

Microsoft Dynamic HTML

When comparing the implementation of Dynamic HTML in Navigator 4 and Internet Explorer 4, you get the impression that Internet Explorer 4's DHTML is more flexible and better integrated into the whole of authoring for the browser. Some of this advantage comes from being a more recent product in the game of leapfrog that Microsoft and Netscape play with browser and technology releases. Internet Explorer 4's style sheets use the Cascading Style Sheets (CSS) recommendation as a basis for its extensions. The same is true for the CSS-Positioning (CSS-P) capabilities in the browser.

Authors, however, pay a price for the flexibility built into Internet Explorer 4. The extensive object model and its large complement of properties, methods, and event handlers creates an enormous vocabulary to get to know and work with. Experienced scripters may not have a problem with this, because knowing where to look for information is almost second nature. But newcomers, or those upgrading their skills from HTML to DHTML and scripting, have much to become acquainted with before feeling comfortable in the environment.

Internet Explorer 4 Document Object Model

At the core of Internet Explorer's DHTML and scripting functionality is an impressive document object model. In addition to the typical object containment hierarchy in use since the earliest days of JavaScript, Internet Explorer 4 also turns every document element that can be contained within a tag pair into a scriptable object. This includes style tags (such as ` . . . `) as well as the more obvious block tags (such as `<P> . . . </P>`). Each of these objects has a set of properties, methods, and event handlers.

43

CHAPTER



In This Chapter

The Internet Explorer document object model

Working with the style object for dynamic content and styles

The map game tailored for Internet Explorer 4



A typical object

To give you an idea of the kinds of properties, methods, and event handlers assigned to a tag as simple as the bold tag, Table 43-1 lists the scriptable support for the `` tag.

Table 43-1
Internet Explorer 4 B Object (``)
Properties, Methods, and Events

<i>Property</i>	<i>Description</i>
<code>className</code>	Value assigned to a <code>CLASS</code> attribute of the tag
<code>document</code>	Reference to the containing document
<code>id</code>	Value assigned to an <code>ID</code> attribute of the tag
<code>innerHTML</code>	Text of HTML between the start and end tags
<code>innerText</code>	Text between the start and end tags
<code>isTextEdit</code>	Boolean flag whether the object can be used to create a text range
<code>lang</code>	ISO language to use in rendering tag content
<code>language</code>	Scripting language of the current script
<code>offsetHeight</code>	Element height within the parent coordinate system
<code>offsetLeft</code>	Element left position within the parent coordinate system
<code>offsetParent</code>	Reference to the container element
<code>offsetTop</code>	Element top position within the parent coordinate system
<code>offsetWidth</code>	Element width within the parent coordinate system
<code>outerHTML</code>	Text of HTML including tags
<code>outerText</code>	Text between tags
<code>parentElement</code>	Reference to parent element
<code>parentTextEdit</code>	Reference to next higher element that can be used to create a text range
<code>sourceIndex</code>	Index of item within the <code>all</code> collection (array) of document elements
<code>style</code>	Reference to the style object associated with the element
<code>tagName</code>	Name of the tag (without angle brackets)
<code>title</code>	Text of tooltip assigned to the <code>TITLE</code> attribute of the tag

<i>Method</i>	<i>Description</i>
<code>click()</code>	Scripted click of the element
<code>contains()</code>	Whether the current element is in the parent hierarchy above another element
<code>getAttribute()</code>	Value of a particular tag attribute
<code>insertAdjacentHTML()</code>	Insert text and/or HTML into the element
<code>insertAdjacentText()</code>	Insert text into the element
<code>removeAttribute()</code>	Delete an attribute and value from the tag
<code>scrollIntoView()</code>	Scroll the page to bring the element into view
<code>setAttribute()</code>	Set a tag attribute's value
<i>Event Handler</i>	<i>Description</i>
<code>onclick</code>	Mouse click
<code>ondblclick</code>	Mouse double-click
<code>ondragstart</code>	Beginning of dragging an object or selection
<code>onfilterchange</code>	Change of filter or end of a transition
<code>onhelp</code>	Press of F1 key
<code>onkeydown</code>	Key down
<code>onkeypress</code>	Complete key press
<code>onkeyup</code>	Key up
<code>onmousedown</code>	Mouse down
<code>onmousemove</code>	Moving the mouse
<code>onmouseout</code>	Moving the cursor out of the element
<code>onmouseover</code>	Moving the cursor into the element
<code>onmouseup</code>	Mouse up
<code>onselectStart</code>	Beginning to select text of the element

The properties and methods that intrigue me the most are the ones that let scripts modify the content of a tag — or the tag itself — after the document has loaded. The rendering engine in Internet Explorer 4 automatically and quickly reflows a page's content in response to a script-driven change in a tag's content. For many scripters, this represents a breakthrough from the design shackles of having to put all script-modifiable text inside text and textarea input objects. That's what I call truly *dynamic* HTML.

The text range

To facilitate working with modifiable content, Internet Explorer 4 defines an object called `textRange`. The full description of its inner workings is beyond the scope of this book, but I mention it here as a way to fill what may be a gap in understanding how portions of text can be manipulated inside a document or tag.

A text range is a temporary object that defines the start and end point of a chunk of text inside a document. I call it a temporary object because, after you create a text range in a script, it lives only as long as the script function (and any function it calls) is running. Once the function completes its task, the text range goes away.

The `textRange` object contains many methods that enable a script to move the start and end points of a selection within a text range, find text, remove a chunk of text, and insert text at a specific point in the document. As one simple example of how to use the object, the following code fragment performs a search and replace throughout an entire document.

```
// define the whole body as a text range
var range = document.body.createTextRange()
// loop through range as often as it finds the match word
// and set start/end points to found text
for (var i = 0; range.findText("today") != -1; i++) {
    // scroll document to bring matching word into view
    range.scrollIntoView()
    // select it for visual feedback
    range.select()
    // change the found text
    range.text = "tomorrow"
}
```

The style object

Almost every object in the Internet Explorer 4 object model contains a style object. This is the object that manages the properties for DHTML appearance and positioning settings. Initial values for the object property are established in the `<STYLE>` tag set, in a `STYLE` attribute of a tag, or are imported from an external style sheet specification file. The list of properties of the style object is based on the CSS1 recommendation, plus many extensions that are available only in Internet Explorer 4. When a property name contains a hyphen, the scripted property name is generally converted to an interCap format (identifiers in JavaScript cannot contain hyphens).

The range of properties for the style object is quite extensive. Twenty properties alone influence borders around blocks. Later in this chapter you will see a list of properties of the style object.

Referencing objects—the all collection

Internet Explorer 4 objects that are defined in the document by HTML tags become what Internet Explorer 4 calls *element* objects. The main document object has a property that exposes every element object in the document, even when

objects are deeply nested within each other. The document property is the `all` collection (*collection* is a term Internet Explorer 4 uses to represent an array of document objects). A reference that begins with

```
document.all
```

can “see” every element object in the document. Therefore, if you have a tag whose ID attribute is “fred”, then you can create a reference to that object with

```
document.all.fred
```

From that point, you can access the style object associated with the object

```
document.all.fred.style.propertyName
```

or any of the element’s own properties (such as the properties shown previously in Table 43-1).

You can also use the `all` collection to reach all instances of the same tag in a document. The `tags` property lets you specify a particular tag type, from which your script can retrieve an array (collection) of objects with that tag, as shown in the following excerpt:

```
var allBs = document.all.tags("B")
for (var i = 0; i < allBs.length; i++) {
    allBs[i].style.color = "red"
}
```

After the preceding script runs, all contents set to bold via the `` tag will have its text color appear in red.



Caution

Do not confuse the `all` and `tags` keywords in Internet Explorer’s DHTML vocabulary with the same words in the Navigator DHTML vocabulary. The keywords are used entirely differently in the two environments.

Style Object Properties

This section presents all of the properties of the Internet Explorer 4 style object. The list is long, so I have divided the properties into logical groups (Tables 43-2 through 43-6), based as much as possible on the functionality provided by the property or area of interest. One column of the table also indicates which items are reflected in the Cascading Style Sheets recommendations of the W3C. All other items are unique to Internet Explorer.

If you examine Microsoft’s documentation for these properties, you see frequent references to properties being related to Cascading Style Sheets. This is certainly true as far as Microsoft’s extended implementation of CSS goes, but not all of these properties are part of the W3C CSS1 recommendation.

Table 43-2
Internet Explorer 4 Style Object
Background and Color Properties

<i>Property</i>	<i>CSS1</i>	<i>Description</i>
background	✓	Shortcut to numerous background properties
backgroundAttachment	✓	Whether the background should scroll or be fixed
backgroundColor	✓	Background color
backgroundImage	✓	URL of image
backgroundPosition	✓	Top-left position of background image
backgroundPositionX		Left position of background image
backgroundPositionY		Top position of background image
backgroundRepeat	✓	Whether the background image should repeat and how often
color	✓	Text (foreground) color

Table 43-3
Internet Explorer 4 Style Object Box Properties

<i>Property</i>	<i>CSS1</i>	<i>Description</i>
border	✓	Shortcut to border width, style, and color properties
borderBottom	✓	Shortcut to bottom border width, style, and color properties
borderBottomColor		Color of bottom border
borderBottomStyle		Style of bottom border
borderBottomWidth	✓	Width of bottom border
borderColor	✓	Color for all border edges of a box
borderLeft	✓	Shortcut to left border width, style, and color properties
borderLeftColor		Color of left border
borderLeftStyle		Style of left border
borderLeftWidth	✓	Width of left border
borderRight	✓	Shortcut to right border width, style, and color properties
borderRightColor		Color of right border

<i>Property</i>	<i>CSS1</i>	<i>Description</i>
<code>borderRightStyle</code>		Style of right border
<code>borderRightWidth</code>	✓	Width of right border
<code>borderStyle</code>	✓	Style of all border edges of a box
<code>borderTop</code>	✓	Shortcut to top border width, style, and color properties
<code>borderTopColor</code>		Color of top border
<code>borderTopStyle</code>		Style of top border
<code>borderTopWidth</code>	✓	Width of top border
<code>borderWidth</code>	✓	Width of all border edges of a box
<code>clear</code>	✓	Side(s) on which floating elements cannot appear
<code>height</code>	✓	Element height (with units)
<code>paddingBottom</code>	✓	Bottom padding
<code>paddingLeft</code>	✓	Left padding
<code>paddingRight</code>	✓	Right padding
<code>paddingTop</code>	✓	Top padding
<code>styleFloat</code>	✓	How text wraps around an element
<code>width</code>	✓	Element width measure (with units)

Table 43-4
Internet Explorer 4 Style Font and Text Properties

<i>Property</i>	<i>CSS1</i>	<i>Description</i>
<code>font</code>	✓	Shortcut to font properties in one statement
<code>fontFamily</code>	✓	Font family name
<code>fontSize</code>	✓	Font size
<code>fontStyle</code>	✓	Font style
<code>fontVariant</code>	✓	Font variant style
<code>fontWeight</code>	✓	Font weight
<code>letterSpacing</code>	✓	Text letter spacing
<code>lineHeight</code>	✓	Text line height
<code>margin</code>	✓	Shortcut to four margin settings

(continued)

Table 43-4 (*continued*)

<i>Property</i>	<i>CSS1</i>	<i>Description</i>
marginBottom	✓	Bottom margin
marginLeft	✓	Left margin
marginRight	✓	Right margin
marginTop	✓	Top margin
pageBreakAfter		Break page after the element when printing
pageBreakBefore		Break page before the element when printing
pixelHeight		Integer height measure
pixelWidth		Integer width measure
textAlign	✓	Text alignment in the element
textDecoration		Text decoration setting
textDecorationBlink		Blinking text (not supported)
textDecorationLineThrough		Strikethrough text
textDecorationNone		Remove decoration
textDecorationOverline		Overline decoration
textDecorationUnderline		Underline decoration
textIndent	✓	Size of indentation of the first line of a block
textTransform	✓	Initial capital, uppercase, lowercase, none
verticalAlign	✓	Vertical positioning relative to the parent

Table 43-5
Internet Explorer 4 Classification and Housekeeping Properties

<i>Property</i>	<i>CSS1</i>	<i>Description</i>
cssText		String representation of the CSS rule
cursor		Cursor while pointer is atop an element (for example, "hand," "wait")
display	✓	How or if an element should be rendered
filter		Name of a filter effect

<i>Property</i>	<i>CSS1</i>	<i>Description</i>
listStyle	✓	Shortcut to setting list style properties
listStyleImage	✓	List item image URL
listStylePosition	✓	List item indent/outdent style
listStyleType	✓	List item enumeration character type

Table 43-6
Internet Explorer 4 Style Positioning Properties

<i>Property</i>	<i>CSS-P</i>	<i>Description</i>
clip	✓	Rectangle of viewing region of a style
left	✓	Element left position (with units)
overflow	✓	How to handle overflow content
pixelLeft		Integer left coordinate
pixelTop		Integer top coordinate
posHeight		Floating-point height measure
position	✓	Absolute or relative positioning
posLeft		Floating-point left measure
posTop		Floating-point top measure
posWidth		Floating-point width measure
top	✓	Element top coordinate (with units)
visibility	✓	Whether item can be seen
zIndex	✓	Z-order of element among peers

Dynamic Positioning

Internet Explorer blends the CSS1 and CSS-P recommendations into the document object model and especially the style object. All positioning properties for elements are reflected through the style object, as shown in Table 43-6.

To demonstrate dynamic positioning in action in Internet Explorer, I present the third and final version of the map puzzle game that is implemented earlier in this book in a cross-platform version (Chapter 41) and Navigator-only version (Chapter 42). In this chapter's version, the implementation uses Internet Explorer 4's Cascading Style Sheets for positioning and document object model for scripting the elements. The version shown here will not run successfully in Navigator 4.

Navigator puzzle game overview

The Explorer-only version of the game consists of one HTML file and numerous image files. The document loads to reveal the play area (Figure 43-1). The HTML for the normally hidden panel is integrated into the main document, but as a distinct block-level element. This element specifies a couple of styles for some of the nested elements to help format the content as desired. Image files for this and the other two versions of the game are completely identical.

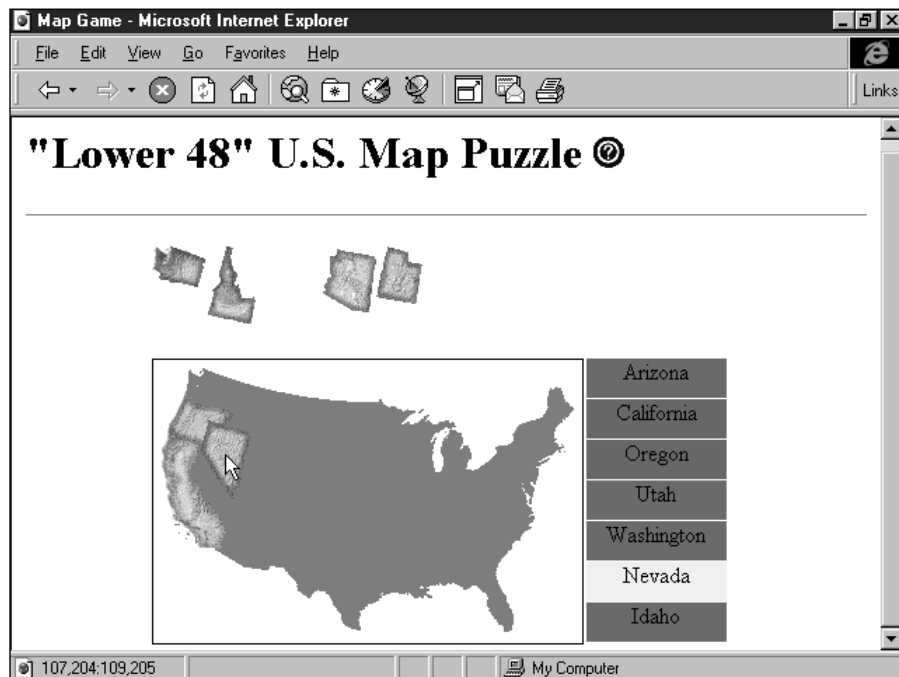


Figure 43-1: The Internet Explorer-only version of the puzzle game (Image courtesy Cartesia Software — www.map-art.com)

Structure of the HTML document is straightforward. A large `<SCRIPT>` tag segment in the Head portion contains all the scripting for the document (including the dynamic help panel). Document-level event capture is set in the `<BODY>` tag for the main document to control the picking up, dragging, and release of state maps.

The document

Listing 43-1 contains the entire source code for the document. In most cases, the scripting code is identical to the Internet Explorer portion of scripts in the cross-platform version. No external DHTML API file is needed because the scripts can take direct advantage of Explorer's own vocabulary for style properties. Another small difference is that you have no need for a link surrounding the help image in this version. Internet Explorer 4's document object model defines numerous event handlers for the image object directly.

Positionable elements are defined as CSS-P items, with the style for each element defined in a <STYLE> tag at the beginning of the document. <DIV> tags in the HTML associate the styles with content blocks. The block for the help panel includes CSS1 styles for a couple of the tags to assist in text formatting.

For in-depth descriptions of the functions in the script and the structure of the positionable objects, see the commentary associated with Listing 41-2.

Listing 43-1: The Internet Explorer Puzzle Document (IEmapgm.htm)

```
<HTML>
<HEAD><TITLE>Map Game</TITLE>
<STYLE TYPE="text/css">

    #help {position:absolute; visibility:hidden; top:80; width:300;
background-color:"#98FB98";}
    #bgmap {position:absolute; left:100; top:180; width:1;}

    #azlabel {position:absolute; left:310; top:0; background-
color:red; width:100; height:28; text-align:center;}
    #calabel {position:absolute; left:310; top:29; background-
color:red; width:100; height:28; text-align:center;}
    #orlabel {position:absolute; left:310; top:58; background-
color:red; width:100; height:28; text-align:center;}
    #utlabel {position:absolute; left:310; top:87; background-
color:red; width:100; height:28; text-align:center;}
    #walabel {position:absolute; left:310; top:116; background-
color:red; width:100; height:28; text-align:center;}
    #nvlabel {position:absolute; left:310; top:145; background-
color:red; width:100; height:28; text-align:center;}
    #idlabel {position:absolute; left:310; top:174; background-
color:red; width:100; height:28; text-align:center;}

    #camap {position:absolute; left:20; top:100; width:1;}
    #ormap {position:absolute; left:60; top:100; width:1;}
    #wamap {position:absolute; left:100; top:100; width:1;}
    #idmap {position:absolute; left:140; top:100; width:1;}
    #nvmap {position:absolute; left:180; top:100; width:1;}
    #azmap {position:absolute; left:220; top:100; width:1;}
    #utmap {position:absolute; left:260; top:100; width:1;}

    #congrats {position:absolute; visibility:hidden; left:20;
top:100; width:1;}

</STYLE>

<SCRIPT LANGUAGE="JavaScript">
//var engaged = false
var offsetX = 0
var offsetY = 0
```

(continued)

Listing 43-1 (*continued*)

```

var selectedObj
var selectedState = ""
var selectedStateIndex

function state(abbrev, fullName, x, y) {
    this.abbrev = abbrev
    this.fullName = fullName
    this.x = x
    this.y = y
    this.done = false
}

var states = new Array()
states[0] = new state("ca", "California", 107, 234)
states[1] = new state("or", "Oregon", 107, 204)
states[2] = new state("wa", "Washington", 123, 188)
states[3] = new state("id", "Idaho", 148, 197)
states[4] = new state("az", "Arizona", 145, 285)
states[5] = new state("nv", "Nevada", 127, 241)
states[6] = new state("ut", "Utah", 155, 249)

function showProps(obj, objName) {
    var result = ""
    count = 0
    for (var i in obj) {
        result += objName + "." + i + "=" + obj[i] + "\n"
        count++
        if (count == 25) {
            alert(result)
            result = ""
            count = 0
        }
    }
    alert(result)
}

function getSelectedMap() {
    selectedObj = (window.event.srcElement).parentElement
    if (selectedObj) {
        var stateName = selectedObj.id.substring(0,2)
        selectedObj = selectedObj.style
        for (var i = 0; i < states.length; i++) {
            if (states[i].abbrev == stateName ) {
                selectedStateLabel = document.all(stateName +
"label")

                selectedStateIndex = i
                selectedObj.zIndex = 100
                return
            }
        }
        selectedObj = null
        selectedStateLabel = null
        selectedStateIndex = null
    }
}

```

```

        return
    }
    function dragIt() {
        if (selectedObj) {
            selectedObj.pixelLeft = (window.event.clientX - offsetX)
            selectedObj.pixelTop = (window.event.clientY - offsetY)
        }
    }
    function toggleEngage() {
        if (selectedObj) {
            release()
        } else {
            engage()
        }
    }
    function engage() {
        getSelectedMap()
        if (selectedObj) {
            offsetX = window.event.offsetX - document.body.scrollLeft
            offsetY = window.event.offsetY - document.body.scrollTop
            selectedStateLabel.style.backgroundColor = "yellow"
        }
    }
    function release() {
        if (selectedObj) {
            if (onTarget()) {
                selectedStateLabel.style.backgroundColor = "green"
                states[selectedStateIndex].done = true
                if (isDone()) {
                    document.all.congrats.style.visibility = "visible"
                }
            } else {
                selectedStateLabel.style.backgroundColor = "red"
                states[selectedStateIndex].done = false
                document.all.congrats.style.visibility = "hidden"
            }
            selectedObj.zIndex = 0
            selectedObj = null
            selectedState = ""
        }
    }
    function onTarget() {
        var x = states[selectedStateIndex].x
        var y = states[selectedStateIndex].y
        var objX = selectedObj.pixelLeft
        var objY = selectedObj.pixelTop
        if ((objX >= x-2 && objX <= x+2) && (objY >= y-2 && objY <=
y+2)) {
            selectedObj.pixelLeft = x
            selectedObj.pixelTop = y
            return true
        }
        return false
    }

```

(continued)

Listing 43-1 (continued)

```

    }
    function isDone() {
        for (var i = 0; i < states.length; i++) {
            if (!states[i].done) {
                return false
            }
        }
        return true
    }
    function showHelp() {
        var help = document.all.help.style
        help.pixelLeft = document.body.scrollWidth
        help.visibility = "visible"
        help.zIndex = 1000
        intervalID = setInterval("moveHelp()",1)
    }
    function moveHelp() {
        var help = document.all.help.style
        help.pixelLeft -= 5
        if (help.pixelLeft <= (document.body.scrollWidth/2) -
(help.pixelWidth/2)) {
            clearInterval(intervalID)
        }
    }
    function hideMe() {
        clearInterval(intervalID)
        document.all.help.style.visibility = "hidden"
        document.all.help.style.pixelLeft = document.body.scrollWidth
    }
</SCRIPT>
</HEAD>
<BODY onMouseDown="toggleEngage()" onMouseMove="dragIt()"
<H1>"Lower 48" U.S. Map Puzzle<IMG onClick="showHelp()"
onMouseOver="status='Show help panel...';return true"
onMouseOut="status='' SRC="info.gif" HEIGHT=22 WIDTH=22 BORDER=0></H1>
<HR>
<DIV ID="bgmap"><IMG SRC="us11.gif" WIDTH=306 HEIGHT=202
BORDER=1>&nbsp;</IMG>
<DIV ID="azlabel">Arizona</DIV>
<DIV ID="calabel">California</DIV>
<DIV ID="orlabel">Oregon</DIV>
<DIV ID="utlabel">Utah</DIV>
<DIV ID="walabel">Washington</DIV>
<DIV ID="nvlabel">Nevada</DIV>
<DIV ID="idlabel">Idaho</DIV>
</DIV>

<DIV ID="camap"><IMG SRC="ca.gif" WIDTH=47 HEIGHT=82 BORDER=0></DIV>
<DIV ID="ormap"><IMG SRC="or.gif" WIDTH=57 HEIGHT=45 BORDER=0></DIV>
<DIV ID="wamap"><IMG SRC="wa.gif" WIDTH=38 HEIGHT=29 BORDER=0></DIV>

```

```

<DIV ID="idmap"><IMG SRC="id.gif" WIDTH=34 HEIGHT=55 BORDER=0></DIV>
<DIV ID="azmap"><IMG SRC="az.gif" WIDTH=38 HEIGHT=45 BORDER=0></DIV>
<DIV ID="nvmap"><IMG SRC="nv.gif" WIDTH=35 HEIGHT=56 BORDER=0></DIV>
<DIV ID="utmap"><IMG SRC="ut.gif" WIDTH=33 HEIGHT=41 BORDER=0></DIV>

<DIV ID="congrats" STYLE="color:red"><H1>Congratulations!</H1></DIV>

<DIV ID="help" onClick="hideMe()">
<P STYLE="margin-top:5"><CENTER><B>Instructions</B></CENTER></P>
<HR COLOR="seagreen">
<OL STYLE="margin-right:20">
<LI>Click on a state map to pick it up. The label color turns yellow.
<LI>Move the mouse and map into position, and click the mouse to drop
the state map.
<LI>If you are close to the actual location, the state snaps into
place and the label color turns green.
</OL>
<FORM>
<CENTER>
<INPUT TYPE="button" VALUE="Close">
</FORM>
</DIV>
</BODY>
</HTML>

```

Lessons learned

The tight integration of dynamic positioning into Internet Explorer 4's CSS implementation provides a fairly logical framework for page design and scripting. Certainly the scriptable access to style properties on virtually any tag is a plus. Perhaps the addition of some higher-level methods for styles (for example, to be able to move an element in one method statement, instead of setting *x* and *y* pixel coordinates individually) would make the scripting more enjoyable, but the current way is not intolerable. With commitments by both Netscape and Microsoft to support standards, I wouldn't be surprised if a future version of Navigator supported the same kind of object and style access that Internet Explorer 4 provides now.

