

The Layer Object

One of the hallmarks of the Navigator 4 and Internet Explorer 4 generations of browsers is Dynamic HTML — the ability to alter content on the fly in response to user interaction. Dynamic HTML (DHTML) has two components that often work together: style sheets and positioning. You can read more about these in Chapters 41 through 43.

Netscape Layers

As often happens when browser providers work faster than standards groups, both Microsoft and Netscape have developed supersets of proposed standards of DHTML functionality and implementation. This chapter focuses on Netscape's contribution to DHTML, the layer object. While it may sound as though the layer object is tied directly to Netscape's new `<LAYER>` tag, the layer object lets you script (primarily for positioning purposes) elements created in a cross-platform, standards-body-approved HTML syntax of Cascading Style Sheets (CSS).

Layer Object

<i>Properties</i>	<i>Methods</i>	<i>Event Handlers</i>
above	load()	onBlur=
background	moveAbove()	onFocus=
below	moveBelow()	onLoad=
bgcolor	moveBy()	onMouseOut=
clip.bottom	moveTo()	onMouseOver=
clip.left	moveToAbsolute()	
clip.right	resizeBy()	
clip.top	resizeTo()	
document		
left		

(continued)

19

CHAPTER



In This Chapter

Relationships between document and layer objects

Dynamically changing content in Navigator

How to move, hide, and show content in Navigator



<i>Properties</i>	<i>Methods</i>	<i>Event Handlers</i>
name		
pageX		
pageY		
parentLayer		
siblingAbove		
siblingBelow		
src		
top		
visibility		
zIndex		

Syntax

Creating a layer:

```

<LAYER | ILAYER
  ID="LayerName"
  SRC="DocumentURL"
  [LEFT="PixelCount"]
  [TOP="PixelCount"]
  [PAGEX="PixelCount"]
  [PAGEY="PixelCount"]
  [ABOVE="OtherLayerName"]
  [BELOW="OtherLayerName"]
  [Z-INDEX="LayerIndex"]
  [HEIGHT="PixelCount" | "PercentageValue%"]
  [WIDTH="PixelCount" | "PercentageValue%"]
  [CLIP="LeftPixel, TopPixel, RightPixel, BottomPixel"]
  [BGCOLOR="HexTriplet" | "ColorName"]
  [BACKGROUND="ImageURL"]
  [VISIBILITY="SHOW" | "HIDDEN" | "INHERIT"]
  [onBlur="handlerTextOrFunction"]
  [onFocus="handlerTextOrFunction"]
  [onLoad="handlerTextOrFunction"]
  [onMouseOut="handlerTextOrFunction"]
  [onMouseOver="handlerTextOrFunction"]>
</LAYER | /ILAYER>
<NOLAYER>
  [Content for no layer support]
</NOLAYER>

layerObject = new Layer(pixelWidth [, parentLayerObject])

<STYLE type="text/css">
  #layerName {position:positionType [,attributeLabel1:value1 ...]}
</STYLE>

```

```
<HTMLTag
  ID="layerName"
  STYLE=" position:positionType [;attributeLabel1:value1 ...]>
  LayerContent
</ HTMLTag >
```

Accessing layer properties or methods:

```
[window.] document.layerName.[document.layerName. ...] property |
method([parameters])
[window.] document.layers[index]. [document.layerName. ...]property |
method([parameters])
```

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

About this object

Perhaps you have seen how animated cartoons were created before computer animation changed the art. Layers of clear acetate sheets were assembled atop a static background. Each sheet contained one character or portion of a character. When all the sheets were carefully positioned atop each other, the view, as captured by a still camera, formed a composite frame of the cartoon. To create the next frame of the cartoon, the artist moved one of the layers a fraction of an inch along its intended path — and then another picture was taken.

If you can visualize how that operation works, you have a good starting point for understanding how layers work in Navigator (from Version 4 onward). In Navigator, each layer contains some kind of HTML content that exists in its own plane above the main document that loads in a window. The content of an individual layer can be changed or replaced on the fly without affecting the other layers; and the entire layer can be repositioned, resized, or hidden under script control.

One aspect of layers that goes beyond the cartoon analogy is that a layer can contain other layers. When that happens, any change that affects the primary layer — such as moving the layer 10 pixels downward — also affects the layers nested inside. It's as if the nested layers are passengers of the outer layer. When the outer layer goes somewhere, the passengers do, too. And yet, within the “vehicle” the passengers may change seats by moving themselves around without regard for what's going on outside.

Layer references

The task of assembling JavaScript references to layers and the objects they contain is very much like doing the same for framesets (in fact, conceptually, a layer is like a dynamically movable and resizable free-floating frame). Therefore, before you start writing the reference, you must know the relationship between the document *containing* the script and the target of the reference.

To demonstrate how this works, I start with a script in the base document loaded into a window that needs to change the background color (`bgColor` property) of a layer defined in the document. The skeletal HTML is as follows:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<LAYER NAME="Flintstones" SRC="flintstonesFamily.html">
</LAYER>
</BODY>
</HTML>
```

From a script in the Head section, the statement that changes the layer's `bgColor` property is

```
document.Flintstones.bgColor = "yellow"
```

This syntax looks like the way you address any object in a document, such as a link or image, but things get tricky in that each layer automatically contains a document object. That document object is what holds the content of the layer. Therefore, if you wanted to inspect the `lastModified` property of the HTML document loaded into the layer, the statement would be

```
var modDate = document.Flintstones.document.lastModified
```

The situation gets more complex if the layer has another layer nested inside it (one of those “passengers” that goes along for the ride). If the structure changes to

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<LAYER NAME="Flintstones" SRC="flintstonesFamily.html">
  <LAYER NAME="Fred" SRC="fredFlintstone.html">
    <LAYER NAME="Wilma" SRC="wilmaFlintstone.html">
  </LAYER>
</BODY>
</HTML>
```

references to items in the second level of frames get even longer. For example, to get the `lastModified` property of the `wilmaFlintstone.html` file loaded into the nested Fred layer, the reference from the Head script would be

```
document.Flintstones.document.Fred.document.lastModified
```

The reason for this is Navigator does not have a shortcut access to every layer defined in a top-level document. As stated in the description of the `document.layers` property in Chapter 16, the property reflects only the first level of layers defined in a document. You must know the way to San Jose if you want to get its `lastModified` property.

Because each layer has its own document, you cannot spread a form across multiple layers. Each layer's document must define its own `<FORM>` tags. If you need to submit one form from content located in multiple layers, one of the forms should have an `onSubmit=` event handler to capture all the related form values

and place them in hidden input fields in the document containing the submitted form. In that case, you need to know how to devise references from a nested layer outward.

As a demonstration of reverse-direction references, I'll start with the following skeletal structure containing multiple nested layers:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<FORM NAME="personal">
  <INPUT TYPE="text" NAME="emailAddr">
</FORM>
<LAYER NAME="product" SRC="ultraGizmoLine.html">
  <LAYER NAME="color" SRC="colorChoice.html">
    <LAYER NAME="size" SRC="sizeChoice.html">
      <LAYER NAME="sendIt" SRC="submission.html"
</LAYER>
</BODY>
</HTML>
```

Each of the HTML files loaded into the layers also has a `<FORM>` tag defining some fields or select lists for relevant user choices, such as which specific model of the UltraGizmo line is selected, what color, and in what size (these last two are defined as separate layers because their positions are animated when they are displayed). The assumption here is that the Submit button is in the `sendIt` layer. That layer's document also includes hidden input fields for data to be pulled from the main document's form and three other layer forms. Two of those layers are at the same nested level as `sendIt`, one is above it, and the main document's form is at the highest level.

To reach the `value` property of a field named `theColor` in the `color` layer, a script in the `sendIt` layer would use the reference

```
parentLayer.document.color.document.forms[0].theColor.value
```

Analogous to working with frames, the reference starts with a reference to the next higher level (`parentLayer`), and then starts working its way down through the parent layer's document, the `color` layer, the `color` layer's document, and finally the form therein.

To reach the `value` property of a field named `modelNum` in the `product` layer, the reference starts the same way, but because the form is at the parent layer level, the reference goes immediately to that layer's document and form:

```
parentLayer.document.forms[0].modelNum.value
```

It may seem odd that a reference to an object at a different layer level is shorter than one at the same level (for example, the `color` layer), but the route to the parent layer is shorter than going via the parent layer to a sibling. Finally, to reach the value of the `emailAddr` field in the base document, the reference must ratchet out one more layer, as follows:

```
parentLayer.parentLayer.document.forms[0].emailAddr.value
```

The two `parentLayer` entries step the reference out two levels, at which point the scope is in the base layer containing the main document and its form.

Cross-platform concerns

Navigator does not rely only on the `<LAYER>` tag or `new Layer()` construction for layer objects. If the document contains Cascading Style Sheet-Positioning (CSS-P) information, as shown in the syntax listing at the start of this section, then Navigator automatically turns each named, positioned item into a layer object. You have the same scripted positioning control over CSS-P-generated layers as over Navigator layer objects.

In the property, method, and event handler listings throughout this chapter, you will see that the syntaxes apply only to Navigator 4. While some Internet Explorer language items are the same, fundamental differences exist in the way positioned objects are references in JScript and JavaScript. Moreover, Internet Explorer 4's object model is much richer in that its positionable objects have more properties that allow such powers as dynamically swapping text without reloading the document in a layer. However, the point of view for this book is from that of Navigator and its implementation of a document object model and JavaScript.

For more details about style sheets and positioning in Navigator and Internet Explorer (separately and in cross-platform scenarios), see Chapters 41 through 43.

Properties

above
below
siblingAbove
siblingBelow

Value: Layer Object **Gettable:** Yes **Settable:** No

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

Each layer exists in its own physical layer. Given that width and height are traditionally represented by the variables `x` and `y`, the third dimension — the position of a layer relative to the stack of layers — is called the *z-order*. Layer orders are assigned automatically according to the loading order, with the highest number being the topmost layer. That topmost layer is the one closest to you as you view the page on the monitor.

If two layers are on a page, one layer must always be in front of the other, even if they both appear to be transparent and visually overlap each other. Knowing which layer is above the other can be important for scripting purposes, especially if your script needs to re-order the layering in response to user action. Layer

objects have four properties to help you determine the layers adjacent to a particular layer.

The first pair of properties, `layer.above` and `layer.below`, takes a global look at all layers defined on the page, regardless of how one layer may contain any number of nested layers separate from other batches on the screen. If a layer lies above the one in question, the property contains a reference to that other layer; if no layer exists in that direction, then the value is null. Attempts to get properties of a nonexistent layer results in runtime scripting errors indicating that the object does have properties (of course not — an object must exist before it can have properties).

To understand these two properties better, consider a document that contains three layers (in any nesting arrangement you like). The first layer to be defined is on the bottom of the stack. It has a layer above it, but none below it. The second layer in the middle has a layer both above and below it. And the topmost layer has a layer only below it, with no more layers above it (that is, coming toward your eye).

Another pair of properties, `layer.siblingAbove` and `layer.siblingBelow`, confine themselves to whatever group of layers are contained by a parent layer. Just as in real family life, siblings are descended from (teens might say “contained by”) the same parent. An only child layer has no siblings, so both the `layer.siblingAbove` and `layer.siblingBelow` values are null. For two layers from the same parent, the first one to be defined has a sibling layer above it; the other has a sibling layer below it.

It is important to understand the difference between absolute layering and sibling layering to use these properties correctly. A nested layer might be the fifth layer from the bottom among all layers on the page, but at the same time is the first layer among siblings within its family group. As you can see, these two sets of properties enable your script to be very specific about the relationships under examination.

Example

Listing 19-1 lets you experiment with just one set of these properties: `layer.above` and `layer.below`. The page is almost in the form of a laboratory and quiz that lets you query yourself about the values of these properties for two swappable layers.

Listing 19-1: A Layer Quiz

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function checkAbove(oneLayer) {
    document.forms[0].errors.value = ""
    document.forms[0].output.value = oneLayer.above.name
}
function checkBelow(oneLayer) {
    document.forms[0].errors.value = ""
    document.forms[0].output.value = oneLayer.below.name
}
function swapLayers() {
```

(continued)

Listing 19-1 (*continued*)

```

        if (document.yeller.above) {
            document.yeller.moveAbove(document.greeny)
        } else {
            document.greeny.moveAbove(document.yeller)
        }
    }
    function onerror(msg) {
        document.forms[0].output.value = ""
        document.forms[0].errors.value = msg
        return true
    }
</SCRIPT>
</HEAD>
<BODY>
<B>Layer Ordering</B>
<HR>
<FORM>
Results:<INPUT TYPE="text" NAME="output"><P>
<INPUT TYPE="button" VALUE="Who's ABOVE the Yellow layer?"
onClick="checkAbove(document.yeller)"><BR>
<INPUT TYPE="button" VALUE="Who's BELOW the Yellow layer?"
onClick="checkBelow(document.yeller)"><P>
<INPUT TYPE="button" VALUE="Who's ABOVE the Green layer?"
onClick="checkAbove(document.greeny)"><BR>
<INPUT TYPE="button" VALUE="Who's BELOW the Green layer?"
onClick="checkBelow(document.greeny)"><P>
<INPUT TYPE="button" VALUE="Swap Layers" onClick="swapLayers()"><P>
If there are any errors caused by missing <BR>
properties, they will appear below:<BR>
<TEXTAREA NAME="errors" COLS=30 ROWS=3 WRAP="virtual"></TEXTAREA>
</FORM>
<LAYER NAME="yeller" BGCOLOR="yellow" TOP=60 LEFT=300 WIDTH=200
HEIGHT=200>
<B>This is just a yellow layer.</B>
</LAYER>
<LAYER NAME="greeny" BGCOLOR="lightgreen" TOP=100 LEFT=340 WIDTH=200
HEIGHT=200>
<B>This is just a green layer.</B>
</LAYER>
</BODY>
</HTML>

```

The page contains two layers, one colored yellow, the other light green. Legends on four buttons ask you to guess whether one layer is above or below the other. For example, if you click the button labeled “Who’s ABOVE the Yellow layer?” and the green layer is above it, the name of that green layer appears in the Results field. But if layers are oriented such that the returned value is null, the error

message (indicating that the nonexistent object doesn't have a name property) appears in the error field at the bottom. Another button lets you swap the order of the layers so you can try your hand at predicting the results based on your knowledge of layers and the `above` and `below` properties.

Related Items: `layer.parentLayer` property; `layer.moveAbove()` method; `layer.moveBelow()` method.

background

Value: Image Object **Gettable:** Yes **Settable:** Yes

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

You can assign a background image to a layer. The initial image is usually set by the `BACKGROUND` attribute of the `<LAYER>` tag, but you can assign a new image whenever you like via the `layer.background` property.

Layer background images are typically like those used for entire Web pages. They tend to be subtle or at least of such a design and color scheme as to not distract from the primary content of the layer. On the other hand, the background image may in fact be the content. If so, then have a blast with whatever images suit you.

The value of the `layer.background` property is an image object (see Chapter 18). To change the image in that property on the fly, you must set the `layer.background.src` property to the URL of the desired image (just like changing a `document.image` on the fly). You can remove the background image by setting the `layer.background.src` property to `null`. Background images smaller than the rectangle of the layer repeat themselves, just like document background pictures; images larger than the rectangle clip themselves to the rectangle of the layer, rather than scaling to fit.

Example

A simple example (Listing 19-2) defines one layer that features five buttons to change the background image of a second layer. I put the buttons in a layer because I want to make sure the buttons and background layer rectangles align themselves along their top edges on all platforms.

As the second layer loads, I merely assign a gray background color to it and write some reverse (white) text. Most of the images are of the small variety that repeat in the layer. One is a large photograph to demonstrate how images are clipped to the rectangle. Along the way, I hope you also heed the lesson of readability demonstrated by the difficulty of reading text on a wild-looking background.

Listing 19-2: Setting Layer Backgrounds

```

<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function setBg(URL) {
    document.bgExpo.background.src = URL
}
</SCRIPT>
</HEAD>
<BODY>
<B>Layer Backgrounds</B>
<HR>
<LAYER NAME="buttons" TOP=50>
    <FORM>
        <INPUT TYPE="button" VALUE="The Usual"
onClick="setBg('cr_kraft.gif')"><BR>
        <INPUT TYPE="button" VALUE="A Big One"
onClick="setBg('arch.gif')"><BR>
        <INPUT TYPE="button" VALUE="Not So Usual"
onClick="setBg('wh86.gif')"><BR>
        <INPUT TYPE="button" VALUE="Decidedly Unusual"
onClick="setBg('sb23.gif')"><BR>
        <INPUT TYPE="button" VALUE="Quick as..."
onClick="setBg('lightnin.gif')"><BR>
    </FORM>
</LAYER>
<LAYER NAME="bgExpo" BGCOLOR="gray" TOP=50 LEFT=250 WIDTH=300
HEIGHT=260>
<B><FONT COLOR="white">Some text, which may or may not read well with
the various backgrounds.</FONT></B>
</LAYER>
</BODY>
</HTML>

```

Related Items: layer.bgColor property; image object.

bgColor

Value: String **Gettable:** Yes **Settable:** Yes

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

A layer's background color fills the entire rectangle with the color set either in the `<LAYER>` tag or from a script at a later time. Color values are the same as for document-related values, and may be in the hexadecimal triplet format or one of

the Netscape plain-language colors. You can turn a layer transparent by setting its `bgColor` property to `null`.

Example

You can have some fun with Listing 19-3, which utilizes a number of layer scripting techniques. The page presents a kind of palette of eight colors, each one created as a small layer (see Figure 19-1). Another, larger layer is the one that has its `bgColor` property changed as you roll the mouse over any color in the palette.

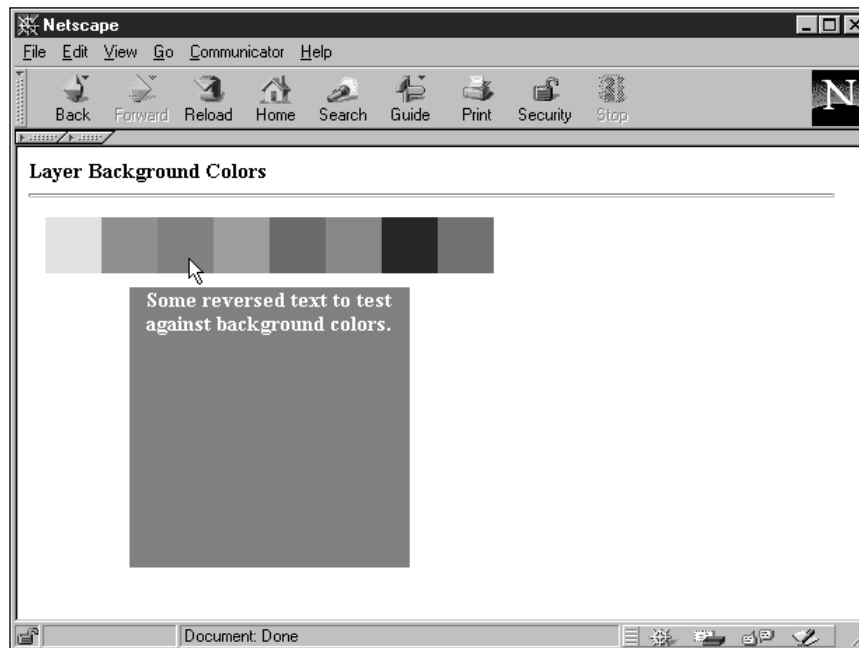


Figure 19-1: Drag the mouse across the palette to change the layer's background color.

To save HTML lines to create those eight color palette layers, I use a script to establish an array of colors, and then `document.write()` the `<LAYER>` tags with appropriate attribute settings so the layers all line up in a contiguous row. By predefining a number of variable values for the size of the color layers, I can make all of them larger or smaller with the change of only a few script characters.

The job of capturing the `mouseover` events is handed to the document object. I turn on the document's `captureEvents()` method such that it traps all `mouseover` events and hands them to the `setColor()` function. The `setColor()` function reads the target object's `bgColor` and sets the larger layer's `bgColor` property to the same. If this page had other objects that can receive `mouseover` events for other purposes, I would use `routeEvents()` to let those events pass on to their intended targets. For the purposes of this example, however, the events need go no further.

Listing 19-3: Layer Background Colors

```

<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function setColor(e) {
    document.display.bgColor = e.target.bgColor
}
document.captureEvents(Event.MOUSEOVER)
document.onmouseover = setColor
</SCRIPT>
</HEAD>
<BODY>
<B>Layer Background Colors</B>
<HR>
<SCRIPT LANGUAGE="JavaScript">
var oneLayer
var top = 50
var left = 20
var width = 40
var height = 40
var colorPalette = new
Array("aquamarine","coral","forestgreen","goldenrod","red","magenta",
"navy","teal")
for (var i = 0; i < colorPalette.length; i++) {
    oneLayer = "<LAYER NAME=swatch" + i + " TOP=" + top + " LEFT="
+ ((width * i) + 20)
    oneLayer += " WIDTH=" + width + " HEIGHT=" + height
    oneLayer += " BGCOLOR=" + colorPalette[i] + "></LAYER>\n"
    document.write(oneLayer)
}
</SCRIPT>
<LAYER NAME="display" BGCOLOR="gray" TOP=100 LEFT=80 WIDTH=200
HEIGHT=200>
<B><FONT COLOR="white"><CENTER>Some reversed text to test against
background colors.</CENTER></FONT></B>
</LAYER>
</BODY>
</HTML>

```

Related Items: `layer.background` property; `layer.onMouseOver`= event handler.

clip

Value: String **Gettable:** Yes **Settable:** Yes

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

The `layer.clip` property is itself an object (the only one in Netscape 4's document object model that exposes itself as a rectangle object) with six geographical properties defining the position and size of a rectangular area of a layer visible to the user. Those six properties are

```
clip.top  
clip.left  
clip.bottom  
clip.right  
clip.width  
clip.height
```

The unit of measure is pixels, and the values are relative to the top-left corner of the layer object.

A clip region can be the same size as or smaller than the layer object. If the `CLIP` attribute is not defined in the `<LAYER>` tag, the clipping region is the same size as the layer. In this case, the `clip.left` and `clip.top` values are automatically zero, because the clip region starts at the very top-left corner of the layer's rectangle (measurement is relative to the layer object whose clip property you're dealing with). The height and width of the layer object are not available properties in Navigator 4. Therefore, you may have to use other means to get that information into your scripts if you need it (I do it in Listing 19-4). Also be aware that even if you set the `HEIGHT` and `WIDTH` attributes of a layer tag, the content rules the initial size of the visible layer unless the tag also includes specific clipping instructions. Images, for example, expand a layer to fit the `HEIGHT` and `WIDTH` attributes of the `` tag; text (either from an external HTML file or inline in the current file) adheres to the `<LAYER>` tag's `WIDTH` attribute, but flows down as far as necessary to display every character.

Setting a `clip` property does not move the layer or the content of the layer — only the visible area of the layer. Each adjustment has a unique impact on the apparent motion of the visible region. For example, if the `clip.left` value is increased from its original position of 0 to 20, the entire left edge of the rectangle shifts to the right by 20 pixels. No other edge moves. Changes to the `clip.width` property move only the right edge; changes to the `clip.height` property affect only the bottom edge. Unfortunately, no shortcuts exist to adjusting multiple edges at once. JavaScript is fast enough on most client machines to give the impression that multiple sides are moving if you issue assignment statements to different edges in sequence.

Example

Due to the edge movement behavior of adjustments to `layer.clip` properties, I have devised Listing 19-4 to let you experiment with adjustments to each of the six properties. The document loads one layer that is adjustable by entering alternative values into six text fields — one per property. Figure 19-2 shows the page.

As you enter values, all properties are updated to show their current values (via the `showValues()` function). Pay particular attention to the apparent motion of the edge and the effect the change has on at least one other property. For example, a change to the `layer.clip.left` value also affects the `layer.clip.width` property value.

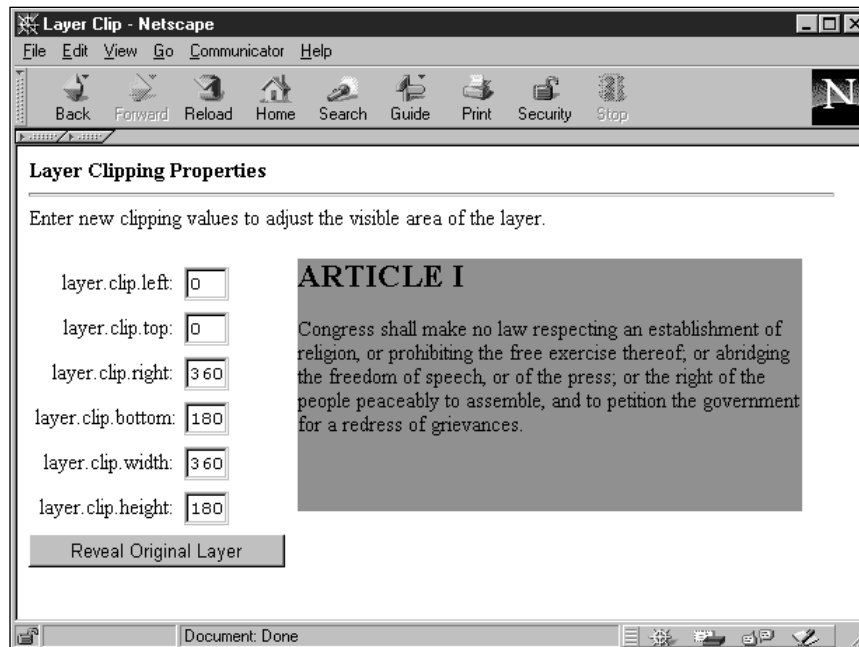


Figure 19-2: Experiment with layer.clip properties

Listing 19-4: Adjusting layer.clip properties

```
<HTML>
<HEAD>
<TITLE>Layer Clip</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var origLayerWidth = 0
var origLayerHeight = 0
function initializeXY() {
    origLayerWidth = document.display.clip.width
    origLayerHeight = document.display.clip.height
    showValues()
}

function setClip(field) {
    var clipVal = parseInt(field.value)
    document.display.clip[field.name] = clipVal
    showValues()
}

function showValues() {
    var form = document.layers[0].document.forms[0]
    var propName
    for (var i = 0; i < form.elements.length; i++) {
        propName = form.elements[i].name
        if (form.elements[i].type == "text") {
```

```

        form.elements[i].value = document.display.clip[propName]
    }
}
var intervalID
function revealClip() {
    var midWidth = Math.round(origLayerWidth / 2)
    var midHeight = Math.round(origLayerHeight / 2)
    document.display.clip.left = midWidth
    document.display.clip.top = midHeight
    document.display.clip.right = midWidth
    document.display.clip.bottom = midHeight
    intervalID = setInterval("stepClip()",1)
}
function stepClip() {
    var widthDone = false
    var heightDone = false
    if (document.display.clip.left > 0) {
        document.display.clip.left += -2
        document.display.clip.right += 2
    } else {
        widthDone = true
    }
    if (document.display.clip.top > 0) {
        document.display.clip.top += -1
        document.display.clip.bottom += 1
    } else {
        heightDone = true
    }
    showValues()
    if (widthDone && heightDone) {
        clearInterval(intervalID)
    }
}
</SCRIPT>
</HEAD>
<BODY onLoad="initializeXY()">
<B>Layer Clipping Properties</B>
<HR>
Enter new clipping values to adjust the visible area of the layer.<P>
<LAYER TOP=80>
<FORM>
<TABLE>
<TR>
    <TD ALIGN="right">layer.clip.left:</TD>
    <TD><INPUT TYPE="text" NAME="left" SIZE=3
onChange="setClip(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right">layer.clip.top:</TD>
    <TD><INPUT TYPE="text" NAME="top" SIZE=3
onChange="setClip(this)"></TD>
</TR>

```

(continued)

Listing 19-4 (*continued*)

```

<TR>
  <TD ALIGN="right">layer.clip.right:</TD>
  <TD><INPUT TYPE="text" NAME="right" SIZE=3
onChange="setClip(this)"></TD>
</TR>
<TR>
  <TD ALIGN="right">layer.clip.bottom:</TD>
  <TD><INPUT TYPE="text" NAME="bottom" SIZE=3
onChange="setClip(this)"></TD>
</TR>
<TR>
  <TD ALIGN="right">layer.clip.width:</TD>
  <TD><INPUT TYPE="text" NAME="width" SIZE=3
onChange="setClip(this)"></TD>
</TR>
<TR>
  <TD ALIGN="right">layer.clip.height:</TD>
  <TD><INPUT TYPE="text" NAME="height" SIZE=3
onChange="setClip(this)"></TD>
</TR>
</TABLE>
<INPUT TYPE="button" VALUE="Reveal Original Layer"
onClick="revealClip()">
</FORM>
</LAYER>
<LAYER NAME="display" BGCOLOR="coral" TOP=80 LEFT=200 WIDTH=360
HEIGHT=180>
<H2>ARTICLE I</H2>
<P>
Congress shall make no law respecting an establishment of religion, or
prohibiting the free exercise thereof; or abridging the freedom of
speech, or of the press; or the right of the people peaceably to
assemble, and to petition the government for a redress of grievances.
</P>
</LAYER>
</BODY>
</HTML>

```

Listing 19-4 has a lot of other scripting in it to demonstrate a couple other clip area techniques. After the document loads, the `onLoad=` event handler initializes two global variables that represent the starting height and width of the layer as determined from the `clip.height` and `clip.width` properties. Because the `<LAYER>` tag does not specify any CLIP attributes, the `layer.clip` region is ensured of being the same as the layer's dimensions at load time.

I preserve the initial values for a somewhat advanced set of functions that act in response to the Reveal Original Layer button. The goal of the button is to temporarily shrink the clipping area to nothing, and then expand the clip rectangle gradually from the very center of the layer. The effect is analogous to a zoom-out visual effect.

The clip region shrinks to practically nothing by setting all four edges to the same point midway along the height and width of the layer. The script then uses `setInterval()` to control the animation in `setClip()`. To make the zoom even on both axes, I first made sure that the initial size of the layer was an even ratio: Twice as wide as it is tall. Each time through the `setClip()` function, the `clip.left` and `clip.right` values are adjusted in their respective directions by two pixels, `clip.top` and `clip.bottom` by one pixel.

To make sure the animation stops when the layer is at its original size, I check whether the `clip.top` and `clip.left` values are their original zero values. If they are, I set a Boolean variable for each side. When both variables indicate that the clip rectangle is its original size, the script cancels the `setInterval()` action.

Related Items: `layer.pageX` property; `layer.pageY` property; `layer.resizeTo()` method.

document

Value: Document Object **Gettable:** Yes **Settable:** No

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

Your scripts will practically never have to retrieve the `document` property of a layer. But it is important to remember that it is always there as the actual container of content in the layer. As described at length in the opening section about the layer object, the document object reference plays a large role in assembling addresses to content items and properties in other layers. A document inside a layer has the same powers, properties, and methods of the main document in the browser window or a frame.

Related Items: document object.

left top

Value: Integer **Gettable:** Yes **Settable:** Yes

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

The `layer.left` and `layer.top` properties correspond to the `LEFT` and `TOP` attributes of the `<LAYER>` tag. These integer values determine the horizontal and vertical coordinate point of the top-left corner of the layer relative to the browser

window, frame, or parent layer in which it lives. The coordinate system of the layer's most immediate container is the one that these properties reflect.

Adjustments to these properties reposition the layer without adjusting its size. Clipping area values are untouched by changes in these properties. Thus, if you are creating a draggable layer object that needs to follow a dragged mouse pointer in a straight line along the x or y axis, it is more convenient to adjust one of these properties than to use the `layer.moveTo()` method.

Example

To let you experiment with manually setting `layer.top` and `layer.left` properties, I present a modified version of the `layer.clip` example (Listing 19-4) in Listing 19-5. The current example again has the one modifiable layer, but only four text fields in which you can enter values. Two fields are for the `layer.left` and `layer.top` properties; the other two are for the `layer.clip.left` and `layer.clip.top` properties. I present both sets of values here to help reinforce the lack of connection between layer and clip location properties in the same layer object.

Listing 19-5: Comparison of Layer and Clip Location Properties

```
<HTML>
<HEAD>
<TITLE>Layer vs. Clip</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function setClip(field) {
    var clipVal = parseInt(field.value)
    document.display.clip[field.name] = clipVal
    showValues()
}
function setLayer(field) {
    var layerVal = parseInt(field.value)
    document.display[field.name] = layerVal
    showValues()
}
function showValues() {
    var form = document.layers[0].document.forms[0]
    form.elements[0].value = document.display.left
    form.elements[1].value = document.display.top
    form.elements[2].value = document.display.clip.left
    form.elements[3].value = document.display.clip.top
}
</SCRIPT>
</HEAD>
<BODY onLoad="showValues()">
<B>Layer vs. Clip Location Properties</B>
<HR>
Enter new layer and clipping values to adjust the layer.<P>
<LAYER TOP=80>
<FORM>
<TABLE>
<TR>
    <TD ALIGN="right">layer.left:</TD>
```

```

        <TD><INPUT TYPE="text" NAME="left" SIZE=3
onChange="setLayer(this)"></TD>
</TR>
<TR>
        <TD ALIGN="right">layer.top:</TD>
        <TD><INPUT TYPE="text" NAME="top" SIZE=3
onChange="setLayer(this)"></TD>
</TR>
<TR>
        <TD ALIGN="right">layer.clip.left:</TD>
        <TD><INPUT TYPE="text" NAME="left" SIZE=3
onChange="setClip(this)"></TD>
</TR>
<TR>
        <TD ALIGN="right">layer.clip.top:</TD>
        <TD><INPUT TYPE="text" NAME="top" SIZE=3
onChange="setClip(this)"></TD>
</TR>
</TABLE>
</FORM>
</LAYER>
<LAYER NAME="display" BGCOLOR="coral" TOP=80 LEFT=200 WIDTH=360
HEIGHT=180>
<H2>ARTICLE I</H2>
<P>
Congress shall make no law respecting an establishment of religion, or
prohibiting the free exercise thereof; or abridging the freedom of
speech, or of the press; or the right of the people peaceably to
assemble, and to petition the government for a redress of grievances.
</P>
</LAYER>
</BODY>
</HTML>

```

Related Items: layer.clip properties; layer.parentLayer property.

name

Value: String **Gettable:** Yes **Settable:** No

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

The `layer.name` property reflects the `NAME` attribute of the `<LAYER>` tag or name you assign to a similar CSS-P object. This property is read-only. If you don't assign a name to a layer as it is being created, Navigator generates a name for the layer in the format:

```
js_layer_nn
```

where *nn* is a serial number. That serial number is not the same every time the page loads, so you cannot rely on the automatically generated name to help you script an absolute reference to the layer.

Related Items: None.

pageX pageY

Value: String **Gettable:** Yes **Settable:** Yes

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

In Netscape's coordinate terminology, the page is the content area of a browser window (or frame) object. The top-left corner of the page space is point 0,0, and any layer, including a nested layer, can be positioned on the page relative to this corner. In the `<LAYER>` tag, the attributes that let authors set the position are `PAGEX` and `PAGEY`. These values are retrievable and modifiable as the `layer.pageX` and `layer.pageY` properties. Note the capitalization of the final letters of these property names.

The `layer.pageX` and `layer.pageY` values are identical to `layer.left` and `layer.top` only when the layer in question is at the main document level. That's because the `layer.left` and `layer.top` values are measured by the next higher container's coordinate system — which in this case happens to be the same as the page.

The situation gets more interesting when you're dealing with nested layers. For a nested layer, the `layer.pageX` and `layer.pageY` values are still measured relative to the page, while `layer.left` and `layer.top` are measured relative to the next higher layer. If conceiving of these differences makes your head hurt, the example in Listing 19-6 should help clear things up for you.

Adjusting the `layer.pageX` and `layer.pageY` values of any layer has the same effect as using the `layer.moveToAbsolute()` method, which measures its coordinate system based on the page. If you are creating flying layers on your page, you won't go wrong by setting the `layer.pageX` and `layer.pageY` properties (or using the `moveToAbsolute()` method) in your script. That way, should you add another layer in the hierarchy between the base document and the flying layer, the animation will be in the same coordinate system as before the new layer was added.

Example

Listing 19-6 defines one outer layer and one nested inner layer of different colors (see Figure 19-3). The inner layer contains some text content, while the outer layer is sized initially to present a colorful border by being below the inner layer and 10 pixels wider and taller.

Two sets of fields display (and let you change) the `layer.pageX`, `layer.pageY`, `layer.left`, and `layer.top` properties for each of the nested layers. Each set of fields is color-coded to its corresponding layer.

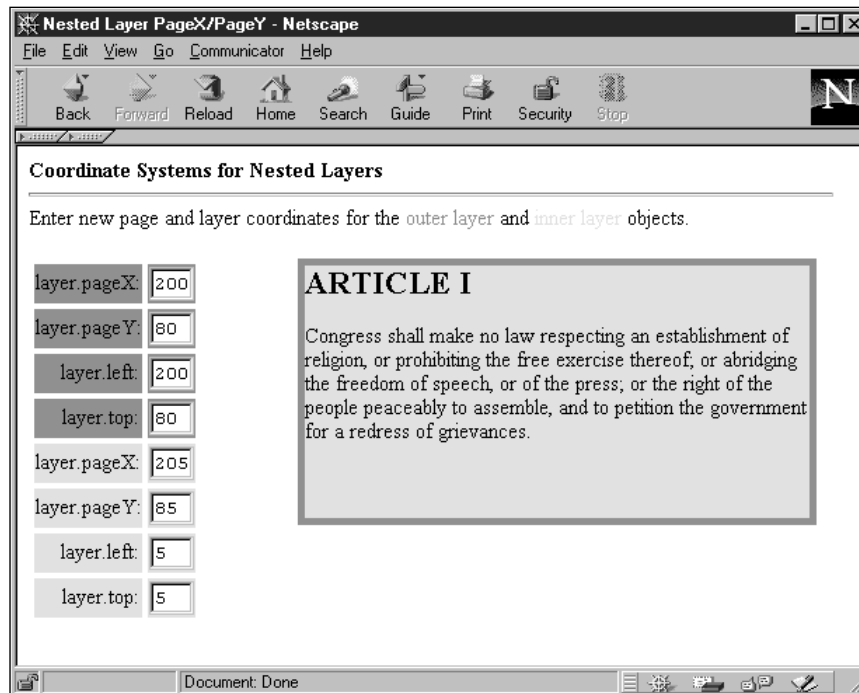


Figure 19-3: Testing the position properties of nested layers

When you change any value, all values are recalculated and displayed in the other fields. For example, the initial `pageX` position for the outer layer is 200 pixels; for the inner layer the `pageX` value is 205 pixels (accounting for the five-pixel “border” around the inner layer). If you change the outer layer’s `pageX` value to 220, the outer layer moves to the right by 20 pixels, taking the inner layer along for the ride. The `layer.pageX` value for the inner layer after the move is 225 pixels.

The outer layer values for the pairs of values are always the same no matter what. But for the inner layer, the page values are significantly different from the `layer.left` and `layer.top` values, because these latter values are measured relative to their containing layer, the outer layer. If you move the outer layer, the inner layer values for `layer.left` and `layer.top` don’t change one iota.

Listing 19-6: Testing Nested Layer Coordinate Systems

```
<HTML>
<HEAD>
<TITLE>Nested Layer PageX/PageY</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function setOuterPage(field) {
    var layerVal = parseInt(field.value)
    document.outerDisplay[field.name] = layerVal
    showValues()
}
```

(continued)

Listing 19-6 (*continued*)

```

}
function setOuterLayer(field) {
    var layerVal = parseInt(field.value)
    document.outerDisplay[field.name] = layerVal
    showValues()
}
function setInnerPage(field) {
    var layerVal = parseInt(field.value)
    document.outerDisplay.document.innerDisplay[field.name] =
layerVal
    showValues()
}
function setInnerLayer(field) {
    var layerVal = parseInt(field.value)
    document.outerDisplay.document.innerDisplay[field.name] =
layerVal
    showValues()
}
function showValues() {
    var form = document.layers[0].document.forms[0]
    form.elements[0].value = document.outerDisplay.pageX
    form.elements[1].value = document.outerDisplay.pageY
    form.elements[2].value = document.outerDisplay.left
    form.elements[3].value = document.outerDisplay.top
    form.elements[4].value =
document.outerDisplay.document.innerDisplay.pageX
    form.elements[5].value =
document.outerDisplay.document.innerDisplay.pageY
    form.elements[6].value =
document.outerDisplay.document.innerDisplay.left
    form.elements[7].value =
document.outerDisplay.document.innerDisplay.top
}
</SCRIPT>
</HEAD>
<BODY onLoad="showValues()">
<B>Coordinate Systems for Nested Layers</B>
<HR>
Enter new page and layer coordinates for the <FONT COLOR="coral">outer
layer</FONT> and <FONT COLOR="aquamarine">inner layer</FONT>
objects.<P>
<LAYER TOP=80>
<FORM>
<TABLE>
<TR>
    <TD ALIGN="right" BGCOLOR="coral">layer.pageX:</TD>
    <TD BGCOLOR="coral"><INPUT TYPE="text" NAME="pageX" SIZE=3
        onChange="setOuterPage(this)"></TD>
</TR>
<TR>

```

```

        <TD ALIGN="right" BGCOLOR="coral">layer.pageY:</TD>
        <TD BGCOLOR="coral"><INPUT TYPE="text" NAME="pageY" SIZE=3
            onChange="setOuterPage(this)"></TD>
    </TR>
    <TR>
        <TD ALIGN="right" BGCOLOR="coral">layer.left:</TD>
        <TD BGCOLOR="coral"><INPUT TYPE="text" NAME="left" SIZE=3
            onChange="setOuterLayer(this)"></TD>
    </TR>
    <TR>
        <TD ALIGN="right" BGCOLOR="coral">layer.top:</TD>
        <TD BGCOLOR="coral"><INPUT TYPE="text" NAME="top" SIZE=3
            onChange="setOuterLayer(this)"></TD>
    </TR>
    <TR>
        <TD ALIGN="right" BGCOLOR="aquamarine">layer.pageX:</TD>
        <TD BGCOLOR="aquamarine"><INPUT TYPE="text" NAME="pageX" SIZE=3
            onChange="setInnerPage(this)"></TD>
    </TR>
    <TR>
        <TD ALIGN="right" BGCOLOR="aquamarine">layer.pageY:</TD>
        <TD BGCOLOR="aquamarine"><INPUT TYPE="text" NAME="pageY" SIZE=3
            onChange="setInnerPage(this)"></TD>
    </TR>
    <TR>
        <TD ALIGN="right" BGCOLOR="aquamarine">layer.left:</TD>
        <TD BGCOLOR="aquamarine"><INPUT TYPE="text" NAME="left" SIZE=3
            onChange="setInnerLayer(this)"></TD>
    </TR>
    <TR>
        <TD ALIGN="right" BGCOLOR="aquamarine">layer.top:</TD>
        <TD BGCOLOR="aquamarine"><INPUT TYPE="text" NAME="top" SIZE=3
            onChange="setInnerLayer(this)"></TD>
    </TR>
</TABLE>
</FORM>
</LAYER>
<LAYER NAME="outerDisplay" BGCOLOR="coral" TOP=80 LEFT=200 WIDTH=370
HEIGHT=190>
<LAYER NAME="innerDisplay" BGCOLOR="aquamarine" TOP=5 LEFT=5 WIDTH=360
HEIGHT=180>
<H2>ARTICLE I</H2>
<P>
Congress shall make no law respecting an establishment of religion, or
prohibiting the free exercise thereof; or abridging the freedom of
speech, or of the press; or the right of the people peaceably to
assemble, and to petition the government for a redress of grievances.
</P>
</LAYER>
</LAYER>
</BODY>
</HTML>

```

Related Items: `layer.left` property; `layer.top` property; `layer.moveToAbsolute()` method; `window.innerHeight` property; `window.innerWidth` property.

parentLayer

Value: Object **Gettable:** Yes **Settable:** No

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

Every layer has a parent that contains that layer. In the case of a layer defined at the main document level, its parent layer is the window or frame containing that document (the “page”). For this kind of layer, the `layer.parentLayer` property object is a window object. But for any nested layer contained by a layer, the `parentLayer` property is a layer object.

Be aware of the important distinction between `layer.parentLayer` and `layer.below`. As a parent layer can contain multiple layers in the next containment level, each of those layers’ `parentLayer` properties evaluate to that same parent layer. But because each layer object is its own physical layer among the stack of layers on a page, the `layer.below` property in each layer points to a different object — the layer next lower in z-order.

Keeping the direction of things straight can get confusing. On the one hand, you have a layer’s parent, which, by connotation, is higher up the hierarchical chain of layers. On the other hand, the order of physical layers is such that a parent more than likely has a lower z-order than its children because it is defined earlier in the document.

The `layer.parentLayer` property is used primarily to assemble references to other nested layers. See the discussion about layer references at the beginning of this chapter for several syntax examples.

Related Items: `layer.above` property; `layer.below` property.

siblingAbove

siblingBelow

(See `layer.above` and `layer.below` properties earlier in this chapter.)

src

Value: String **Gettable:** Yes **Settable:** Yes

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

Content for a layer may come from within the document that defines the layer or from an external source, such as an HTML or image file. If defined by a `<LAYER>` tag, an external file is specified by the `SRC` attribute. It is this attribute that is reflected by the `layer.src` property.

The value of this property is a string of the URL of the external file. If no `SRC` property is specified in the `<LAYER>` tag, the value returns `null`. Do not set this property to an empty string in an effort to clear the layer of content: `document.write()` or load an empty page instead. Otherwise, the empty string is treated like a URL and loads the current client directory.

You can, however, change the content of a layer by loading a new source file into the layer. Simply assign a new URL to the `layer.src` property. Again, if a layer has nested layers inside it, those nested layers get blown away by the content that loads into the layer whose `src` property you are changing. The new file, of course, might be an HTML file that defines its own nested layers, which then become part of the page's object model.

Netscape also provides the `layer.load()` method to put new content into a layer. One advantage of the method is that an optional second parameter lets you redefine the width of the layer at the same time as you specify a new document. But if you are simply replacing the content in the same width layer, you can use either way of loading new content.

Be aware that the height and width of a replacement layer is as much governed by its hard-coded content size as by the initial loading of any layer. For example, if your layer is initially sized at a width of 200 pixels and your replacement layer document includes an image whose width is set to 500 pixels, the layer expands its width to accommodate the larger content — unless you also restrict the view of the layer via the `layer.clip` properties. Similarly, longer text content flows beyond the bottom of the previously sized layer unless restricted by clipping properties.

Example

Setting the `layer.src` property of a layer that is a member of a layer family (that is, one with at least one parent and one child) can be tricky business if you're not careful. Listing 19-7 presents a workspace for you to see how changing the `src` property of outer and inner layers affects the scenery.

When you first load the document, one outer layer contains one inner layer (each with a different background color). Control buttons on the page let you set the `layer.src` property of each layer independently. Changes to the inner layer content affect only that layer. Long content forces the inner layer to expand its depth, but its view is automatically clipped by its parent layer.

Changing the outer layer content, however, removes the inner layer completely. Code in the listing shows one way to examine for the presence of a particular layer before attempting to load new content in it. If the inner layer doesn't exist, the script creates a new layer on the fly to replace the original inner layer.

Listing 19-7: Setting Nested Layer Source Content

```
<HTML>
<HEAD>
<TITLE>Layer Source</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function loadOuter(doc) {
    document.outerDisplay.src = doc
}
function loadInner(doc) {
    var nested = document.outerDisplay.document.layers
    if (nested.length > 0) {
        // inner layer exists, so load content or restore
        if (doc) {
            nested[0].src = doc
        } else {
            restoreInner(nested[0])
        }
    } else {
        // prompt user about restoring inner layer
        if (confirm("The inner layer has been removed by loading an
outer document. Restore the original layers?")) {
            restoreLayers(doc)
        }
    }
}
function restoreLayers(doc) {
    // reset appearance of outer layer
    document.outerDisplay.bgColor = "coral"
    document.outerDisplay.resizeTo(370,190) // sets clip
    document.outerDisplay.document.write("")
    document.outerDisplay.document.close()
    // generate new inner layer
    var newInner = new Layer(360, document.layers["outerDisplay"])
    newInner.bgColor = "aquamarine"
    newInner.moveTo(5,5)
    if (doc) {
        // user clicked an inner content button
        newInner.src = doc
    } else {
        // return to pristine look
        restoreInner(newInner)
    }
    newInner.visibility = "show"
}
function restoreInner(inner) {
    inner.document.write("<HTML><BODY><P><B>Placeholder text for raw
inner layer.</B></P></BODY></HTML>")
    inner.document.close()
    inner.resizeTo(360,180) // sets clip
}
```

```

</SCRIPT>
</HEAD>
<BODY>
<B>Setting the <TT>layer.src</TT> Property of Nested Layers</B>
<HR>
Click the buttons to see what happens when you load new source
documents into the <FONT COLOR="coral">outer layer</FONT> and <FONT
COLOR="aquamarine">inner layer</FONT> objects.<P>
<LAYER TOP=100 BGCOLOR="coral">
<FORM>
Load into outer layer:<BR>
<INPUT TYPE="button" VALUE="Article I"
onClick="loadOuter('article1.htm')"><BR>
<INPUT TYPE="button" VALUE="Entire Bill of Rights"
onClick="loadOuter('bofright.htm')"><BR>
</FORM>
</LAYER>
<LAYER TOP=220 BGCOLOR="aquamarine">
<FORM>
Load into inner layer:<BR>
<INPUT TYPE="button" VALUE="Article I"
onClick="loadInner('article1.htm')"><BR>
<INPUT TYPE="button" VALUE="Entire Bill of Rights"
onClick="loadInner('bofright.htm')"><BR>
<INPUT TYPE="button" VALUE="Restore Original"
onClick="loadInner()"><BR>
</FORM>
</LAYER>
<LAYER NAME="outerDisplay" BGCOLOR="coral" TOP=100 LEFT=200 WIDTH=370
HEIGHT=190>
  <LAYER NAME="innerDisplay" BGCOLOR="aquamarine" TOP=5 LEFT=5
WIDTH=360 HEIGHT=180>
    <P><B>Placeholder text for raw inner layer.</B></P>
  </LAYER>
</LAYER>
</BODY>
</HTML>

```

The restoration of the original layers via script (as opposed to reloading the document) does not perform a perfect restoration. The key difference is the scripts use the `layer.resizeTo()` method to set the layers to the height and width established by the `<LAYER>` tags that created the layers in the first place. This method, however, sets the clipping rectangle of the layer, not the layer's size. Therefore, if you use the script to restore the layers, loading the longer text file into either layer does not force the layer to expand to display all the content: The clipping region governs the view.

Related Items: `layer.load()` method; `layer.resizeTo()` method.

visibility

Value: String **Gettable:** Yes **Settable:** Yes

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

A layer's `visibility` property can use one of three settings: `show`, `hide`, or `inherit` — the same values you can assign to the `VISIBILITY` attribute of the `<LAYER>` tag. But to be compatible with the same property in Internet Explorer 4, Navigator 4 also lets you set the property to `hidden` and `visible`.

Unlike many other layer properties, the `visibility` property can be set such that a layer can either follow the behavior of its parent or strike out on its own. By default, a layer's `visibility` property is set to `inherit`, which means the layer's visibility is governed solely by that of its parent (and of *its* parent, if the nesting goes many layers). When the governing parent's property is, say, `hide`, the child's property remains `inherit`. Thus, you cannot tell whether an inheriting layer is presently visible or not without checking up the hierarchy (with the help of the `layer.parentLayer` property). However, you can override the parent's behavior by setting the current layer's property explicitly to `show` or `hide`. This action does not alter in any way other parent-child relationships between layers.

Example

Use the page in Listing 19-8 to see how the `layer.visibility` property settings affect a pair of nested layers. When the page first loads, the default `inherit` setting is in effect. Changes you make to the outer layer by clicking on the outer layer buttons affect the inner layer, but setting the inner layer's properties to `hide` or `show` severs the visibility relationship between parent and child.

Listing 19-8: Nested Layer Visibility Relationships

```
<HTML>
<HEAD>
<TITLE>Layer Source</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function setOuterVis(type) {
    document.outerDisplay.visibility = type
}
function setInnerVis(type) {
    document.outerDisplay.document.innerDisplay.visibility = type
}
</SCRIPT>
</HEAD>
<BODY>
<B>Setting the <TT>layer.visibility</TT> Property of Nested Layers</B>
<HR>
```

```

Click the buttons to see what happens when you change the visibility
of the <FONT COLOR="coral">outer layer</FONT> and <FONT
COLOR="aquamarine">inner layer</FONT> objects.<P>
<LAYER TOP=100 BGCOLOR="coral">
<FORM>
Control outer layer property:<BR>
<INPUT TYPE="button" VALUE="Hide Outer Layer"
onClick="setOuterVis('hide')"><BR>
<INPUT TYPE="button" VALUE="Show Outer Layer"
onClick="setOuterVis('show')"><BR>
</FORM>
</LAYER>
<LAYER TOP=220 BGCOLOR="aquamarine">
<FORM>
Control inner layer property:<BR>
<INPUT TYPE="button" VALUE="Hide Inner Layer"
onClick="setInnerVis('hide')"><BR>
<INPUT TYPE="button" VALUE="Show Inner Layer"
onClick="setInnerVis('show')"><BR>
<INPUT TYPE="button" VALUE="Inherit Outer Layer"
onClick="setInnerVis('inherit')"><BR>
</FORM>
</LAYER>
<LAYER NAME="outerDisplay" BGCOLOR="coral" TOP=100 LEFT=200 WIDTH=370
HEIGHT=190>
    <LAYER NAME="innerDisplay" BGCOLOR="aquamarine" TOP=5 LEFT=5
WIDTH=360 HEIGHT=180>
        <P><B>Placeholder text for raw inner layer.</B></P>
    </LAYER>
</LAYER>
</BODY>
</HTML>

```

Related Items: None.

zIndex

Value: Integer **Gettable:** Yes **Settable:** Yes

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

Close relationships exist among the `layer.above`, `layer.below`, and `layer.zIndex` properties. When you define a layer in a document with the `<LAYER>` tag, you can supply only one of the three attributes (`ABOVE`, `BELOW`, and `Z-INDEX`). After the layer is generated with any one of those attributes, the document object model automatically assigns values to at least two of those properties (`layer.above` and `layer.below`) unless you specify the `Z-INDEX`.

attribute, in which case all three properties are assigned to the layer. If you don't specify any of these properties, the physical stacking order of the layers is the same as in the HTML document. The `layer.above` and `layer.below` properties are set as described in their discussion earlier in this chapter. But the `layer.zIndex` properties for all layers are zero.

Note

The attribute is spelled with a hyphen after the “z”. Because a JavaScript property name cannot contain a hyphen, the character has been removed for the property name. The capital “I” is important, because JavaScript properties are case-sensitive.

Changes to `layer.zIndex` values affect the stacking order only of sibling layers. It is possible to assign the same value to two layers, but the last layer to have its `layer.zIndex` property set will be physically above the other one. Therefore, if you want to ensure a stacking order, you should set the `zIndex` values for all layers within a container. Each value should be a unique number.

Stacking order is determined simply by the value of the integer assigned to the property. If you want to stack three sibling layers, the order would be the same if they were assigned the values of 1, 2, 3 or 10, 13, 50. As you modify a `layer.zIndex` value, the `layer.above` and `layer.below` properties for all affected layers change as a result.

Caution

Avoid setting `zIndex` property values to negative numbers in Navigator 4. Negative values are treated as their absolute (positive) values for ordering.

Example

The relationships among the three stacking properties can be difficult to visualize. Listing 19-9 offers a way to see the results of changing the `layer.zIndex` properties of three overlapping sibling layers. The beginning organization of layers after the page loads is shown in Figure 19-4.

Original stacking order is governed by the position of the `<LAYER>` tags in the document. Because the attribute is not set in the HTML, the initial values appear as zero for all three layers. But, as the page reveals, the `layer.above` and `layer.below` properties are automatically established. When a layer has no other layer object above it, the page shows “(none)”. Also, if the layer below the bottom of the stack is the main window, a strange inner layer name is assigned (something like `_js_layer_21`).

To experiment with this page, first make sure you understand the `layer.above` and `layer.below` readings for the default order of the layers. Then assign different orders to the layers with value sequences such as 3-2-1, 1-3-2, 2-2-2, and so on. Each time you enter one new value, check the actual layers to see if their stacking order changed and how that affected the other properties of all layers.

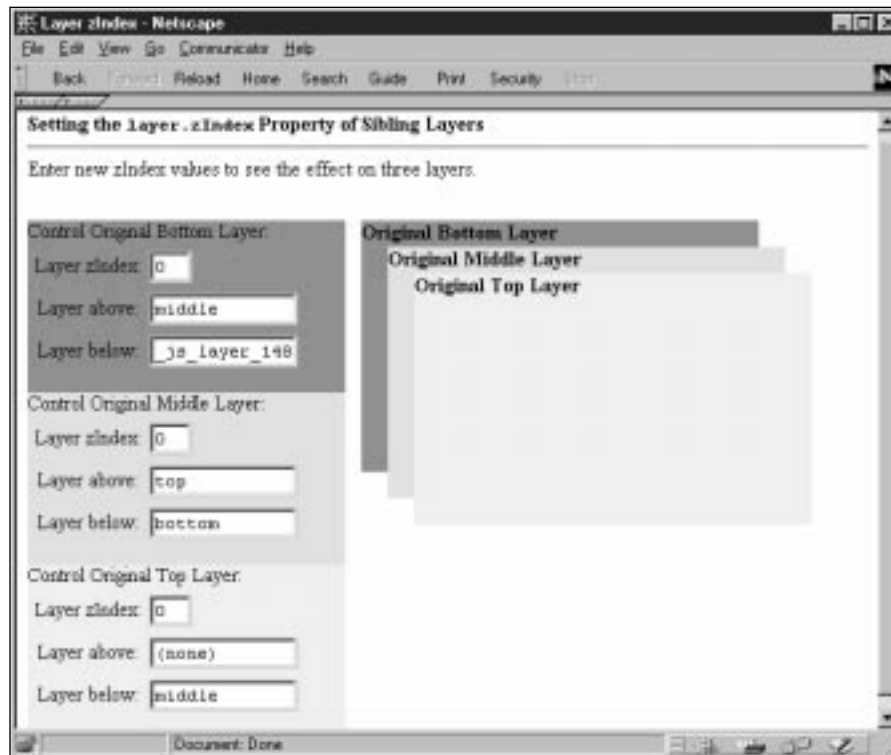


Figure 19-4: A place to play with zIndex property settings

Listing 19-9: Relationships among zIndex, Above and Below

```
<HTML>
<HEAD>
<TITLE>Layer zIndex</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function setZ(field) {
    switch (field.name) {
        case "top" :
            document.top.zIndex = parseInt(field.value)
            break
        case "mid" :
            document.middle.zIndex = parseInt(field.value)
            break
        case "bot" :
            document.bottom.zIndex = parseInt(field.value)
    }
    showValues()
}
function showValues() {
```

(continued)

Listing 19-9 (*continued*)

```

        document.layers[0].document.forms[0].bot.value =
document.bottom.zIndex
        document.layers[1].document.forms[0].mid.value =
document.middle.zIndex
        document.layers[2].document.forms[0].top.value =
document.top.zIndex

        document.layers[0].document.forms[0].above.value =
(document.bottom.above) ? document.bottom.above.name : "(none)"
        document.layers[1].document.forms[0].above.value =
(document.middle.above) ? document.middle.above.name : "(none)"
        document.layers[2].document.forms[0].above.value =
(document.top.above) ? document.top.above.name : "(none)"

        document.layers[0].document.forms[0].below.value =
(document.bottom.below) ? document.bottom.below.name : "(none)"
        document.layers[1].document.forms[0].below.value =
(document.middle.below) ? document.middle.below.name : "(none)"
        document.layers[2].document.forms[0].below.value =
(document.top.below) ? document.top.below.name : "(none)"
    }
</SCRIPT>
</HEAD>
<BODY onLoad="showValues()">
<B>Setting the <TT>layer.zIndex</TT> Property of Sibling Layers</B>
<HR>
Enter new zIndex values to see the effect on three layers.<P>
<LAYER TOP=90 WIDTH=240 BGCOLOR="coral">
<FORM>
Control Original Bottom Layer:<BR>
<TABLE>
<TR><TD ALIGN="right">Layer zIndex:</TD><TD><INPUT TYPE="text"
NAME="bot" SIZE=3 onChange="setZ(this)"></TD></TR>
<TR><TD ALIGN="right">Layer above:</TD><TD><INPUT TYPE="text"
NAME="above" SIZE=13></TD></TR>
<TR><TD ALIGN="right">Layer below:</TD><TD><INPUT TYPE="text"
NAME="below" SIZE=13></TD></TR>
</TABLE>
</FORM>
</LAYER>
<LAYER TOP=220 WIDTH=240 BGCOLOR="aquamarine">
<FORM>
Control Original Middle Layer:<BR>
<TABLE>
<TR><TD ALIGN="right">Layer zIndex:</TD><TD><INPUT TYPE="text"
NAME="mid" SIZE=3 onChange="setZ(this)"></TD></TR>
<TR><TD ALIGN="right">Layer above:</TD><TD><INPUT TYPE="text"
NAME="above" SIZE=13></TD></TR>
<TR><TD ALIGN="right">Layer below:</TD><TD><INPUT TYPE="text"
NAME="below" SIZE=13></TD></TR>

```



```

</TABLE></FORM>
</LAYER>
<LAYER TOP=350 WIDTH=240 BGCOLOR="yellow">
<FORM>
Control Original Top Layer:<BR>
<TABLE><TR><TD ALIGN="right">Layer zIndex:</TD><TD><INPUT TYPE="text"
NAME="top" SIZE=3 onChange="setZ(this)"></TD></TR>
<TR><TD ALIGN="right">Layer above:</TD><TD><INPUT TYPE="text"
NAME="above" SIZE=13></TD></TR>
<TR><TD ALIGN="right">Layer below:</TD><TD><INPUT TYPE="text"
NAME="below" SIZE=13></TD></TR>
</TABLE>
</FORM>
</LAYER>
<LAYER NAME="bottom" BGCOLOR="coral" TOP=90 LEFT=260 WIDTH=300
HEIGHT=190>
    <P><B>Original Bottom Layer</B></P>
</LAYER>
    <LAYER NAME="middle" BGCOLOR="aquamarine" TOP=110 LEFT=280
WIDTH=300 HEIGHT=190>
    <P><B>Original Middle Layer</B></P>
</LAYER>
<LAYER NAME="top" BGCOLOR="yellow" TOP=130 LEFT=300 WIDTH=300
HEIGHT=190>
    <P><B>Original Top Layer</B></P>
</LAYER>
</LAYER>
</BODY>
</HTML>

```

Related Items: `layer.above` property; `layer.below` property; `layer.moveAbove()` method; `layer.moveBelow()` method.

Methods

`load("URL", newLayerWidth)`

Returns: Nothing.

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

One way to change the content of a layer after it has loaded is to use the `layer.load()` method. This method has an advantage over setting the `layer.src` property, because the second parameter is a new layer width for the content if one is desired. If you don't specify the second parameter, a small default value is substituted for you (unless the new document has hard-wired widths to its

elements that must expand the current width). If you are concerned about a new document being too long for the existing height of the layer, use the `layer.resizeTo()` method or set the individual `layer.clip` properties before loading the new document. This keeps the viewable area of the layer at a fixed size.

Example

Buttons in Listing 19-10 let you load short and long documents into a layer. The first two buttons don't change the width (in fact, the second parameter to `layer.load()` is the `layer.clip.left` value). For the second two buttons, a narrower width than the original is specified. Click the Restore button frequently to return to a known state.

Listing 19-10: Loading Documents into Layers

```
<HTML>
<HEAD>
<TITLE>Layer zIndex</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function loadDoc(URL,width) {
    if (!width) {
        width = document.myLayer.clip.width
    }
    document.myLayer.load(URL, width)
}
</SCRIPT>
</HEAD>
<BODY>
<B>Loading New Documents</B>
<HR>
<LAYER TOP=90 WIDTH=240 BGCOLOR="yellow">
<FORM>
Loading new documents:<BR>
<INPUT TYPE="button" VALUE="Small Doc/Existing Width"
onClick="loadDoc('article1.htm')"><BR>
<INPUT TYPE="button" VALUE="Large Doc/Existing Width"
onClick="loadDoc('bofright.htm')"><P>
<INPUT TYPE="button" VALUE="Small Doc/Narrower Width"
onClick="loadDoc('article1.htm',200)"><BR>
<INPUT TYPE="button" VALUE="Large Doc/Narrower Width"
onClick="loadDoc('bofright.htm',200)"><P>
<INPUT TYPE="button" VALUE="Restore"
onClick="location.reload()"></FORM>
</LAYER>
<LAYER NAME="myLayer" BGCOLOR="yellow" TOP=90 LEFT=300 WIDTH=300
HEIGHT=190>
    <P><B>Text loaded in original document.</B></P>
</LAYER>
</BODY>
</HTML>
```

Related Items: `layer.src` property.

`moveAbove(layerObject)`
`moveBelow(layerObject)`

Returns: Nothing.

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

With the exception of the `layer.zIndex` property, the layer object does not let you adjust properties that affect the global stacking order of layers. The `layer.moveAbove()` and `layer.moveBelow()` methods let you adjust a layer in relation to another layer object. Both layers in the transaction must be siblings — they must be in the same container, whether it be the base document window or some other layer. You cannot move existing layers from one container to another, but must delete the layer from the source and create a new layer in the destination. Neither of these methods affects the viewable size or coordinate system location of the layer.

The syntax for these methods is a little strange at first, because the statement that makes these work has two layer object references in it. Named first in the statement (to the left of the method name, separated by a period) is the layer object you want to move. The sole parameter for each method is a reference to the layer object that is the physical reference point for the trip. For example, in the statement

```
document.fred.moveAbove(document.ginger)
```

the instruction is to move the `fred` layer above the `ginger` layer. The `fred` layer can be in any stacking relation to `ginger`, but, again, both must be in the same container.

Obviously, after one of these moves, the `layer.above` and `layer.below` properties of some or all layers in the container feel the ripple effects of the shift in order. If you have several layers that may have gotten out of order due to user interaction with your scripts, you can reorder them using these methods or, more practically, by setting their `layer.zIndex` properties. In the latter case, it is easier to visualize through your code how the ordering is being handled with increasing `zIndex` values for each layer.

Example

You can see the `layer.moveAbove()` method at work in Listing 19-1, earlier in this chapter.

Related Items: `layer.above` property; `layer.below` property; `layer.zIndex` property.

```
moveBy(deltaX,deltaY)
moveTo(x,y)
moveToAbsolute(x,y)
```

Returns: Nothing.

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

Much of what CSS-Positioning is all about is being able to precisely plant an element on a Web page. The unit of measure is the pixel, with the coordinate space starting at an upper-left corner at location 0,0. That coordinate space for a layer is typically the container (parent layer) for that layer. The `layer.moveTo()` and `layer.moveBy()` methods let scripts adjust the location of a layer inside that coordinate space — very much the way `window.moveTo()` and `window.moveBy()` work for window objects.

Moving a layer entails moving it (and its nested layers) without adjusting its size or stacking order. Animation of a layer can be accomplished by issuing a series of `layer.moveTo()` methods if you know the precise points along the path. Or you can nudge the layer by increments in one or both axes with the `layer.moveBy()` method.

In case you need to position a layer with respect to the page's coordinate system (for example, you are moving items from multiple containers to a common point), the `layer.moveToAbsolute()` method bypasses the layer's immediate container. The 0,0 point for this method is the top-left corner of the content window or frame. Be aware, however, that you can move a layer to a position such that some or all of it is out of range of the container's clip rectangle.

Example

A demonstration of the `layer.moveTo()` method is shown in Listing 19-11. It is a simple script that lets you click and drag a layer around the screen. The script employs the coordinate values of the `MouseMove` event and, after compensating for the offset within the layer at which the click occurs, moves the layer to track the mouse action.

I wanted to present this example for an additional reason: to explain an important user interface difference between Windows and Macintosh versions of Navigator 4. In Windows versions, you can click and hold the mouse button down on an object and let the object receive all the `MouseMove` events as you drag the cursor around the screen. On the Macintosh, however, Navigator tries to compensate for the lack of a second mouse button by popping up a context-sensitive menu at the cursor position when the user holds the mouse button down for more than just a click. To replicate the action of dragging on the Macintosh, users must Ctrl-click (and release). That action engages a drag mode that remains in effect until the user clicks again. I suspect that fewer than ten percent of Mac Navigator users know this. Therefore, if you script any layer dragging for a cross-

platform audience, be sure to include instructions for Mac users about how to effect the drag (you can add this only for Mac users by analyzing the `navigator.userAgent` string, as shown in Listing 19-11).

Notice in the listing how the layer captures a number of mouse events. Each one plays an important role in creating a mode that is essentially like a `mouseStillDown` event (which doesn't exist in Navigator's event model). The `mousedown` event sets a Boolean flag (`engaged`) indicating that the user clicked down in the layer. At the same time, the script records how far away from the layer's top-left corner the `mousedown` event occurred. This offset information is needed so that any setting of the layer's location takes this offset into account (otherwise the top-left corner of the layer would jump to the cursor position and be dragged from there).

During the drag (`mousedown` events firing with each mouse movement), the `dragIt()` function checks whether the drag mode is engaged. If so, the layer is moved to the page location calculated by subtracting the original downstroke offset from the `mousemove` event location on the page. When the user releases the mouse button, the `mouseup` event turns off the drag mode Boolean value.

Listing 19-11: Dragging a Layer

```
<HTML>
<HEAD>
<TITLE>Layer Dragging</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var engaged = false
var offsetX = 0
var offsetY = 0
function dragIt(e) {
    if (engaged) {
        document.myLayer.moveTo(e.pageX - offsetX, e.pageY -
offsetY)
    }
}
function engage(e) {
    engaged = true
    offsetX = e.pageX - document.myLayer.left
    offsetY = e.pageY - document.myLayer.top
}
function release() {
    engaged = false
}
</SCRIPT>
</HEAD>
<BODY>
<B>Dragging a Layer</B>
<HR>
<LAYER NAME="myLayer" BGCOLOR="lightgreen" TOP=90 LEFT=100 WIDTH=300
HEIGHT=190>
    <P><B>Drag me around the window.</B></P>
    <SCRIPT LANGUAGE="JavaScript">
        if (navigator.userAgent.indexOf("Mac") != -1) {
```

(continued)

Listing 19-11 (*continued*)

```

        document.write(" (Mac users: Ctrl-Click me first; then Click
to stop dragging.) ")
    }
    </SCRIPT>
</LAYER>
<SCRIPT LANGUAGE="JavaScript">
document.myLayer.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP |
Event.MOUSEMOVE)
document.myLayer.onMouseDown = engage
document.myLayer.onMouseUp = release
document.myLayer.onMouseMove = dragIt
</SCRIPT>
</BODY>
</HTML>

```

Related Items: `layer.resizeBy()` **method**; `layer.resizeTo()` **method**;
`window.moveBy()` **method**; `window.moveTo()` **method**.

`resizeBy(deltaX,deltaY)` `resizeTo(width,height)`

Returns: Nothing.

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

The basic functionality and parameter requirements of the `layer.resizeBy()` and `layer.resizeTo()` methods are similar to the identically named methods of the window object. You should, however, be cognizant of some considerations unique to layers.

Unlike resizing a window, which causes all content to reflow to fit the new size, the layer sizing methods aren't truly adjusting the size of the layer. Instead, the methods control the size of the clipping region of the layer. Therefore, the content of the layer does not automatically reflow when you use these methods — any more than they do when you change individual `layer.clip` values.

Another impact of this clipping region relationship has to do with content that extends beyond the bounds of the layer. For example, if you provide `HEIGHT` and `WIDTH` attributes to a `<LAYER>` tag, content that requires more space to display itself than those attribute settings automatically expands the viewable area of the layer. To rein in such runaway content, you can set the `CLIP` attribute. But because the layer resize methods adjust the clipping rectangle, oversized content won't overflow the `<LAYER>` tag's height and width settings. This may be beneficial for you or not, depending on your design intentions. Adjusting the size of a layer with

either method affects only the position of the right and bottom edges of the layer. The top-left location of the layer does not move.

Example

It is important to understand the ramifications of content flow when a layer is resized by these two methods. Listing 19-12 (and companion document Listing 19-13) shows you how to set the lower-right corner of a layer to be dragged by a user for resizing the layer (much like grabbing the resize corner of a document window). Three radio buttons let you choose whether and when the content should be redrawn to the layer: never, after resizing, or during resizing.

Event capture is very much like that in Listing 19-11 for layer dragging. The primary difference is that drag mode is engaged only when the mouse event takes place in the region of the lower-right corner. A different kind of offset value is saved here than for dragging, because for resizing, the script needs to know the mouse event offset from the right and bottom edges of the layer.

Condition statements in the `resizeIt()` and `release()` functions check whether a specific radio button is checked to determine when (or if) the content should be redrawn. I design this page with the knowledge that its content might be redrawn. Therefore, I build the content of the layer as a separate HTML document that is loaded in the `<LAYER>` tag.

Redrawing the content requires reloading the document into the layer. I use the `layer.reload()` method, because I want to send the current `layer.clip.width` as a parameter for the width of the clip region to accommodate the content as it loads.

An important point to know about reloading content into a layer is that all property settings for the layer's event capture are erased when the document loads. To overcome this behavior requires setting the layer's `onLoad=` event handler to set the layer's event capture mechanism. If the layer event capturing had been specified as part of the statements at the end of the document, the layer would ignore some important events needed for the dynamic resizing after the document reloaded the first time.

As you experiment with the different feel for resizing and redrawing behavior, you will see that redrawing during resizing is a slow process due to the repetitive loading (from cache) needed each time. On slower client machines, it is easy for the cursor to outrun the layer region, causing the layer to not get `mouseover` events at all. It may not be the best-looking solution, but I prefer to redraw after resizing the layer.

Listing 19-12: Resizing a Layer

```
<HTML>
<HEAD>
<TITLE>Layer Resizing</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var engaged = false
var offsetX = 0
var offsetY = 0
function resizeIt(e) {
    if (engaged) {
```

(continued)

Listing 19-12 (*continued*)

```

        document.myLayer.resizeTo(e.pageX + offsetX, e.pageY +
offsetY)
        if (document.forms[0].redraw[2].checked) {
            document.myLayer.load("lst19-13.htm",
document.myLayer.clip.width)
        }
    }
    function engage(e) {
        if (e.pageX > (document.myLayer.clip.right - 10) && e.pageY >
(document.myLayer.clip.bottom - 10)) {
            engaged = true
            offsetX = document.myLayer.clip.right - e.pageX
            offsetY = document.myLayer.clip.bottom - e.pageY
        }
    }
    function release() {
        if (engaged && document.forms[0].redraw[1].checked) {
            document.myLayer.load("lst19-13.htm",
document.myLayer.clip.width)
        }
        engaged = false
    }
    function grabEvents() {
        document.myLayer.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP |
Event.MOUSEMOVE)
    }
</SCRIPT>
</HEAD>
<BODY>
<B>Resizing a Layer</B>
<HR>
<FORM>
Redraw layer content:<BR>
<INPUT TYPE="radio" NAME="redraw" CHECKED>Never
<INPUT TYPE="radio" NAME="redraw">After resize
<INPUT TYPE="radio" NAME="redraw">During resize
</FORM>
<LAYER NAME="myLayer" SRC="lst19-13.htm" BGCOLOR="lightblue" TOP=120
LEFT=100 WIDTH=300 HEIGHT=190 onLoad="grabEvents()">
</LAYER>
<SCRIPT LANGUAGE="JavaScript">
document.myLayer.onMouseDown = engage
document.myLayer.onMouseUp = release
document.myLayer.onMouseMove = resizeIt
</SCRIPT>
</BODY>
</HTML>

```


Listing 19-13: Content for the Resizable Layer

```

<HTML>
<BODY>
  <P><B>Resize me by dragging the lower right corner.</B></P>
  <SCRIPT LANGUAGE="JavaScript">
    if (navigator.userAgent.indexOf("Mac") != -1) {
      document.write("(Mac users: Ctrl-Click me first; then Click
to stop dragging.)")
    }
  </SCRIPT>
</BODY>
</HTML>

```

Related Items: `layer.moveBy()` method; `layer.moveTo()` method; `window.resizeBy()` method; `window.resizeTo()` method.

Event handlers

`onBlur=`
`onFocus=`

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

A user gets no visual clue when a layer receives focus. But a click on the clipping region of a layer triggers a focus event that can be handled with an `onFocus=` event handler. Clicking anywhere on the page outside of that layer area fires a blur event. Changing the stacking order of sibling layers does not fire either event unless mouse activity occurs in one of the layers.

If your layer contains elements that have their own focus and blur events (such as text fields), those objects' event handlers still fire, even if you also have the same event handlers defined for the layer. The layer's events fire after the text field's events.

Unlike comparable event handlers in windows, layer events for blur and focus do not have companion methods to bring a layer into focus or to blur it. However, if you use the `focus()` and/or `blur()` methods on applicable form elements in a layer, the layer's corresponding event handlers are triggered as a result.

Related Items: `textbox.blur()` method; `textbox.focus()` method.

`onLoad=`

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

Scripting layers can sometimes lead to instances of unfortunate sequences of loading. For example, if you want to set some layer object properties via a script (that is, not in the `<LAYER>` tag), you can do so only after the layer object exists in the document object model. One way to make sure the object exists is to place the scripting in `<SCRIPT>` tags at the end of the document. Another is to specify an `onLoad=` event handler in the tag, as shown in Listing 19-12.

Each time you load a document into the layer — either via the `SRC` attribute in the `<LAYER>` tag or by invoking the `layer.load()` method, the `onLoad=` event handler runs. But also be aware that an interaction occurs between a layer's `onLoad=` event handler and an `onLoad=` event handler in the `<BODY>` tag of a document loaded into a layer. If the document body has an `onLoad=` event handler, then the layer's `onLoad=` event handler does not fire. You get one or the other, but not both.

Related Items: `window.onLoad=` event handler.

onMouseOut= onMouseOver=

	Nav2	Nav3	Nav4	IE3/J1	IE3/J2	IE4/J3
Compatibility			✓			

A layer knows when the cursor rolls into and out of its clipping region. Like several other objects in the document object model, the layer object has `onMouseOver=` and `onMouseOut=` event handlers to let you perform any number of actions in response to those user activities. Typically, a layer's `onMouseOver=` event handler changes colors, hides or shows pseudo-borders devised of colored layers behind the primary layer, or even changes the text or image content. The status bar is also available to plant plain-language legends about the purpose of the layer or offer other relevant help.

Both events occur only once per entrance and egress from a layer's region by the cursor. If you want to script actions dependent upon the location of the cursor in the layer, you can use `layer.captureEvents()` to grab `mousemove` and all types of mouse button events. This kind of event capture generates an event object (see Chapter 33) that includes information about the coordinate position of the cursor at the time of the event.

Related Items: `link.onMouseOut=` event handler; `link.onMouseOver=` event handler; `area.onMouseOut=` event handler; `area.onMouseOver=` event handler.

