CHAPTER 31

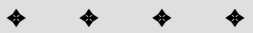
Positioned Objects

In This Chapter

Layer concepts

How to move, hide, and show content

The end of the LAYER element



This is an oddball chapter within the scheme of Part III. Thus far, I have devoted each chapter to a distinct set of object model objects. This chapter breaks away from that mold for just a moment. The main reason that this chapter even exists has to do more with the history of Dynamic HTML — the capability to alter content on the fly in response to user interaction — particularly with respect to Netscape Navigator 4. The impetus for this separate discussion is the NN4 LAYER element and its associated object. What makes this discussion awkward is that the LAYER element and object became dead-end entities that never made it into the W3C standards process. NN6 instead has adopted the W3C standards for dynamic content, which more closely mimic the way Microsoft implemented its DHTML features starting with IE4. NN6 explicitly does not provide backward compatibility with scripted LAYER element objects, which also means that you must rewrite legacy applications to work in NN6.

That leaves an ungainly task in this chapter to create a bridge between the LAYER element and the more modern way of working with elements that can be positioned on the page, flown across the page, stacked in front of other elements, or hidden from view. The IE4+ and NN6 way to accomplish all of this is through CSS style sheets and the scripting thereof. In years to come, the NN4 LAYER element will be only a distant memory. Until then, we must acknowledge it and understand how to work the same magic with style sheets. To that end, this chapter provides details on both the NN4 layer object and the comparable syntax for using IE4+ and NN6 style sheets to get and set properties or invoke methods. Chapter 48 applies these techniques in some DHTML applications.

What Is a Layer?

Terminology in the area of positioned elements has become a bit confusing over time. Because NN4 was the earliest browser to be released with positioned elements (the LAYER element), the term *layer* became synonymous with any positioned element. When IE4 came on the scene, it was convenient to call a style sheet-positioned element (in other words, an element governed by a style sheet rule with the `position` attribute) a *layer* as a generic term for any positioned element. In fact, NN4 even treated an element that was positioned through style sheets as if it were a genuine layer object (although with some minor differences).

In the end, the layer term made good sense because no matter how it was achieved, a positioned element acted like a layer in front of the body content of a page. 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 took another picture.

If you can visualize how that operation works, you have a good starting point for understanding how layers work. Each layer contains some kind of HTML content that exists in its own plane above the main document that loads in a window. You can change or replace the content of an individual layer on the fly without affecting the other layers; you can also reposition, resize, or hide the entire layer 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 around without regard for what's going on outside.

With this analogy in mind, many commercial DHTML development tools and content authors refer to positioned elements as layers, which you can move, resize, stack, and hide independently of the body background. Therefore, references throughout this book to layers may mean anything from the NN4 layer object to an element positioned by way of style sheets.

NN4 Layer Object

Properties	*Methods*	*Event Handlers*
above	captureEvents()	onBlur
background	handleEvent()	onFocus
below	load()	onLoad
bgcolor	moveAbove()	onMouseOut
clip.bottom	moveBelow()	onMouseOver
clip.left	moveBy()	
clip.right	moveTo()	
clip.top	moveToAbsolute()	
document	releaseEvents()	
left	resizeBy()	
name	resizeTo()	
pageX	routeEvent()	
pageY		
parentLayer		
siblingAbove		
siblingBelow		
src		
top		
visibility		
zIndex		

Syntax

Accessing layer object properties or methods:

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

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

About this object

You can create a layer object in NN4 in one of three ways. The first two ways use NN4-only syntax: the `<LAYER>` tag in HTML and the `new Layer()` constructor in JavaScript. The tag offers numerous attributes that establish the location, stacking order, and visibility. These attributes, in turn, become scriptable properties. If you create the layer through the constructor, you then use JavaScript to assign values to the object's properties.

The third way to create an NN4 layer object is to assign an absolute-positioned style sheet rule to a block-level element — most typically a DIV element. This is the way that IE4+ and NN6 do it, too. In practice, however, a positioned DIV element is not as robust (from rendering and scriptability standpoints) in NN4 as a genuine LAYER element. Therefore, it is sometimes necessary to branch a page's code to use `document.write()` for a `<LAYER>` tag in NN4 and a `<DIV>` tag in IE4+ and NN6.

Layer references

The task of assembling JavaScript references to NN4 layers and the objects they contain resembles the same process 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 this:

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

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

```
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>
    <LAYER NAME="Wilma" SRC="wilmaFlintstone.html"></LAYER>
</LAYER>
</BODY>
</HTML>
```

references to items in the second level of layers get even longer. For example, to get the `lastModified` property of the `fredFlintstone.html` file loaded into the nested `Fred` layer, use this reference from the Head script:

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

The reason for this is that NN4 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 18, 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.

Layers and forms

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

As a demonstration of reverse-direction references, I start with the following skeletal structure that contains 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>
    <LAYER NAME="size" SRC="sizeChoice.html"></LAYER>
    <LAYER NAME="sendIt" SRC="submission.html"></LAYER>
</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 uses this 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.

Layers and tables

The document-centered nature of NN4 layers also makes it difficult — if not impossible at times — to incorporate them inside tables. Even defining a layer that is contained by a TD table cell can cause countless problems.

If you need to have absolute-positioned elements that look as though they are part of a table, I suggest you define the layers as freestanding elements outside of the table. After that, you can position the layers to make them look like they live in the table. You may also need to create empty placeholders in your table to make room for the overlaid layer. You can do this by way of a relative-positioned element inside the table cell whose visibility is hidden. This allows the element to flow as the page loads to accommodate the current browser window dimensions. Scripts can then read the location of the relative-positioned element and use those coordinates to move the absolute-positioned elements that are to overlay the hidden elements.

Properties

```
above
below
siblingAbove
siblingBelow
```

Value: Layer object Read-Only

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Each layer object is its own physical layer. Given that the variables x and y traditionally represent width and height, 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 is important for scripting purposes, especially if your script needs to reorder 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, *layerObject*.above and *layerObject*.below, takes a global look at all layers defined on the page regardless of the fact that 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 retrieve properties of a nonexistent layer result in runtime scripting errors indicating that the object does not 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, *layerObject*.siblingAbove and *layerObject*.siblingBelow, confines itself to the group of layers inside a parent layer container. 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 *layerObject*.siblingAbove and *layerObject*.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 be 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.

Positioned objects in IE4+ and NN6 have no comparable properties to the four described in this section.

Example (with Listing 31-1) on the CD-ROM

Related Items: layer.parentLayer property; layer.moveAbove(), layer.moveBelow() methods.

background

Value: Image object Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

You can assign a background image to a layer. The BACKGROUND attribute of the <LAYER> tag usually sets the initial image, but you can assign a new image whenever you like via the *layerObject*.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 not to 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 *layerObject*.background property is an image object (see Chapter 22). To change the image in that property on the fly, you must set the *layerObject*.background.src property to the URL of the desired image (just like changing document.*imageName*.src on the fly). You can remove the background image by setting the *layerObject*.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.

The IE4+ and NN6+ way of handling background images is through the style.backgroundImage property.

Example (with Listing 31-2) on the CD-ROM

Related Items: layer.bgColor property; image object.

bgColor

Value: String Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

A layer's background color fills the entire rectangle with the color set in the <LAYER> tag or from a script at a later time. Color values are the same as for document-related values; they may be in the hexadecimal triplet format or in one of the plain-language color names. You can turn a layer transparent by setting its bgColor property to null.

You control the corresponding behavior in IE4+ and NN6+ via the style.backgroundColor property.

Example (with Listing 31-3) on the CD-ROM

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

clip

Value: String Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

The *`layerObject`*`.clip` property is an object (the only one in NN4'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 NN4. Therefore, you may have to use other means to get that information into your scripts if you need it. (I do it in Listing 31-4.) 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 `<IMG>` 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 you increase the `clip.left` value 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 affect only the right edge; changes to the `clip.height` property affect only the bottom edge. Unfortunately, no shortcuts exist to adjust 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.

IE4+ and NN6+ have the `style.clip` property to assist in adjusting the clipping rectangle of a layer. But the W3C DOM's `style.clip` object does not offer additional subproperties to access individual edges or dimensions of the clipping rectangle. IE5's read-only `currentStyle` object does provide properties for the four edge dimensions. Listing 31-15 demonstrates how to adjust clipping in IE5+ and NN6+ syntax.

Example (with Listing 31-4) on the CD-ROM

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

document

Value: `document` object Read-Only

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Your scripts 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 in a frame.

Related Items: `document` object.

left
top

Value: Integer — Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

The *layerObject*.left and *layerObject*.top properties correspond to the LEFT and TOP attributes of the <LAYER> tag. These integer values determine the horizontal and vertical pixel 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 create 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 *layerObject*.moveTo() method.

IE4+ and NN6+ provide various properties to determine the coordinate location of a positioned element — all through the style object.

Example (with Listing 31-5) on the CD-ROM

Related Items: layer.clip, layer.parentLayer properties.

name

Value: String — Read-Only

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

The *layerObject*.name property reflects the NAME attribute of the <LAYER> tag or name you assign to a positioned DIV or SPAN element. This property is read-only. If

you don't assign a name to a layer when you create it, Navigator generates a name for the layer in this format:

```
js_layer_nn
```

Here, *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: Integer Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

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

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

The situation gets more interesting when you're dealing with nested layers. For a nested layer, the *`layerObject`*`.pageX` and *`layerObject`*`.pageY` values are still measured relative to the page, while *`layerObject`*`.left` and *`layerObject`*`.top` are measured relative to the next higher layer. If trying to conceive of these differences makes your head hurt, the example in Listing 31-6 should help clear things up for you.

Adjusting the *`layerObject`*`.pageX` and *`layerObject`*`.pageY` values of any layer has the same effect as using the *`layerObject`*`.moveToAbsolute()` method, which

measures its coordinate system based on the page. If you create flying layers on your page, you can't go wrong by setting the *layerObject*.pageX and *layerObject*.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 is in the same coordinate system as before the new layer was added.

IE4+ does not provide a pair of properties to determine the location of a positioned element relative to the page, but the offsetLeft and offsetTop properties provide coordinates within the element's next outermost positioning context. Thus, you may have to "walk" the offsetParent trail to accumulate complete coordinate values. In NN6, the offsetLeft and offsetTop properties use the page as the positioning context.

Example (with Listing 31-6) on the CD-ROM

Related Items: layer.left, layer.top, window.innerHeight, window.innerWidth properties; layer.moveToAbsolute() method.

parentLayer

Value: Object Read-Only

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
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 *layerObject*.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 *layerObject*.parentLayer and *layerObject*.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.

Use the *layerObject*.parentLayer property to assemble references to other nested layers. See the discussion about layer references at the beginning of this chapter for several syntax examples.

IE4+ offers an offsetParent property, which comes close to the functionality of the *layerObject*.parentLayer property.

Related Items: layer.above, layer.below properties.

siblingAbove
siblingBelow

See layer.above and layer.below properties earlier in this chapter.

src

Value: String Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
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. This attribute is reflected by the *layerObject*.src property.

The value of this property is a string of the URL of the external file. If you do not specify an SRC attribute 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 it 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 *layerObject*.src property. Again, if a layer has nested layers inside it, those nested layers are blown away by the content that loads into the layer whose src property you change. The new file, of course, can be an HTML file that defines its own nested layers, which then become part of the page's object model.

Netscape also provides the *layerObject*.load() method to insert new content into a layer. One advantage of this method is that an optional second parameter enables you to redefine the width of the layer at the same time 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 are governed as much by their 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 *layerObject*.clip properties. Similarly, longer text content flows beyond the bottom of the previously sized layer unless restricted by clipping properties.

Positioned elements in IE4+ and NN6+ do provide a way to load external content into them. That's what the W3C sees as the purpose of the IFRAME element. Even so, as Listing 31-18 shows, you can script your way around this limitation if it's absolutely necessary.

Example (with Listing 31-7) on the CD-ROM

Related Items: layer.load(), layer.resizeTo() methods.

visibility

Value: String — Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
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 NN4 also enables you to set the property to hidden and visible, which are the values for the style.visibility property used in IE4+ and NN6+.

Unlike many other layer properties, you can set the visibility property 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 includes 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 *layerObject*.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 (with Listing 31-8) on the CD-ROM

Related Items: None.

zIndex

Value: Integer Read/Write

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

Close relationships exist among the *layerObject*.above, *layerObject*.below, and *layerObject*.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 (*layerObject*.above and *layerObject*.below) unless you specify the Z-INDEX attribute; in this 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 *layerObject*.above and *layerObject*.below properties are set as described in their discussion earlier in this chapter. But the *layerObject*.zIndex properties for all layers are zero.

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

Changes to layer*Object*.zIndex values affect the stacking order only of sibling layers. You can assign the same value to two layers, but the last layer to have its *layerObject*.zIndex property set lies physically above the other one. Therefore, if you want to ensure a stacking order, 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 is the same if you assign

them the values of 1, 2, 3 or 10, 13, 50. As you modify a *layerObject*.zIndex value, the *layerObject*.above and *layerObject*.below properties for all affected layers change as a result.

Avoid setting zIndex property values to negative numbers in NN4. Negative values are treated as their absolute (positive) values for ordering.

For IE4+ and NN6+, the style.zIndex property controls z-order.

Example (with Listing 31-9) on the CD-ROM

Related Items: layer.above, layer.below properties; layer.moveAbove(), layer.moveBelow() methods.

Methods

load("*URL*", *newLayerWidth*)

Returns: Nothing.

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

One way to change the content of an NN4 layer after it loads is to use the *layerObject*.load() method. This method has an advantage over setting the *layerObject*.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 hard-wires 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 *layerObject*.resizeTo() method or set the individual *layerObject*.clip properties before loading the new document. This keeps the viewable area of the layer at a fixed size.

IE4+ and NN6 object models don't have a method like this, but you can work around the situation (as shown in Listing 31-18) and then adjust the style.width property of the positioned element.

Example (with Listing 31-10) on the CD-ROM

Related Item: layer.src property.

moveAbove(*layerObject*)
moveBelow(*layerObject*)

Returns: Nothing.

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

With the exception of the *layerObject*.zIndex property, the layer object does not let you adjust properties that affect the global stacking order of layers. The *layerObject*.moveAbove() and *layerObject*.moveBelow() methods enable you to 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; you 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 this statement,

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

the instruction moves the fred layer above the ginger layer. The fred layer can be in any stacking relation to ginger; but, again, both layers must be in the same container.

Obviously, after one of these moves, the *layerObject*.above and *layerObject*.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 are out of order because of user interaction with your scripts, you can reorder them using these methods — or, more practically, by setting their *layerObject*.zIndex properties. In the latter case, it is easier to visualize through your code how the ordering is handled with increasing zIndex values for each layer.

There is no comparable method for IE4+ or NN6.

Example on the CD-ROM

Related Items: layer.above, layer.below, layer.zIndex properties.

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

Returns: Nothing.

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
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 *layerObject*.moveTo() and *layerObject*.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. You can accomplish animation of a layer by issuing a series of *layerObject*.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 *layerObject*.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 *layerObject*.moveToAbsolute() method bypasses the layer's immediate container. The 0,0 point for this method is the top-left corner of the document. Be aware, however, that you can move a layer to a position such that some or all of it lies out of range of the container's clip rectangle.

Moving positioned layers in IE4+ and NN6 requires adjusting the style.left and style.top properties (or the style.pixelLeft, style.pixelTop, style.posLeft, and style.posTop properties in IE4+).

Example (with Listing 31-11) on the CD-ROM

Related Items: layer.resizeBy(), layer.resizeTo(), window.moveBy(), window.moveTo() methods.

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

Returns: Nothing.

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

The basic functionality and parameter requirements of the *layerObject*.resizeBy() and *layerObject*.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 don't adjust the size of the layer. Instead, these methods control the size of the clipping region of the layer. Therefore, the content of the layer does not reflow automatically when you use these methods any more than it does when you change individual *layerObject*.clip values.

Another impact of this clipping region relationship deals 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 afford 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, outsized content doesn'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.

Neither IE4+ nor NN6 provides a similar method, but you can accomplish the same effects by adjusting the style properties of a positioned element.

Example (with Listings 31-12a and 31-12b) on the CD-ROM

Related Items: layer.moveBy(), layer.moveTo(), window.resizeBy(), window.resizeTo() methods.

Event handlers

onBlur
onFocus

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
Compatibility			✓						

A user gets no visual cue 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()`, `textbox.focus()` methods.

onLoad

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
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 way is to specify an `onLoad` event handler in the tag, as shown in Listing 31-12a.

Each time you load a document into the layer — either via the SRC attribute in the <LAYER> tag or by invoking the *layerObject*.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 Item: window.onLoad event handler.

onMouseOut
onMouseOver

	NN2	NN3	NN4	NN6	IE3/J1	IE3/J2	IE4	IE5	IE5.5
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 that enable you to 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; sometimes, it even changes the text or image content. The statusbar 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 to, 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 *layerObject*.captureEvents() to grab mouseMove and all types of mouse button events. This kind of event capture generates an event object (see Chapter 29) that includes information about the coordinate position of the cursor at the time of the event.

Related Items: link.onMouseOut, link.onMouseOver, area.onMouseOut, area.onMouseOver event handlers.

Positioned Elements in the Modern DOM

With the dwindling NN4 installed base, you can focus on applying "layer" techniques in browsers whose object models expose every element of an object and whose rendering engines automatically reflow content in response to changes. This

section follows the sequence of examples in the discussion about NN4's layer object but shows you how to accomplish the same operations and learn the behavior of positioned elements in IE4+ and NN6+.

An important facet that these newer browsers have in common is the `style` property of every renderable element object. Most adjustments to the location, layering, size, and visibility of positioned elements use the `style` object associated with each element. Cross-browser complications ensue, however, with some aspects of nested layers. Plus, there is the ever-present difference between the IE- and NN-class browsers with respect to the event objects — how to reference the `event` object and the names of its properties. Some of the examples that follow have more code in them than their corresponding NN4 layer version shown earlier in this chapter. Most of the additional code concerns itself with accommodating the different event object models.

One more point about the following examples: The syntax adopted for references to element objects uses the W3C DOM `document.getElementById()` method, which is supported in IE5+ and NN6. If you intend to apply any of the techniques in these examples to applications that run exclusively in an IE environment (and must be compatible with IE4), you can substitute the `document.all` referencing syntax. Conversely, you can employ the `document.all` equalization routine shown in Chapter 14 to let IE4+ and NN6 use `document.all` references.

Changing element backgrounds

Listing 31-13 demonstrates the syntax and behavior of setting background images via the `style.backgroundImage` property. Note the CSS-style syntax for the URL value assigned to the `style.backgroundImage` property. It's a good lesson to learn that most `style` properties are strings, and their values are in the same format as the values normally assigned in a style sheet definition.

Removing a background image requires setting the URL to `null`. Also, a background image overlays whatever color (if any) you assign to the element. If the background image has transparent regions, the background color shows through.

Listing 31-13: **Setting Layer Backgrounds (W3C)**

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function setBg(URL) {
    document.getElementById("bgExpo").style.backgroundImage = "url(" + URL + ")"
}
</SCRIPT>
</HEAD>
<BODY>
```

```
<H1>Layer Backgrounds (W3C)</H1>
<HR>
<DIV ID="buttons" STYLE="position:absolute; top:100">
<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')"><P>
<INPUT TYPE="button" VALUE="Remove Image" onClick="setBg(null)"><BR>
</FORM>
</DIV>
<DIV ID="bgExpo" STYLE="position:absolute; top:100; left:250; width:300;
height:260; background-color:gray" >
<SPAN STYLE="font-weight:bold; color:white">Some text, which may or may not read
well with the various backgrounds.</SPAN>
</DIV>
</BODY>
</HTML>
```

Listing 31-14 focuses on background color. A color palette is laid out as a series of rectangles. As the user rolls atop a color in the palette, the color is assigned to the background of the layer. Because of the regularity of the DIV elements generated for the palette, this example uses scripts to dynamically write them to the page as the page loads. This lets the `for` loop handle all the positioning math based on initial values set as global variables.

Perhaps of more interest here than the background color setting is the event handling. First of all, because the target browsers all employ event bubbling, the page lets a single event handler at the document level wait for `mouseover` events to bubble up to the document level. But because the `mouseover` event of every element on the page bubbles there, the event handler must filter the events and process only those on the palette elements.

The `setColor()` method begins by equalizing the IE4+ and NN6 event object models. If an object is assigned to the `evt` parameter variable, then that means the NN6 browser is processing the event; otherwise, it's IE4+ — meaning that the `window.event` object contains the event information. Whichever browser performs the processing, the event object is assigned to the `evt` variable. After verifying that a valid event triggered the function, the next step is to equalize the different, event-model-specific property names for the event's target element. For NN6, the property is `target`, while IE4+ uses `srcElement`. The final validation is to check the `className` property of the event's target element. Because all elements acting as palette colors share the same `CLASS` attribute, the `className` property is examined. If the value is `palette`, then the `mouseover` event has occurred on one of the colors. Now it's time to extract the target element's `style.backgroundColor` property and assign that color to the same property of the main positioned element.

Listing 31-14: **Layer Background Colors (W3C)**

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function setColor(evt) {
    evt = (evt) ? evt : (window.event) ? window.event : ""
    if (evt) {
        var elem = (evt.target) ? evt.target : evt.srcElement
        if (elem.className == "palette") {
            document.getElementById("display").style.backgroundColor =
elem.style.backgroundColor
        }
    }
}
document.onmouseover = setColor
</SCRIPT>
</HEAD>
<BODY>
<H1>Layer Background Colors (W3C)</H1>
<HR>
<SCRIPT LANGUAGE="JavaScript">
var oneLayer
var colorTop = 100
var colorLeft = 20
var colorWidth = 40
var colorHeight = 40
var colorPalette = new Array("aquamarine","coral","forestgreen",
                   "goldenrod","red","magenta","navy","teal")
for (var i = 0; i < colorPalette.length; i++) {
    oneLayer = "<DIV ID='swatch" + i + "' CLASS='palette'"
    oneLayer += "STYLE='position:absolute; top:" + colorTop + ";"
    oneLayer += "left:" + ((colorWidth * i) + colorLeft) + ";"
    oneLayer += "width:" + colorWidth + "; height:" + colorHeight + ";"
    oneLayer += "background-color:" + colorPalette[i] + "'></DIV>\n"
    document.write(oneLayer)
}
</SCRIPT>
<DIV ID="display" STYLE="position:absolute; top:150; left:80; width:200;
height:200; background-color:gray">
<SPAN STYLE="font-weight:bold; color:white; text-align:center">
Some reversed text to test against background colors.</SPAN>
</DIV>
</BODY>
</HTML>
```

Layer clipping

Working with clipping rectangles is a bit cumbersome using CSS syntax because the object model standard does not provide separate readouts or controls over individual edges of a clipping rectangle. IE5+ enables you to read individual edge dimensions via the `currentStyle` object (for example, `currentStyle.clipTop`), but these properties are read-only. NN6 has not connected all the pieces of W3C DOM Level 2 that expose individual edges of a clipping rectangle yet.

Based on these limitations, Listing 31-15 is implemented in a way that, for the sake of convenience, preserves the current clipping rectangle edge values as global variables. Any adjustments to individual edge values are first recorded in those variables (in the `setClip()` function), and then the `style.clip` property is assigned the long string of values in the required format (in the `adjustClip()` function). The `showValues()` function reads the variable values and displays updated values after making the necessary calculations for the width and height of the clipping rectangle.

As a demonstration of a "reveal" visual effect (which you can carry out more simply in IE4+/Windows via a transition filter), the `revealClip()` function establishes beginning clip values at the midpoints of the width and height of the layer. Then the `setInterval()` method loops through `stepClip()` until the clipping rectangle dimensions match those of the layer.

Listing 31-15: **Adjusting Layer clip Properties (W3C)**

```
<HTML>
<HEAD>
<TITLE>Layer Clip</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var origLayerWidth = 0
var origLayerHeight = 0
var currTop, currRight, currBottom, currLeft
function init() {
    origLayerWidth = parseInt(document.getElementById("display").style.width)
    origLayerHeight = parseInt(document.getElementById("display").style.height)
    currTop = 0
    currRight = origLayerWidth
    currBottom = origLayerHeight
    currLeft = 0
    showValues()
}

function setClip(field) {
    var val = parseInt(field.value)
    switch (field.name) {
        case "top" :
            currTop = val
```

Continued

Listing 31-15 *(continued)*

```
            break
        case "right" :
            currRight = val
            break
        case "bottom" :
            currBottom = val
            break
        case "left" :
            currLeft = val
            break
        case "width" :
            currRight = currLeft + val
            break
        case "height" :
            currBottom = currTop + val
            break
    }
    adjustClip()
    showValues()
}

function adjustClip() {
    document.getElementById("display").style.clip = "rect(" + currTop + "px " +
    currRight + "px " + currBottom + "px " + currLeft + "px)"
}

function showValues() {
    var form = document.forms[0]
    form.top.value = currTop
    form.right.value = currRight
    form.bottom.value = currBottom
    form.left.value = currLeft
    form.width.value = currRight - currLeft
    form.height.value = currBottom - currTop
}
var intervalID
function revealClip() {
    var midWidth = Math.round(origLayerWidth /2)
    var midHeight = Math.round(origLayerHeight /2)
    currTop = midHeight
    currBottom = midHeight
    currRight = midWidth
    currLeft = midWidth
    intervalID = setInterval("stepClip()",1)
}
function stepClip() {
    var widthDone = false
    var heightDone = false
    if (currLeft > 0) {
        currLeft += -2
        currRight += 2
```

```
    } else {
        widthDone = true
    }
    if (currTop > 0) {
        currTop += -1
        currBottom += 1
    } else {
        heightDone = true
    }
    adjustClip()
    showValues()
    if (widthDone && heightDone) {
        clearInterval(intervalID)
    }
}
</SCRIPT>
</HEAD>
<BODY onLoad="init()">
<H1>Layer Clipping Properties (W3C)</H1>
<HR>
Enter new clipping values to adjust the visible area of the layer.<P>
<DIV STYLE="position:absolute; top:130">
<FORM>
<TABLE>
<TR>
    <TD ALIGN="right">layer.style.clip (left):</TD>
    <TD><INPUT TYPE="text" NAME="left" SIZE=3 onChange="setClip(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right">layer.style.clip (top):</TD>
    <TD><INPUT TYPE="text" NAME="top" SIZE=3 onChange="setClip(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right">layer.style.clip (right):</TD>
    <TD><INPUT TYPE="text" NAME="right" SIZE=3 onChange="setClip(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right">layer.style.clip (bottom):</TD>
    <TD><INPUT TYPE="text" NAME="bottom" SIZE=3 onChange="setClip(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right">layer.style.clip (width):</TD>
    <TD><INPUT TYPE="text" NAME="width" SIZE=3 onChange="setClip(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right">layer.style.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>
</DIV>
```

Continued

Listing 31-15 *(continued)*

```
<DIV ID="display" STYLE="position:absolute; top:130; left:220; width:360;
height:180; clip:rect(0px 360px 180px 0px); background-color:coral">
<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>
</DIV>
</BODY>
</HTML>
```

Listing 31-16 enables you to compare the results of adjusting a clipping rectangle versus the size of a positioned element. This example goes a bit further than the corresponding NN4 layer version (Listing 31-5) in that it enables you to adjust the dimensions of the entire layer (via the `style.left` and `style.right` properties) as well as the right and bottom edges of the clipping rectangle associated with the layer. As a bonus, the code includes a function that converts the `style.clip` string into an object representing the rectangle of the clipping rectangle (in other words, with four properties, one for each edge). Values from that `rectangle` object populate two of the fields on the page, providing dynamic readouts of the clipping rectangle's right and bottom edges.

Global variables still temporarily store the clipping rectangle values so that the `adjustClip()` function can operate just as it does in Listing 31-15. Note that the clipping rectangle is explicitly defined in the style sheet rule for the positioned element. This is necessary for the element's `style.clip` property to have some values with which to start.

Listing 31-16: Comparison of Layer and Clip Location Properties (W3C)

```
<HTML>
<HEAD>
<TITLE>Layer vs. Clip</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var currClipTop = 0
var currClipLeft = 0
var currClipRight = 360
var currClipBottom = 180
function setClip(field) {
    var val = parseInt(field.value)
```

```
    switch (field.name) {
        case "clipBottom" :
            currClipBottom = val
            break
        case "clipRight" :
            currClipRight = val
            break
    }
    adjustClip()
    showValues()
}
function adjustClip() {
    document.getElementById("display").style.clip = "rect(" + currClipTop +
    "px " + currClipRight + "px " + currClipBottom + "px " + currClipLeft +
    "px)"
}

function setLayer(field) {
    var val = parseInt(field.value)
    switch (field.name) {
        case "width" :
            document.getElementById("display").style.width = val + "px"
            break
        case "height" :
            document.getElementById("display").style.height = val + "px"
            break
    }
    showValues()
}
function showValues() {
    var form = document.forms[0]
    var elem = document.getElementById("display")
    var clipRect = getClipRect(elem)
    form.width.value = parseInt(elem.style.width)
    form.height.value = parseInt(elem.style.height)
    form.clipRight.value = clipRect.right
    form.clipBottom.value = clipRect.bottom
}
// convert clip property string to an object
function getClipRect(elem) {
    var clipString = elem.style.clip
    // assumes "rect(npx, npx, npx, npx)" form
    // get rid of "rect("
    clipString = clipString.replace(/rect\(/,"")
    // get rid of "px)"
    clipString = clipString.replace(/px\)/,"")
    // get rid of remaining "px" strings
    clipString = clipString.replace(/px/g,",")
    // turn remaining string into an array
    clipArray = clipString.split(",")
    // make object out of array values
```

Continued

Listing 31-16 *(continued)*

```
    var clipRect = {top:parseInt(clipArray[0]), right:parseInt(clipArray[1]),
    bottom:parseInt(clipArray[2]), left:parseInt(clipArray[3])}
    return clipRect
}
</SCRIPT>
</HEAD>
<BODY onLoad="showValues()">
<H1>Layer vs. Clip Dimension Properties (W3C)</H1>
<HR>
Enter new layer and clipping values to adjust the layer.<P>
<DIV STYLE="position:absolute; top:130">
<FORM>
<TABLE>
<TR>
    <TD ALIGN="right">layer.style.width:</TD>
    <TD><INPUT TYPE="text" NAME="width" SIZE=3 onChange="setLayer(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right">layer.style.height:</TD>
    <TD><INPUT TYPE="text" NAME="height" SIZE=3 onChange="setLayer(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right">layer.style.clip (right):</TD>
    <TD><INPUT TYPE="text" NAME="clipRight" SIZE=3
onChange="setClip(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right">layer.style.clip (bottom):</TD>
    <TD><INPUT TYPE="text" NAME="clipBottom" SIZE=3
onChange="setClip(this)"></TD>
</TR>
</TABLE>
</FORM>
</DIV>
<DIV ID="display" STYLE="position:absolute; top:130; left:250; width:360;
height:180; clip:rect(0px, 360px, 180px, 0px); background-color:coral">
<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>
</DIV>
</BODY>
</HTML>
```

Scripting nested layers

Working with nested layer locations, especially in a cross-browser manner, presents numerous browser-specific syntax problems that need equalization to behave the same to all users. Some discrepancies even appear between Windows and Macintosh versions of IE.

The scenario for Listing 31-17 consists of one positioned layer (greenish) nested inside another (reddish). The inner layer is initially sized and positioned so that the outer layer extends five pixels in each direction. Text boxes enable you to adjust the coordinates for either layer relative to the entire page as well as the layer's positioning context. If you make a change to any one value, all the others are recalculated and displayed to show you the effect the change has on other coordinate values.

As you see when you load the page, the outer element's positioning context is the page, so the "page" and "container" coordinates are the same (although the calculations to achieve this equality are not so simple across all browsers). The inner layer's initial page coordinates are to the right and down five pixels in each direction, and the coordinates within the container show those five pixels.

Because of browser idiosyncrasies, calculating the coordinates within the page takes the most work. The `getGrossOffsetLeft()` and `getGrossOffsetTop()` functions perform those calculations in the page. Passed a reference to the positioned element to be measured, the first number to grab is whatever the browser returns as the `offsetLeft` or `offsetTop` value of the element (see Chapter 15). These values are independent of the `style` property, and they can report different values for different browsers. IE, for example, measures the offset with respect to whatever it determines as the next outermost positioning context. NN6, on the other hand, treats the page as the positioning context regardless of nesting. So, as long as there is an `offsetParent` element, a `while` loop starts accumulating the `offsetLeft` measures of each succeeding offset parent element going outward from the element. But even before that happens, a correction for IE/Macintosh must be accounted for. If there is a difference between the `style.left` and `offsetLeft` property values of an element, that difference is added to the offset. In IE5/Mac, for example, failure to correct this results in the "page" and "container" values of the outer layer being 10 pixels different in each direction. Values returned from these two gross measures are inserted in the readouts for the "page" measures of both inner and outer elements.

Reading the coordinates relative to each element's "container" is easy: The `style.left` and `style.top` properties have the correct values for all browsers. Moving a layer with respect to its positioning context (the "container" values) is equally easy: assign the entered values to the same `style.left` and `style.top` properties.

Moving the layers with respect to the page coordinate planes (via the `setOuterPage()` and `setInnerPage()` functions) involves going the long way to assign values that

take each browser's positioning idiosyncrasies into account. The way you move a positioned element (cross-browser, anyway) is to assign a value to the `style.left` and `style.top` properties. These values are relative to their positioning context, but NN6 doesn't offer any shortcuts to reveal what element is the positioning context for a nested element. Calls to the `getNetOffsetLeft()` and `getNetOffsetTop()` functions do the inverse of the `getGrossOffsetLeft()` and `getGrossOffsetTop()` functions. Because the values received from the text box are relative to the entire page, the values must have any intervening positioning contexts subtracted from that value in order to achieve the net positioning values that can be applied to the `style.left` and `style.top` properties. To get there, however, a call to the `getParentLayer()` function cuts through the browser-specific implementations of container references to locate the positioning context so that its coordinate values can be subtracted properly. The same kind of correction for IE/Mac is required here as in the gross offset calculations; but here, the correction is subtracted from the value that eventually is returned as the value for either the `style.left` or `style.top` of the layer.

Let me add one quick word about the condition statements of the `while` constructions in the `getNetOffsetLeft()` and `getNetOffsetTop()` functions. You see here a construction not used frequently in this book, but one that is perfectly legal. When the conditional expression evaluates, the `getParentLayer()` method is invoked, and its returned value is assigned to the `elem` variable. That expression evaluates to the value returned by the function. As you can see from the `getParentLayer()` function definition, a value is returned as either an element reference or `null`. The `while` condition treats a value of `null` as `false`; any reference to an object is treated as `true`. Thus, the conditional expression does not use a comparison operator but rather executes some code and branches based on the value returned by that code. NN6 reports JavaScript warnings (not errors) for this construction because it tries to alert you to a common scripting bug that occurs when you use the = operator when you really mean the == operator. But an NN6 warning is not the same as a script error, so don't be concerned when you see these messages in the JavaScript Console window during your debugging.

Listing 31-17: **Testing Nested Layer Coordinate Systems (W3C)**

```
<HTML>
<HEAD>
<TITLE>Nested Layer Coordinates (W3C)</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// offsets within page
function getGrossOffsetLeft(elem) {
    var offset = 0
    while (elem.offsetParent) {
        // correct for IE/Mac discrepancy between offset and style coordinates,
        // but not if the parent is HTML element (NN6)
        offset += (elem.offsetParent.tagName != "HTML") ?
            parseInt(elem.style.left) - parseInt(elem.offsetLeft) : 0
```

```
        elem = elem.offsetParent
        offset += elem.offsetLeft
    }
    return offset
}
function getGrossOffsetTop(elem) {
    var offset = 0
    while (elem.offsetParent) {
        // correct for IE/Mac discrepancy between offset and style coordinates,
        // but not if the parent is HTML element (NN6)
        offset += (elem.offsetParent.tagName != "HTML") ?
            parseInt(elem.style.top) - parseInt(elem.offsetTop) : 0
        elem = elem.offsetParent
        offset += elem.offsetTop
    }
    return offset
}

// offsets within element's positioning context
function getNetOffsetLeft(offset, elem) {
    while (elem = getParentLayer(elem)) {
        // correct for IE/Mac discrepancy between offset and style coordinates,
        // but not if the parent is HTML element (NN6)
        offset -= (elem.offsetParent.tagName != "HTML") ?
            parseInt(elem.style.left) - parseInt(elem.offsetLeft) : 0
        offset -= elem.offsetLeft
    }
    return offset
}
function getNetOffsetTop(offset, elem) {
    while (elem = getParentLayer(elem)) {
        // correct for IE/Mac discrepancy between offset and style coordinates,
        // but not if the parent is HTML element (NN6)
        offset -= (elem.offsetParent.tagName != "HTML") ?
            parseInt(elem.style.top) - parseInt(elem.offsetTop) : 0
        offset -= elem.offsetTop
    }
    return offset
}
// find positioning context parent element
function getParentLayer(elem) {
    if (elem.parentNode) {
        while (elem.parentNode != document.body) {
            elem = elem.parentNode
            while (elem.nodeType != 1) {
                elem = elem.parentNode
            }
            if (elem.style.position == "absolute" || elem.style.position ==
"relative") {
                return elem
            }
```

Continued

Listing 31-17 *(continued)*

```
            elem = elem.parentNode
        }
        return null
    } else if (elem.offsetParent && elem.offsetParent.tagName != "HTML") {
        return elem.offsetParent
    } else {
        return null
    }
}

// functions that respond to changes in text boxes
function setOuterPage(field) {
    var val = parseInt(field.value)
    var elem = document.getElementById("outerDisplay")
    switch (field.name) {
        case "pageX" :
            elem.style.left = ((elem.offsetParent) ? getNetOffsetLeft(val, elem) :
val) + "px"
            break
        case "pageY" :
            elem.style.top = ((elem.offsetParent) ? getNetOffsetTop(val, elem) :
val) + "px"
            break
    }
    showValues()
}
function setOuterLayer(field) {
    var val = parseInt(field.value)
    switch (field.name) {
        case "left" :
            document.getElementById("outerDisplay").style.left = val + "px"
            break
        case "top" :
            document.getElementById("outerDisplay").style.top = val + "px"
            break
    }
    showValues()
}
function setInnerPage(field) {
    var val = parseInt(field.value)
    var elem = document.getElementById("innerDisplay")
    switch (field.name) {
        case "pageX" :
            elem.style.left = ((elem.offsetParent) ? getNetOffsetLeft(val, elem) :
val) + "px"
            break
        case "pageY" :
            elem.style.top = ((elem.offsetParent) ? getNetOffsetTop(val, elem) :
val) + "px"
            break
    }
```

```
    showValues()
}
function setInnerLayer(field) {
    var val = parseInt(field.value)
    switch (field.name) {
        case "left" :
            document.getElementById("innerDisplay").style.left = val + "px"
            break
        case "top" :
            document.getElementById("innerDisplay").style.top = val + "px"
            break
    }
    showValues()
}
function showValues() {
    var form = document.forms[0]
    var outer = document.getElementById("outerDisplay")
    var inner = document.getElementById("innerDisplay")
    form.elements[0].value = outer.offsetLeft +
    ((outer.offsetParent) ? getGrossOffsetLeft(outer) : 0)
    form.elements[1].value = outer.offsetTop +
    ((outer.offsetParent) ? getGrossOffsetTop(outer) : 0)
    form.elements[2].value = parseInt(outer.style.left)
    form.elements[3].value = parseInt(outer.style.top)
    form.elements[4].value = inner.offsetLeft +
    ((inner.offsetParent) ? getGrossOffsetLeft(inner) : 0)
    form.elements[5].value = inner.offsetTop +
    ((inner.offsetParent) ? getGrossOffsetTop(inner) : 0)
    form.elements[6].value = parseInt(inner.style.left)
    form.elements[7].value = parseInt(inner.style.top)
}
</SCRIPT>
</HEAD>
<BODY onLoad="showValues()">
<H1>Nested Layer Coordinates (W3C)</H1>
<HR>
Enter new page and layer coordinates for the <FONT COLOR="coral">outer
layer</FONT> and <FONT COLOR="aquamarine">inner layer</FONT> objects.<P>
<DIV STYLE="position:absolute; top:130">
<FORM>
<TABLE>
<TR>
    <TD ALIGN="right" BGCOLOR="coral">Page X:</TD>
    <TD BGCOLOR="coral"><INPUT TYPE="text" NAME="pageX" SIZE=3
        onChange="setOuterPage(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right" BGCOLOR="coral">Page Y:</TD>
    <TD BGCOLOR="coral"><INPUT TYPE="text" NAME="pageY" SIZE=3
        onChange="setOuterPage(this)"></TD>
</TR>
```

Continued

Listing 31-17 *(continued)*

```
<TR>
    <TD ALIGN="right" BGCOLOR="coral">Container X:</TD>
    <TD BGCOLOR="coral"><INPUT TYPE="text" NAME="left" SIZE=3
        onChange="setOuterLayer(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right" BGCOLOR="coral">Container Y:</TD>
    <TD BGCOLOR="coral"><INPUT TYPE="text" NAME="top" SIZE=3
        onChange="setOuterLayer(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right" BGCOLOR="aquamarine">Page X:</TD>
    <TD BGCOLOR="aquamarine"><INPUT TYPE="text" NAME="pageX" SIZE=3
        onChange="setInnerPage(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right" BGCOLOR="aquamarine">Page Y:</TD>
    <TD BGCOLOR="aquamarine"><INPUT TYPE="text" NAME="pageY" SIZE=3
        onChange="setInnerPage(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right" BGCOLOR="aquamarine">Container X:</TD>
    <TD BGCOLOR="aquamarine"><INPUT TYPE="text" NAME="left" SIZE=3
        onChange="setInnerLayer(this)"></TD>
</TR>
<TR>
    <TD ALIGN="right" BGCOLOR="aquamarine">Container Y:</TD>
    <TD BGCOLOR="aquamarine"><INPUT TYPE="text" NAME="top" SIZE=3
        onChange="setInnerLayer(this)"></TD>
</TR>
</TABLE>
</FORM>
</DIV>
<DIV ID="outerDisplay" STYLE="position:absolute; top:130; left:200; width:370;
height:190; background-color:coral">
<DIV ID="innerDisplay" STYLE="position:absolute; top:5; left:5; width:360;
height:180; background-color:aquamarine" >
<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>
</DIV>
</DIV>
</BODY>
</HTML>
```

Try entering a variety of values in all text boxes to see what happens. Here is one possible sequence of tests and explanations:

1. Increase the red Page X value to 250. This moves the outer layer to the right by 50 pixels. Because the green layer is nested inside, it moves along with it. The green's Page X value also increases by 50, but its Container X value remains the same because the inner layer maintains the same relationship with the outer layer as before.
2. Increase the green Page X value to 300. This action shifts the position of the green inner layer by 45 pixels, making it a total of 50 pixels inset within its positioning context. Because the outer layer does not have its clipping rectangle set, the inner layer's content bleeds beyond the width of the red layer.
3. Set the Container Y value to -50. This action moves the green inner layer upward so that its top is 50 pixels above the top of its red container. As a result, the Page Y value of the inner layer is 80, while the Page Y value of the red outer layer remains at 130 (thus, the 50-pixel difference).

As you experiment with moving the layers around, you may encounter some screen refresh problems where traces of the inner layer remain when moved beyond the outer layer's rectangle. Take these bugs into account when you design the actions of your script-controlled positioning.

Loading external HTML into a layer

The NN4 layer object had an unfair advantage when it came to loading external content into it: the element was designed to do just that, acting in some ways like the W3C-endorsed IFRAME element.

Because the IE4+ and NN6 object models embrace the IFRAME element, using that element may be the easy way for you to designate a space within a page for external content. In fact, you can even assign a style sheet rule that absolute-positions the IFRAME precisely on the page where you want it. Be sure to set the `FRAMEBORDER` attribute to `0` unless you want the border to be visible to the user (and then watch out for content that may overrun the rectangle and cause scrollbars to appear). In this case, you must then leave all the formatting and style sheet control of that content to the HTML loaded into the IFRAME, just as if it were in a separate window or frame. To load different content into the element, assign a different URL to the `src` property of the IFRAME element object.

As one more example that more closely simulates the loading of external content into a layer, Listing 31-18 demonstrates a somewhat ugly workaround that lets a layer's background color or image show through some kinds of HTML content. The technique works only in IE5.5+ and NN6 because these browser generations are the first to offer scripted access to the HTML you need to load into an intermediate (and hidden) IFRAME before stuffing the content into the layer.

A hidden IFRAME element is the initial recipient of the external HTML file, as loaded by the `loadOuter()` method. When that file loads, the `transferHTML()` method is invoked to copy the `innerHTML` of just the BODY element of the content window of the IFRAME (note the different syntax for NN6 — the `contentDocument` property — and IE5.5 — the `contentWindow` property). By eliminating the BODY element and any tags in the HEAD, you prevent the tags in the layer from conflicting with the tags for the main document. As a result, however, notice how the background color set for the layer shows through the HTML plugged into the layer.

HTML element objects (other than IFRAME) were not designed to get their content from external files. But, as Listing 31-18 shows, where there is a will there is a way — even if the workaround isn't pretty.

Listing 31-18: **Setting Layer Source Content (W3C)**

```
<HTML>
<HEAD>
<TITLE>Loading External Content into a Layer (W3C)</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function loadOuter(doc) {
    document.getElementById("hiddenContent").src = doc
    // workaround for missing onLoad event in IFRAME for NN6
    if (!document.getElementById("hiddenContent").onload) {
        setTimeout("transferHTML()", 1000)
    }
}
function transferHTML() {
    var srcFrame = document.getElementById("hiddenContent")
    var srcContent = (srcFrame.contentDocument) ?
    srcFrame.contentDocument.getElementsByTagName("BODY")[0].innerHTML :
    (srcFrame.contentWindow) ?
    srcFrame.contentWindow.document.body.innerHTML : ""
    document.getElementById("outerDisplay").innerHTML = srcContent
}
</SCRIPT>
</HEAD>
<BODY>
<H1>Loading External Content into a Layer (W3C)</H1>
<HR>
<P>Click the buttons to see what happens when you load new source documents into
the <FONT COLOR="coral">layer</FONT> object.</P>
<DIV STYLE="position:absolute; top:150; width:200; background-color: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>
</DIV>
```

```
<DIV ID="outerDisplay" STYLE="position:absolute; top:150; left:250; width:370;
height:190; background-color:coral">
    <P><B>Placeholder text for layer.</B></P>
</DIV>
<IFRAME ID="hiddenContent" STYLE="visibility:hidden"
onLoad="transferHTML()"></IFRAME>
</BODY>
</HTML>
```

Positioned element visibility behavior

There is very little code in Listing 31-19 because it simply adjusts the `style.visibility` property of an outer layer and a nested, inner layer. You can see that when the page loads, the green inner layer's `visibility` is automatically set to inherit the `visibility` of its containing outer layer. When you click the outer layer buttons, the inner layer blindly follows the settings.

Things change, however, once you start adjusting the properties of the inner layer independently of the outer layer. With the outer layer hidden, you can show the inner layer. Only by setting the `visibility` property of the inner layer to `inherit` can you make it rejoin the outer layer in its behavior.

Listing 31-19: Nested Layer Visibility Relationships (W3C)

```
<HTML>
<HEAD>
<TITLE>layer.style.visibility (W3C)</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function setOuterVis(type) {
    document.getElementById("outerDisplay").style.visibility = type
}
function setInnerVis(type) {
    document.getElementById("innerDisplay").style.visibility = type
}
</SCRIPT>
</HEAD>
<BODY>
<H1>Setting the <TT>layer.style.visibility</TT> Property of Nested Layers
(W3C)</H1>
<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>
<DIV STYLE="position:absolute; top:150; width:180; background-color:coral">
<FORM>
Control outer layer property:<BR>
```

Continued

Listing 31-19 *(continued)*

```
<INPUT TYPE="button" VALUE="Hide Outer Layer"
onClick="setOuterVis('hidden')"><BR>
<INPUT TYPE="button" VALUE="Show Outer Layer"
onClick="setOuterVis('visible')"><BR>
</FORM>
</DIV>
<DIV STYLE="position:absolute; top:270; width:180; background-color:aquamarine">
<FORM>
Control inner layer property:<BR>
<INPUT TYPE="button" VALUE="Hide Inner Layer"
onClick="setInnerVis('hidden')"><BR>
<INPUT TYPE="button" VALUE="Show Inner Layer"
onClick="setInnerVis('visible')"><BR>
<INPUT TYPE="button" VALUE="Inherit Outer Layer"
onClick="setInnerVis('inherit')"><BR>
</FORM>
</DIV>
<DIV ID="outerDisplay" STYLE="position:absolute; top:150; left:200; width:370;
height:190; background-color:coral">
    <DIV ID="innerDisplay" STYLE="position:absolute; top:5; left:5; width:360;
height:180; background-color:aquamarine">
    <P><B>Placeholder text for raw inner layer.</B></P>
    </DIV>
</DIV>
</BODY>
</HTML>
```

Scripting layer stacking order

Listing 31-20 is simpler than its NN4 layer-specific version (Listing 31-9) because the W3C DOM, as implemented in IE4+ and NN6, does not have properties that reveal the equivalent of the *layerObject*.`above` or *layerObject*.`below` properties. Therefore, Listing 31-20 confines itself to enabling you to adjust the `style.zIndex` property values of three overlapping layers. All three layers (none of which are nested inside another) initially set their `zIndex` values to `0`, meaning that the source code order rules the stacking order.

If you try this example on both IE4+ and NN6, however, you will experience a significant difference in the behavior of overlapping layers in the two browser categories. For example, if you reload the page to let source code order lay out the layers initially, and then set the green middle layer to, say, 5, the middle layer plants itself in front of the other two in both browser categories. But if you restore the middle layer's `zIndex` value to `0`, IE puts it back in source code order. NN6, on the other hand, leaves it in front of the other two. The rule of thumb (which also applies to NN4) is that if scripts modify the `zIndex` property of multiple layers to all the same value, the most recently set layer stays in front of the others.

There is some method to this seeming madness, which you can experience in Chapter 56's map puzzle game. If you drag one of several draggable elements around the page, you probably will set its `zIndex` to a value higher than that of all the others so that the currently active element stays in front of the rest. But when you complete the dragging, you will want to restore the `zIndex` to its original value, which may be the same as that of all the other draggable items. By keeping the most recently adjusted layer on top, you keep the layer you just dropped in front of the others in case you want to pick it up again.

Listing 31-20: **Relationships Among zIndex Values (W3C)**

```
<HTML>
<HEAD>
<TITLE>layer.style.zIndex</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function setZ(field) {
    switch (field.name) {
        case "top" :
            document.getElementById("topLayer").style.zIndex =
parseInt(field.value)
            break
        case "mid" :
            document.getElementById("middleLayer").style.zIndex =
parseInt(field.value)
            break
        case "bot" :
            document.getElementById("bottomLayer").style.zIndex =
parseInt(field.value)
    }
    showValues()
}
function showValues() {
    var botLayer = document.getElementById("bottomLayer")
    var midLayer = document.getElementById("middleLayer")
    var topLayer = document.getElementById("topLayer")

    document.forms[0].bot.value = botLayer.style.zIndex
    document.forms[1].mid.value = midLayer.style.zIndex
    document.forms[2].top.value = topLayer.style.zIndex
}
</SCRIPT>
</HEAD>
<BODY onLoad="showValues()">
<H1><TT>layer.style.zIndex</TT> Property of Sibling Layers</H1>
<HR>
Enter new zIndex values to see the effect on three layers.<P>
<DIV STYLE="position:absolute; top:140; width:240; background-color:coral">
<FORM>
Control Original Bottom Layer:<BR>
```

Continued

Listing 31-20 *(continued)*

```
<TABLE>
<TR><TD ALIGN="right">Layer zIndex:</TD><TD><INPUT TYPE="text" NAME="bot" SIZE=3
onChange="setZ(this)"></TD></TR>
</TABLE>
</FORM>
</DIV>
<DIV STYLE="position:absolute; top:220; width:240; background-color: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>
</TABLE></FORM>
</DIV>
<DIV STYLE="position:absolute; top:300; width:240; background-color: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>
</TABLE>
</FORM>
</DIV>
<DIV ID="bottomLayer" STYLE="position:absolute; top:140; left:260; width:300;
height:190; z-Index:0; background-color:coral">
    <SPAN><B>Original Bottom Layer</B></SPAN>
</DIV>
<DIV ID="middleLayer" STYLE="position:absolute; top:160; left:280; width:300;
height:190; z-Index:0; background-color:aquamarine">
    <SPAN><B>Original Middle DIV</B></SPAN>
</DIV>
<DIV ID="topLayer" STYLE="position:absolute; top:180; left:300; width:300;
height:190; z-Index:0; background-color:yellow">
    <SPAN><B>Original Top Layer</B></SPAN>
</DIV>
</BODY>
</HTML>
```

Dragging and resizing a layer

Listing 31-21 is an IE4+- and NN6-compatible version of the layer dragging example shown earlier in Listing 31-11. The basic structure is the same, with event handler functions for engaging the drag mode, handling the mouse movement while in drag mode, and releasing the element at the end of the journey.

There is a lot more code in this version for several reasons. The main reason is to accommodate the two event object models in the IE and NN browsers. First of all,

event bubbling is used so that all mouse events are handled at the document level. Thus, all of the event handlers need to equalize the `event` object and event target element, as well as filter events so that the action occurs only when a draggable element (as identified by its `className` property) is the target of the event action.

The toughest job involves the `engage()` function because it must use the two different event and element object models to establish the offset of the `mousedown` event within the draggable element. For IE/Windows, this also means taking the scrolling of the body into account. To get the element to reposition itself with mouse motion, the `dragIt()` function applies browser-specific coordinate values to the `style.left` and `style.top` properties of the draggable element. This function is invoked very frequently in response to the `mousemove` event.

One extra event handler in this version, `onmouseout`, disengages the drag action. This event occurs only if the user moves the cursor faster than the browser can update the position.

Nothing in this example, however, treats the `zIndex` stacking order, which must be addressed if the page contains multiple, draggable items. See the map puzzle game in Chapter 56 for an example of processing multiple, draggable items.

Listing 31-21: **Dragging a Layer (W3C)**

```
<HTML>
<HEAD>
<TITLE>Layer Dragging</TITLE>
<STYLE TYPE="text/css">
.draggable {cursor:hand}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
var engaged = false
var offsetX = 0
var offsetY = 0
function dragIt(evt) {
    evt = (evt) ? evt : (window.event) ? window.event : ""
    var targElem = (evt.target) ? evt.target : evt.srcElement
    if (engaged) {
        if (targElem.className == "draggable") {
            while (targElem.id != "myLayer" && targElem.parentNode) {
                targElem = targElem.parentNode
            }
            if (evt.pageX) {
                targElem.style.left = evt.pageX - offsetX + "px"
                targElem.style.top = evt.pageY - offsetY + "px"
            } else {
                targElem.style.left = evt.clientX - offsetX + "px"
                targElem.style.top = evt.clientY - offsetY + "px"
            }
```

Continued

Listing 31-21 *(continued)*

```
            return false
        }
    }
}
function engage(evt) {
    evt = (evt) ? evt : (window.event) ? window.event : ""
    var targElem = (evt.target) ? evt.target : evt.srcElement
    if (targElem.className == "draggable") {
        while (targElem.id != "myLayer" && targElem.parentNode) {
            targElem = targElem.parentNode
        }
        if (targElem.id == "myLayer") {
            engaged = true
            if (evt.pageX) {
                offsetX = evt.pageX - targElem.offsetLeft
                offsetY = evt.pageY - targElem.offsetTop
            } else {
                offsetX = evt.offsetX - document.body.scrollLeft
                offsetY = evt.offsetY - document.body.scrollTop
                if (navigator.userAgent.indexOf("Win") == -1) {
                    offsetX += document.body.scrollLeft
                    offsetY += document.body.scrollTop
                   }

            }
            return false
        }
    }
}
function release(evt) {
    evt = (evt) ? evt : (window.event) ? window.event : ""
    var targElem = (evt.target) ? evt.target : evt.srcElement
    if (targElem.className == "draggable") {
        while (targElem.id != "myLayer" && targElem.parentNode) {
            targElem = targElem.parentNode
        }
        if (engaged && targElem.id == "myLayer") {
            engaged = false
        }
    }
}
</SCRIPT>
</HEAD>
<BODY>
<H1>Dragging a Layer</H1>
<HR>
<DIV ID="myLayer" CLASS="draggable" STYLE="position:absolute; top:90; left:100;
width:300; height:190; background-color:lightgreen">
    <SPAN CLASS="draggable"><B>Drag me around the window.</B></SPAN>
</LAYER>
```

```
<SCRIPT LANGUAGE="JavaScript">
document.onmousedown = engage
document.onmouseup = release
document.onmousemove = dragIt
document.onmouseout = release
</SCRIPT>
</BODY>
</HTML>
```

The final listing in this section applies many example components used thus far to let scripts control the resizing of a positionable element by dragging the lower-right, 20-pixel region. A lot of the hairy code in the `engage()` function is for determining if the `onmousedown` event occurs in the invisible 20-pixel square.

The `resizeIt()` function of Listing 31-22 resembles the `dragIt()` function of Listing 31-21, but the adjustments are made to the width and height of the positionable element. A fair amount of math determines the width of the element in response to the cursor's instantaneous location and sets the `style.width` and `style.height` properties accordingly.

A user's success with resizing an element this way depends a lot on the browser he or she uses. IE, particularly for Windows, may not redraw the resized element very quickly. In this case, the cursor can easily slip out of the hot spot to end the drag. In other browsers, however, response is very fast, and it's very difficult to have the `onmouseout` event fire the `release()` function.

Listing 31-22: **Resizing a Layer (W3C)**

```
<HTML>
<HEAD>
<TITLE>Layer Resizing</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var engaged = false
var offsetX = 0
var offsetY = 0

function resizeIt(evt) {
    evt = (evt) ? evt : (window.event) ? window.event : ""
    var targElem = (evt.target) ? evt.target : evt.srcElement
    if (targElem.className == "draggable") {
        if (engaged) {
            if (evt.pageX) {
                targElem.style.width = (evt.pageX - targElem.offsetLeft -
offsetX) + "px"
                targElem.style.height = (evt.pageY - targElem.offsetTop -
offsetY) + "px"
```

Continued

Listing 31-22 *(continued)*

```
            } else {
                var elemWidth = evt.clientX - targElem.offsetLeft - offsetX -
                    (parseInt(targElem.style.left) -
parseInt(targElem.offsetLeft))
                var elemHeight = evt.clientY - targElem.offsetTop - offsetY -
                    (parseInt(targElem.style.top) -
parseInt(targElem.offsetTop))
                targElem.style.width = elemWidth + "px"
                targElem.style.height = elemHeight + "px"
            }
        }
    }
}

function engage(evt) {
    evt = (evt) ? evt : (window.event) ? window.event : ""
    var targElem = (evt.target) ? evt.target : evt.srcElement
    if (targElem.className == "draggable") {
       while (targElem.id != "myLayer" && targElem.parentNode) {
           targElem = targElem.parentNode
        }
        if (targElem.id == "myLayer") {
            if (evt.pageX && (evt.pageX > ((parseInt(targElem.style.width) - 20) +
             targElem.offsetLeft)) && (evt.pageY >
             ((parseInt(targElem.style.height) - 20) + targElem.offsetTop)))
{
                offsetX = evt.pageX - parseInt(targElem.style.width) -
targElem.offsetLeft
                offsetY = evt.pageY - parseInt(targElem.style.height) -
targElem.offsetTop
                engaged = true
            } else if ((evt.offsetX > parseInt(targElem.style.width) - 20) &&
(evt.offsetY >
                parseInt(targElem.style.height) - 20)) {
                offsetX = evt.offsetX - parseInt(targElem.style.width) -
document.body.scrollLeft
                offsetY = evt.offsetY - parseInt(targElem.style.height) -
document.body.scrollTop
                engaged = true
                if (navigator.userAgent.indexOf("Win") == -1) {
                    offsetX += document.body.scrollLeft
                    offsetY += document.body.scrollTop
                   }
            }
            return false
        }
    }
}
```

```
function release(evt) {
    evt = (evt) ? evt : (window.event) ? window.event : ""
    var targElem = (evt.target) ? evt.target : evt.srcElement
    if (targElem.className == "draggable") {
        while (targElem.id != "myLayer" && targElem.parentNode) {
            targElem = targElem.parentNode
        }
        if (engaged && targElem.id == "myLayer") {
            engaged = false
        }
    }
}
</SCRIPT>
</HEAD>
<BODY>
<H1>Resizing a Layer (W3C)</H1>
<HR>
<DIV ID="myLayer" CLASS="draggable" STYLE="position:absolute; top:170; left:100;
width:300; height:190; background-color:lightblue">
<SPAN>Here is some content inside the layer. See what happens to it as you
resize the layer via the bottom-right 20-pixel handle.</SPAN>
</DIV>
<SCRIPT LANGUAGE="JavaScript">
document.onmousedown = engage
document.onmouseup = release
document.onmousemove = resizeIt
document.onmouseout = release
</SCRIPT>
</BODY>
</HTML>
```

This chapter only scratches the surface in the kinds of positioned element actions you can control via scripts. You may have seen examples of positioned element scripting at sites around the Web. For example, some pages have subject headers fly into place — even "bounce" around until they settle into position. Or elements can go in circles or spirals to get your attention (or distract you, as the case may be). The authors of those tricks apply formulas from other disciplines (such as games programming) to the `style` object properties of a positioned element.

Sometimes the effects are there just for the sake of looking (at first anyway) cool or because the page author knows how to script those effects. Your chief guide in implementing such features, however, should be whether the scripting genuinely adds value to the content offering. If you don't improve the content by adding a flying doodad or pulsating images, then leave them out. A greater challenge is finding meaningful ways to apply positioning techniques. Done the right way and for the right reason, they can significantly enhance the visitor's enjoyment of your application.

✦ ✦ ✦