Results Layer:</b></div> <div
id="resultsDiv"></div>
```

In order to use the DynLayer load() method, we must have both the dynlayer.js and the dynlayer-common.js (which contains the load function) in the page:

```
<script language="JavaScript" src="../dynlayer/dynlayer.js"></script>
<script language="JavaScript"
src="../dynlayerext/dynlayer-common.js"></script>
```

The resultsDiv layer must be initialized (DynLayerInit() can be used) and the load() method applied to it:

```
function init() {
      DynLayerInit() results.load = DynLayerLoad
}
```

When the form is submitted, we must change the location - by using the load() method - directly to the Perl script:

```
function submitForm() {
        for (var i=0;i<document.myform.os.length;i++) {
            if (document.myform.os[i].checked) {
                var os = document.myform.os[i].value
                break
            }
        }
        results.load("/cgi-bin/dynduo/cgicomm-test.pl")
    }
```

For this "test" case, the cgicomm-test.pl script can be simple, it just writes out a simple page that does what all external files must do - call back to the layer it's being loaded to, to complete the loading sequence.

```
#!/usr/local/bin/perl

print "Content-type: text/html\n\n";
print "<html*gt;<body onLoad=\"parent.results.loadFinish()\"*gt;\n";
print "This text came from a perl script!";
print "</body*gt;</html*gt;\n";
```

View cgicomm2-cgitest.html [source] - to view a preliminary test of the DHTML-to-CGI communication technique.

As you can see this process works fine, so lets finish it up...

Finalizing The Perl Script and Query Strings

For this example the Perl script doesn't really do anything except gather the query strings (only one string actually - "os"), and then prints out a page.

```
#!/usr/local/bin/perl

	# Get the query strings
	@qsets = split (/&/,$ENV{'QUERY_STRING'});
	foreach $qset (@qsets) {
		@qsetpart = split(/=/, $qset);
		$qstr{$qsetpart[0]} = $qsetpart[1];
	}

	# make a list of the full names for the OSes
	$osNames{'win9x'} = "Windows 95/98";
	$osNames{'winnt'} = "Windows NT 3.5/4.0";
	$osNames{'mac'} = "MacOS 7/8";
	$osNames{'linux'} = "Linux";
	$osNames{'solaris'} = "Solaris";
	$osNames{'freebsd'} = "FreeBSD";
	$osNames{'beos'} = "BeOS";
	$osNames{'handheld'} = "Handheld (PalmOS/WinCE)";
	$osNames{'otherunix'} = "Other Unix-based";

	# get the full name of the OS that was selected and sent as a query string 'os'
	$os = $osNames{$qstr{'os'}};

	# print the page
	print "Content-type: text/html\n\n";
	print "<html><body onLoad=\"parent.results.loadFinish()\">\n";
	print "This text came from a perl script!";
	print "<p>You have chosen:<br>$os\n";
	print "</body></html>\n";
```

To test this script out you could point the browser to the script with a query string manually attached: /cgi-bin/dynduo/cgicomm.pl?os=win9x. It'll cause a JavaScript error when it tries to find the "results" layer, but you can see the script works fine otherwise.

There's only one small change we need to make in the JavaScript to finish everything up. We need to send the "os" variable to to the perl script when you submit the form:

```
if (os) results.load("/cgi-bin/dynduo/cgicomm.pl?os="+os)
```

And Voila! We have DHTML and JavaScript working together with Perl!

As you can see it's really not that difficult. This general idea could be extrapolated significantly to open up a wide range of possibilities. I will be doing some more work on this, I may try to build a DHTML interface to my Forum. And I'll expand on this technique further in the future.

Audio Controls (for Netscape 3,4, and IE 4)

Warning: If you are using IE4 or IE5, and you have installed Microsoft Media Player, the following code won't work. This is because Microsoft was kind enough to break the compatibility of IE's multimedia controls (on purpose perhaps?). I haven't bothered to comb through the latest IE documentation for how they now want people to do audio controls. Also beware, some later versions of Netscape seem to screw up when checking for the LiveConnect plugin - it's there, but for some reason you still get errors. I'll eventually get back to this aspect of JavaScript and clean everything up. Anyone want to help?

Controlling audio is quite simple once you know what the commands are. The easiest way I've found to do it is by using the EMBED tag to load the audio file (wav, au, midi etc.). Then using the appropriate commands you can either play or stop the file.

So first, here's the embed tag (it's pretty self-explanitory):

```
<EMBED NAME="myaudio" SRC="myaudio.mid" LOOP=FALSE AUTOSTART=FALSE
HIDDEN=TRUE MASTERSOUND>
```

The code to make Netscape 3 or Netscape 4 play this file is:

```
document.myaudio.play()
```

If you want the file to loop, you have to use .play(true)

And the code for Internet Explorer 4 is very similar:

```
document.myaudio.run()
```

In IE the file will loop depending on the LOOP property in the EMBED tag.

Finally, the code to stop playing the file is the same across all the browsers:

```
document.myaudio.stop()
```

Audio Error Checking

There's a whole slew of things to consider when implementing audio into your page. Not all versions of Netscape 3 and 4 have audio capabilities, and they'll give an error when it reads the EMBED tag. And if the files haven't loaded yet and you try to play them, they'll give some other error. And if you use IE 4 and try to start a file while another one is playing or if you execute an audio command from an HREF tag it won't do anything,... blah, blah, blah.

Generated Layers

Generating layers is an easy concept to understand and it has a lot of applications, especially when developing an entire websites with Dynamic HTML, or getting into more complex DHTML. Using document.write() commands you can generate your CSS and DIV tags according to whatever specifications you want.

A lot of possibilities are opened when you take this approach:

- You could make a layer appear at a random location
- generate possibly hundreds of layers in an ordered fashion
- center layers, or align them according to the browser window dimensions
- make widget objects that generate CSS and DIV's on their own

I'll show how you can accomplish each of these tasks by following a few simple guidelines.

The Basics

To generate a layer is very straight-forward. Just use the document.write() command to script the CSS and DIV's. The only trick is that you have to document.write the <STYLE> tag along with the CSS. If you don't, things tend not to render properly in Netscape. I've found the following set-up to be the most problem-free:

```
var str = '<STYLE TYPE="text/css">\n'+
'#mylayerDiv {position:absolute; left:50; top:70; width:80; height:20;
clip:rect(0,80,20,0); background-color:yellow; layer-background-color:yellow;}\n'+
'</STYLE>'
 document.write(str)
```

Usually there's no problems in IE, but I've found a few problems with Netscape that you'll want to avoid to save yourself a lot of headache.

- write the CSS in a script in either the head of the document, or immediately following the BODY tag.

- avoid writing a STYLE tag inside another layer, this only works if the layer is going to be relatively positioned, I won't bother covering this but feel free to experiment.
- It's a lot cleaner and more efficient to write all of your CSS at the same time by creating a string of the text.
- ABSOLUTELY DO NOT put a corresponding \n at the end of </STYLE>. An early Netscape (4.0-4.05) bug associated when you do this had me baffled for months, it took me forever to figure it out. I don't know why but if you stick a \n at the end of a STYLE tag and document.write() it within a page with lots of text, you get a line break somewhere in the middle of the page. If you're a regular reader of this website you may have noticed this.

Stick to those guidelines and you'll be okay.

Often its not necessary to SCRIPT the writing of the DIV's. You'll only need to do this if you're planning on making a widget of some sort, or write many DIV's that are alike in some manner. It works as expected:

```
<SCRIPT LANGUAGE="JavaScript">
str = '<DIV ID="mylayerDiv">my layer</DIV>'
document.write(str)
</SCRIPT>
```

Always keep the DIV's in the BODY of the document.

So a basic template to follow looks like this:

```
<HTML>
<HEAD>
<TITLE>The Dynamic Duo - Generated Layers Demo [Simple]</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
 var str = '<STYLE TYPE="text/css">\n'+
 '#mylayerDiv {position:absolute; left:50; top:70; width:100; height:20;
clip:rect(0,100,20,0); background-color:yellow; layer-background-color:yellow;}\n'+
 '</STYLE>'
 document.write(str)
  //-->
 </SCRIPT>
 </HEAD>

 <BODY BGCOLOR="#FFFFFF">

 <SCRIPT LANGUAGE="JavaScript">
  <!--

 var str = '<DIV ID="mylayerDiv">my layer</DIV>'
 document.write(str)

  //-->
 </SCRIPT>

 </BODY>
```

```
</HTML>
```

The css() Function

Something I've been doing to make it nicer to generate CSS is to use a central function which returns the CSS syntax for you. This way you avoid having to rewrite the left, top, width, height etc. for each layer. This makes your code cleaner and will save some file size if you're planning on doing a lot of this. In fact, I've found using the following 2 functions so nice that I tend not to manually write the CSS anymore at all. Because of their immense usefulness, I have included these in my DynLayer as well, so you only need to include the css functions if you are not using the DynLayer:

CSS Functions Source Code:

css.js

The syntax for calling the css() function is:

```
css(id,left,top,width,height,color,vis,z,other)
```

The css() function should be pretty self-explanitory, except for the following...

The most notable is how I worked with the height and clip values. 99% of the time you want to set the height of your layer, you also want to clip the layer to that same value. For example, if you want to make a colored square, you'd have the width and the height the same as the clip right and clip bottom values. On the other hand, if you are just placing some text or an image you don't need to clip it and you don't have to set the height either. So what I've done with the CSS function is when you set the height, it also sets the clip values to (0,width,height,0) - which is the most common situation.

However, in the cases where you want the clip values to be different than the width and height, you may use the other property and send your own 'clip:rect()' CSS. When you do this it will write your clip CSS instead of making it's own based on the width and height.

You can also make the layer positioned relatively by sending null for both the left and top values. In fact any of the values you don't want, just send a null value for and it won't write them. And by sending an ID of "START" or "END" it writes the appropriate STYLE tag to start or end the CSS syntax.

Examples:

```
// return "#mylayer {position:absolute; left:50px; top:100px;}"
css('mylayer',50,100)

// return "#mylayer {position:relative; width:200px;
background-color:#ff0000; layer-background-color:#ff0000;}"
css('mylayer',null,null,200,null,'#ff0000')
```

```
// return "#mylayer {position:absolute; left:50px; top:100px;
width:200px; height:200px; clip:rect(0px 200px 200px 0px);}"
css('mylayer',50,100,200,200)
```

There are 2 options in this function:

```
css('START') // returns "<style type="text/css">"
css('END') // returns "</style>"
```

Here's an example of how to use the css() function to write a layer:

```
var str = css('START')+
css('mylayerDiv',50,100)+
css('END')
document.write(str)
```

You just set up a string containing all the CSS needed, and then document write it to the browser. It is recommended that you only write one set of CSS. If you try to do 2 of the above writing it will hang Netscape 4.0 and 4.01. If it's absolutely necessary you can separate each css writing into separate <script> tags.

The writeCSS() Function

```
writeCSS(str,showAlert)
```

The writeCSS() function (also included in css.js) just makes this a little less of a hassle. The str parameter is the string of CSS that you want to write to the page. writeCSS() will automatically add the css('START') and css('END') values to the front and end of the string and write the resultant string to the page:

```
writeCSS ( css('mylayerDiv',50,100) )
```

writeCSS() also has the showAlert option to display an alert dialog of the CSS string being written to the page. This option is only for debugging purposes:

```
writeCSS (
css('mylayer1Div',50,100)+ // must add  css() calls together
css('mylayer2Div',50,100)+
css('mylayer3Div',50,100) ,1) // send true (or 1) to display a dialog
```

So the combination of these 2 functions are really great if plan on doing this stuff a lot. I will be using these functions quite a bit. Here's that last layer template based on the CSS function:

```
<html>
<head>
<title>The Dynamic Duo - Generated Layers Demo [CSS Function]</title>
<script language="JavaScript" src="../dynapi/css.js"></script>
<script language="JavaScript">
<!--
```

```
writeCSS( css('mylayerDiv',50,70,100,20,'yellow') )
//-->
</script>
</head>

<body bgcolor="#FFFFFF">

<div id="mylayerDiv">my layer</div>

</body>
</html>
```

Now that we have a good way to go about generating layers, we can start getting to the whole reason for doing this. By generating layers in this manner we have a great way to substitute static numbers in your CSS for variables and begin getting into the real meat of dynamically generated pages using DHTML.

Example:

```
var x = 20+15/5 var y = 100+50/5

writeCSS( css('mylayer',x,y) )
```

This tactic will be used extensively in the upcoming lessons.

Generating Multiple Layers:

By doing loops you can use this technique to generate any number of layers in any way you want. You could generate dozens of layers in random positions, or create grids of layers.

Using Browser Width/Height

Even with the advent of the screen object, this can't be used reliably to determine the actual size of the browser window. It is important to know the exact width and height (to the pixel) of the browser to give us the ability to generate layers based on these values. We can use those value to generate layers that stretch to the width of the browser, center layers, or right align them etc., thus giving layers extra flexibility even though they are absolutely positioned.

The best way that I know to find the width/height of the browser is by checking the following properties after the BODY tag. You must place whatever code that is dependent on the width/height in SCRIPT located after the body because in IE the body element is used:

In Netscape:

```
window.innerWidth window.innerHeight
```

In IE:

document.body.scrollWidth document.body.scrollHeight

However, these values don't take into consideration the scrollbar. Usually you'll only be concerned about the vertical scrollbar, so you can manually account for it by subtracting 20 from the width in IE, and 16 in Netscape (Netscape excludes the chrome window border). The following is the template I use:

```
<BODY>

<SCRIPT LANGUAGE="JavaScript">

ns4 = (document.layers)? true:false
ie4 = (document.all)? true:false

winW = (ns4)? window.innerWidth-16 : document.body.offsetWidth-20
winH = (ns4)? window.innerHeight : document.body.offsetHeight

// write out the layers accordingly using the CSS function....
writeCSS(css('mylayer',0,0,winW,winH,'black') // one big black square )

</SCRIPT>

<!-- other HTML elements go here -->

<div id="mylayer"></div>

</BODY>
```

Note: this is the only situation where you should ever have to write CSS within the body.

At any time after the winW and winH variables have been defined can they be used (eg. init() function can use them)

Centering Layers

To use these values to center a layer you can do a little math to find where the left/top co-ordinate should be. For example if your layer is 100px wide and 50px tall, you'll need to use the (winW-100)/2 for the left coordinate, and (winH-50)/2 for the top coordinate.

And that translated into code is:

```
writeCSS ( css('centerDiv',(winW-100)/2,(winH-50)/2,100,50,'blue') )
```

You can do similar statements to align a layer to the right or bottom of the screen. The following example places layers in all four corners and the center of the screen.

Netscape Resize Fix Function

```
// Netscape Resize Fix
```

```
	if (document.layers) {
		widthCheck = window.innerWidth
		heightCheck = window.innerHeight
		window.onResize = resizeFix
	}
	function resizeFix() {
		if (widthCheck != window.innerWidth || heightCheck !=
window.innerHeight)
		document.location.href = document.location.href
	}
```

This piece of code can be inserted into pages which suffer the common problem when you resize the Netscape browser, all your layers loose their positioning. That code will reload the page, again using the browser width and height to check if the size has changed.

Liquid Layout Effect

(a derivative of the resize fix function)

A common practice amongst Table-lovers is to use 100% widths everywhere to make the page stretch to fill with width of the browser. Well, us DHTML-lovers can do that too. Just use the winW and winH variables as described above, and add the Liquid Effect JavaScript file to your page (along with the CSS Function):

```
<script language="JavaScript" src="../dynapi/css.js"></script>
<script language="JavaScript" src="../dynapi/liquid.js"></script>
```

That file contains the findWH() function which you may use to find the winW and winH properties. However those properties have been tweaked so that when you draw layers they will fit *exactly* to the width and height of the window - not including the scrollbars or anything.

The real trick with making the page liquid is that the page must reload in order for all the layers to be redrawn (to keep the layers stretched). What you do is add the onResize event to the body tag:

```
<body bgcolor="#FFFFFF" onResize="makeLiquid()">
```

And presto, your page will reload when resized, and therefore redraw all the layers to their desired positions and dimensions.

Cookie Functions

Cookies are a way to store a small piece of information in a visitors browser. Some people get paranoid about them because they feel the web authors are tracking them, but usually they are just used to gather simple information about their visitors. Information such as:

how many times you've visited the site how you navigate the site (to

determine if it's navigation is good)
who you are and to give you a specific page tailored to you
your username and password to by-pass a login screen

Using cookies are not specific to DHTML or JavaScript, almost any programming language for the web is capable to producing them such as Java, Perl, ASP, C++ etc. In server-side languages this is done by printing a line of text before the HTML content is shown. An advantage that JavaScript has is that it can give you a cookie at any time you are viewing a page, you don't have to go to another page in order to give a cookie.

In the DynAPI there is a cookies.js file which you can use to easily save, read, and delete cookies. Include the file and you'll have 3 functions:

saveCookie(name,value,days) The cookie's name is a variable of your choice, it will be how you'll reference the cookie value. The value is the piece of information that you want to store. It can be a String or Number but will actually be stored as a String in either case. The days is how long until the cookie will be stored before expiring. If you only want to store the cookie for the current browser session, then 0 should be the number of days.

```
saveCookie("favourite cookie","chocolate chip",360) // save for 1 year
```

readCookie(name) This will return the value of the cookie as a String. If the cookie does not exist, it will return null. You'd usually read a cookie like this:

```
var favcookie = readCookie('favourite cookie')

if (mycookie==null) { // cookie does not exist

} else { // cookie exists

}
```

If the value is a number you'll have to use parseInt() to make it an integer.

deleteCookie(name) This will remove the cookie. Actually all it does is re-save the cookie with a days value of -1, which means the cookie expired yesterday.

```
deleteCookie('favourite cookie')
```

The example below will count the number of times that you've read the page. When you reload the number will go up. I read the cookie value, and write a sentence depending on the value of the cookie, then I increment the counter using parseInt() to make sure it's a number, and then re-save the cookie:

```
var count = readCookie('pagecount') // read the 'pagecount' cookie
```

```
if (count==null) { // if no cookie
        document.write("<li>never visiting this page before")
        count = 0 // set counter to 0
}
else { // if cookie exists
        document.write("<li>visted this page "+count+" times before")
}

count = parseInt(count)+1 // increment the counter

saveCookie('pagecount',count,360) // save 'pagecount' cookie for 360
days
```