CHAPTER 16
Scripting CSS

Cascading Style Sheets (CSS) is a standard for specifying the visual presentation of HTML documents. CSS is intended for use by graphic designers: it allows a designer to precisely specify fonts, colors, margins, indentation, borders, and even the position of document elements. But CSS is also of interest to client-side JavaScript programmers because CSS styles can be scripted. Scripted CSS enables a variety of interesting visual effects: you can create animated transitions where document content "slides in" from the right, for example, or create an expanding and collapsing outline list in which the user can control the amount of information that is displayed. When first introduced, scripted visual effects like these were revolutionary. The JavaScript and CSS techniques that produced them were loosely referred to as Dynamic HTML or DHTML, a term that has since fallen out of favor.

CSS is a complex standard that, at the time of this writing, is undergoing active development. CSS is a book-length topic of its own and complete coverage is well beyond the scope of *this* book.[1] In order to understand CSS scripting, however, it is necessary to be familiar with CSS basics and with the most commonly scripted styles, so this chapter begins with a concise overview of CSS, followed by an explanation of key styles that are most amenable to scripting. After these two introductory sections, the chapter moves on to explain how to script CSS. §16.3 explains the most common and important technique: altering the styles that apply to individual document elements using the HTML `style` attribute. Although an element's `style` attribute can be used to set styles, it is not useful for querying an element's style. §16.4 explains how to query the *computed style* of any element. §16.5 explains how to alter many styles at once by altering the `style` attribute of an element. It is also possible, though less common, to script stylesheets directly, and §16.6 shows how to enable and disable stylesheets, alter the rules of existing stylesheets, and add new stylesheets.

1. But see *CSS: The Definitive Guide* by Eric Meyer (O'Reilly), for example.

16.1 Overview of CSS

There are many variables in the visual display of an HTML document: fonts, colors, spacing, and so on. The CSS standard enumerates these variables and calls them style *properties*. CSS defines properties that specify fonts, colors, margins, borders, background images, text alignment, element size, and element position. To define the visual appearance of HTML elements, we specify the value of CSS properties. To do this, follow the name of a property with a colon and a value:

```
font-weight: bold
```

In order to fully describe the visual presentation of an element, we usually need to specify the value of more than one property. When multiple name:value pairs are required, they are separated from one another by semicolons:

```
margin-left: 10%;      /* left margin is 10% of page width */
text-indent: .5in;     /* indent by 1/2 inch */
font-size: 12pt;       /* 12 point font size */
```

As you can see, CSS ignores comments between `/*` and `*/`. It does not support comments that begin with `//`, however.

There are two ways to associate a set of CSS property values with the HTML elements whose presentation they define. The first is by setting the `style` attribute of an individual HTML element. This is called the inline style:

```
<p style="margin: 20px; border: solid red 2px;">
This paragraph has increased margins and is
surrounded by a rectangular red border.
</p>
```

It is usually much more useful, however, to separate CSS styles from individual HTML elements and define them in a *stylesheet*. A stylesheet associates sets of style properties with sets of HTML elements that are described using *selectors*. A selector specifies or "selects" one or more elements of a document, based on element ID, class, or tag name, or on more specialized criteria. Selectors were introduced in §15.2.5, which also showed how to use `querySelectorAll()` to obtain the set of elements that match the selector.

The basic element of a CSS stylesheet is a style rule, which consists of a selector followed by a set of CSS properties and their values, enclosed in curly braces. A stylesheet can contain any number of style rules:

```
p {                     /* the selector "p" matches all <p> elements */
    text-indent: .5in;  /* indent the first line by .5 inches */
}

.warning {                    /* Any element with class="warning" */
    background-color: yellow; /* gets a yellow background */
    border: solid black 5px;  /* and a big black border */
}
```

A CSS stylesheet can be associated with an HTML document by enclosing it within `<style>` and `</style>` tags within the `<head>` of a document. Like the `<script>` element, the `<style>` element parses its content specially and does not treat it as HTML:

```
<html>
<head><title>Test Document</title>
<style>
body { margin-left: 30px; margin-right: 15px; background-color: #ffffff }
p { font-size: 24px; }
</style>
</head>
<body><p>Testing, testing</p>
</html>
```

When a stylesheet is to be used by more than one page on a website, it is usually better to store it in its own file, without any enclosing HTML tags. This CSS file can then be included in the HTML page. Unlike the `<script>` element, however, the `<style>` element does not have a `src` attribute. To include a stylesheet in an HTML page, use a `<link>` in the `<head>` of a document:

```
<head>
  <title>Test Document</title>
  <link rel="stylesheet" href="mystyles.css" type="text/css">
</head>
```

That, in a nutshell, is how CSS works. There are a few other points about CSS that are worth understanding, however, and the subsections that follow explain them.

16.1.1 The Cascade

Recall that the C in CSS stands for "cascading." This term indicates that the style rules that apply to any given element in a document can come from a "cascade" of different sources:

- The web browser's default stylesheet
- The document's stylesheets
- The `style` attribute of individual HTML elements

Styles from the `style` attribute override styles from stylesheets. And styles from a document's stylesheets override the browser's default styles, of course. The visual presentation of any given element may be a combination of style properties from all three sources. An element may even match more than one selector within a stylesheet, in which case the style properties associated with all of those selectors are applied to the element. (If different selectors define different values for the same style property, the value associated with the most specific selector overrides the value associated with less specific selectors, but the details are beyond the scope of this book.)

To display any document element, the web browser must combine the `style` attribute of that element with styles from all the matched selectors in the document stylesheets. The result of this computation is the actual set of style properties and values that are

used to display the element. This set of values is known as the *computed style* of the element.

16.1.2 CSS History

CSS is a relatively old standard. CSS1 was adopted in December 1996 and defines properties for specifying colors, fonts, margins, borders, and other basic styles. Browsers as old as Netscape 4 and Internet Explorer 4 include substantial support for CSS1. The second edition of the standard, CSS2, was adopted in May 1998; it defines a number of more advanced features, most notably support for absolute positioning of elements. CSS2.1 clarifies and corrects CSS2, and it removes features that browser vendors never actually implemented. Current browsers have essentially complete support for CSS2.1, although versions of IE prior to IE8 have notable omissions.

Work continues on CSS. For version 3, the CSS specification has been broken into various specialized modules that are going through the standardization process separately. You can find the CSS specifications and working drafts at *http://www.w3.org/Style/CSS/current-work*.

16.1.3 Shortcut Properties

Certain style properties that are commonly used together can be combined using special shortcut properties. For example, the `font-family`, `font-size`, `font-style`, and `font-weight` properties can all be set at once using a single `font` property with a compound value:

```
font: bold italic 24pt helvetica;
```

Similarly, the `border`, `margin`, and `padding` properties are shortcuts for properties that specify borders, margins, and padding (space between the border and element content) for each of the individual sides of an element. For example, instead of using the `border` property, you can use `border-left`, `border-right`, `border-top`, and `border-bottom` properties to specify the border of each side separately. In fact, each of these properties is itself a shortcut. Instead of specifying `border-top`, you can specify `border-top-color`, `border-top-style`, and `border-top-width`.

16.1.4 Nonstandard Properties

When browser vendors implement nonstandard CSS properties, they prefix the property names with a vendor-specific string. Firefox uses `moz-`, Chrome uses `-webkit-`, and IE uses `-ms-`. Browser vendors do this even when implementing properties that are intended for future standardization. One example is the `border-radius` property, which specifies rounded corners for elements. This was implemented experimentally in Firefox 3 and Safari 4 using prefixes. Once the standard had matured sufficiently, Firefox 4 and Safari 5 removed the prefix and supported `border-radius` directly. (Chrome and Opera have supported it for a long time with no prefix. IE9 also supports it without a prefix, but IE8 did not support it, even with a prefix.)

When working with CSS properties that have different names in different browsers, you may find it helpful to define a class for that property:

```
.radius10 {
  border-radius: 10px;          /* for current browsers */
  -moz-border-radius: 10px;     /* for Firefox 3.x */
  -webkit-border-radius: 10px;  /* For Safari 3.2 and 4 */
}
```

With a class like this defined, you can add "radius10" to the `class` attribute of any element to give it a border radius of 10 pixels.

16.1.5 CSS Example

Example 16-1 is an HTML file that defines and uses a stylesheet. It demonstrates tag name, class, and ID-based selectors, and also has an example of an inline style defined with the `style` attribute. Figure 16-1 shows how this example is rendered in a browser.

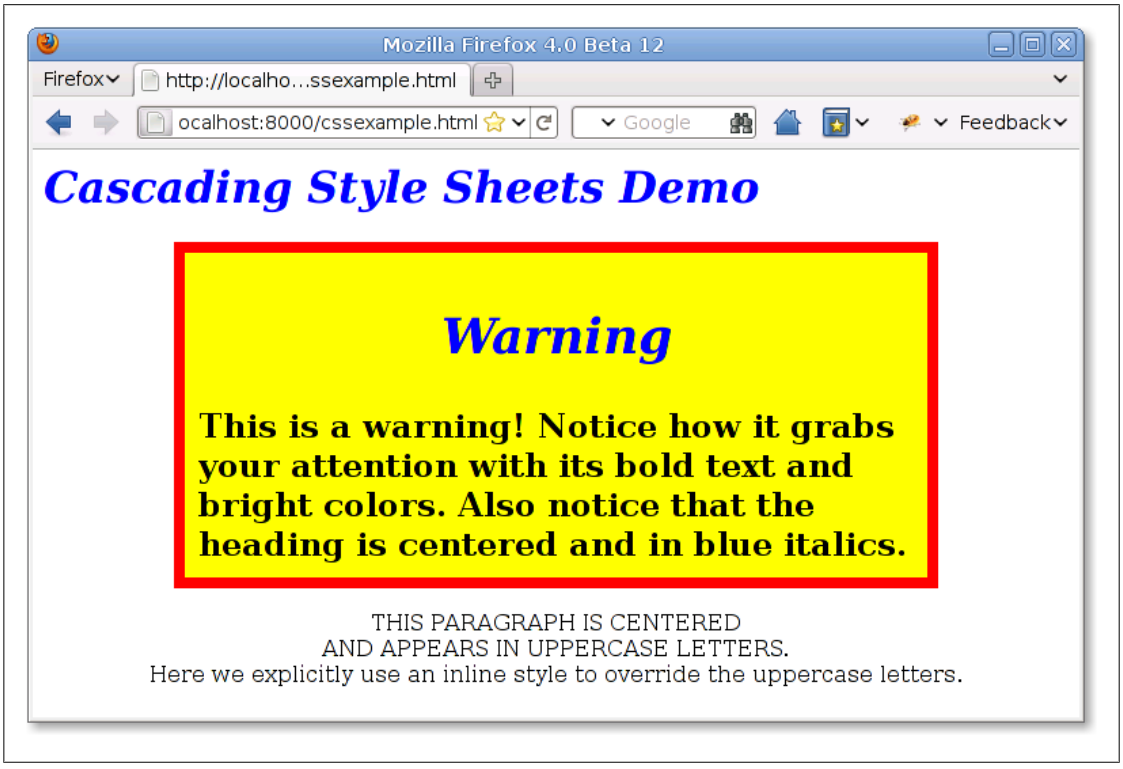

Figure 16-1. A web page styled with CSS

Example 16-1. Defining and using Cascading Style Sheets

```
<head>
<style type="text/css">
/* Specify that headings display in blue italic text. */
h1, h2 { color: blue; font-style: italic }

/*
```

```
 * Any element of class="WARNING" displays in big bold text with large margins
 * and a yellow background with a fat red border.
 */
.WARNING {
          font-weight: bold;
          font-size: 150%;
          margin: 0 1in 0 1in; /* top right bottom left */
          background-color: yellow;
          border: solid red 8px;
          padding: 10px;       /* 10 pixels on all 4 sides */
}

/*
 * Text within an h1 or h2 heading within an element with class="WARNING"
 * should be centered, in addition to appearing in blue italics.
 */
.WARNING h1, .WARNING h2 { text-align: center }

/* The single element with id="special" displays in centered uppercase. */
#special {
      text-align: center;
      text-transform: uppercase;
}
</style>
</head>
<body>
<h1>Cascading Style Sheets Demo</h1>

<div class="WARNING">
<h2>Warning</h2>
This is a warning!
Notice how it grabs your attention with its bold text and bright colors.
Also notice that the heading is centered and in blue italics.
</div>

<p id="special">
This paragraph is centered<br>
and appears in uppercase letters.<br>
<span style="text-transform: none">
Here we explicitly use an inline style to override the uppercase letters.
</span>
</p>
```

Cutting-Edge CSS

As I write this chapter, CSS is in the midst of a revolution, with browser vendors implementing powerful new styles such as `border-radius`, `text-shadow`, `box-shadow`, and `column-count`. Another revolutionary new feature of CSS is *web fonts*: the ability to download and use custom fonts with a CSS `@font-face` rule. (See *http://code.google.com/webfonts* for a selection of fonts free for use on the Web and an easy mechanism for downloading them from Google's servers.)

Another revolutionary development in CSS is CSS Transitions. This is a draft specification that can automatically turn any scripted change to a CSS style into a smoothly

animated transition. (When widely implemented, it will largely obviate the need for CSS animation code like that shown in §16.3.1.) CSS Transitions is implemented in current browsers other than IE, but its style properties are still prefixed with vendor-specific strings. CSS Animations is a related proposal that uses CSS Transitions as a starting point for defining more complex animation sequences. CSS Animations are currently only implemented by Webkit browsers. Neither transitions nor animations are covered in this chapter, but they are technologies that you, as a web developer, should be aware of.

Another CSS draft that web developers should be aware of is CSS Transforms, which allows arbitrary 2D affine transforms (rotation, scaling, translation, or any combination expressed as a matrix) to be applied to any element. All current browsers (including IE9 and later) support the draft, with vendor prefixes. Safari even supports an extension that allows 3D transformations, but it is unclear whether other browsers will follow their lead.

16.2 Important CSS Properties

For client-side JavaScript programmers, the most important features of CSS are the properties that specify the visibility, size, and precise position of individual elements of a document. Other CSS properties allow you to specify stacking order, transparency, clipping region, margins, padding, borders, and colors. In order to script CSS, it is important to understand how these style properties work. They are summarized in Table 16-1 and documented in more detail in the sections that follow.

Table 16-1. Important CSS style properties

Property	Description
`position`	Specifies the type of positioning applied to an element
`top`, `left`	Specify the position of the top and left edges of an element
`bottom`, `right`	Specify the position of the bottom and right edges of an element
`width`, `height`	Specify the size of an element
`z-index`	Specifies the "stacking order" of an element relative to any overlapping elements; defines a third dimension of element positioning
`display`	Specifies how and whether an element is displayed
`visibility`	Specifies whether an element is visible
`clip`	Defines a "clipping region" for an element; only portions of the element within this region are displayed
`overflow`	Specifies what to do if an element is bigger than the space allotted for it
`margin`, `border`, `padding`	Specify spacing and borders for an element.
`background`	Specifies the background color or image of an element.
`opacity`	Specifies how opaque (or translucent) an element is. This is a CSS3 property, supported by some browsers. A working alternative exists for IE.

16.2.1 Positioning Elements with CSS

The CSS `position` property specifies the type of positioning applied to an element. Here are the four possible values for this property:

`static`
: This is the default value and specifies that the element is positioned according to the normal flow of document content (for most Western languages, this is left to right and top to bottom). Statically positioned elements cannot be positioned with `top`, `left`, and other properties. To use CSS positioning techniques with a document element, you must first set its `position` property to one of the other three values.

`absolute`
: This value allows you to specify the position of an element relative to its containing element. Absolutely positioned elements are positioned independently of all other elements and are not part of the flow of statically positioned elements. An absolutely positioned element is positioned either relative to its nearest positioned ancestor or relative to the document itself.

`fixed`
: This value allows you to specify an element's position with respect to the browser window. Elements with `fixed` positioning are always visible and do not scroll with the rest of the document. Like absolutely positioned elements, fixed-position elements are independent of all others and are not part of the document flow. Fixed positioning is supported in most modern browsers but is not available in IE6.

`relative`
: When the `position` property is set to `relative`, an element is laid out according to the normal flow, and its position is then adjusted relative to its position in the normal flow. The space allocated for the element in the normal document flow remains allocated for it, and the elements on either side of it do not close up to fill in that space, nor are they "pushed away" from the new position of the element.

Once you have set the `position` property of an element to something other than `static`, you can specify the position of that element with some combination of the `left`, `top`, `right`, and `bottom` properties. The most common positioning technique uses the `left` and `top` properties to specify the distance from the left edge of the containing element (usually the document itself) to the left edge of the element and the distance from the top edge of the container to the top edge of the element. For example, to place an element 100 pixels from the left and 100 pixels from the top of the document, you can specify CSS styles in a `style` attribute as follows:

```
<div style="position: absolute; left: 100px; top: 100px;">
```

If an element uses absolute positioning, its `top` and `left` properties are interpreted relative to the closest ancestor element that has its `position` property set to something other than `static`. If an absolutely positioned element has no positioned ancestor, the `top` and `left` properties are measured in document coordinates—they are offsets from

the top-left corner of the document. If you wish to absolutely position an element relative to a container that is part of the normal document flow, use `position:relative` for the container and specify a `top` and `left` position of `0px`. This makes the container dynamically positioned but leaves it at its normal place in the document flow. Any absolutely positioned children are then positioned relative to the container position.

Although it is most common to specify the position of the upper-left corner of an element with `left` and `top`, you can also use `right` and `bottom` to specify the position of the bottom and right edges of an element relative to the bottom and right edges of the containing element. For example, to position an element so that its bottom-right corner is at the bottom-right of the document (assuming it is not nested within another dynamic element), use the following styles:

```
position: absolute; right: 0px; bottom: 0px;
```

To position an element so that its top edge is 10 pixels from the top of the window and its right edge is 10 pixels from the right of the window, and so that it does not scroll with the document, you might use these styles:

```
position: fixed; right: 10px; top: 10px;
```

In addition to the position of elements, CSS allows you to specify their size. This is most commonly done by providing values for the `width` and `height` style properties. For example, the following HTML creates an absolutely positioned element with no content. Its `width`, `height`, and `background-color` properties make it appear as a small blue square:

```
<div style="position: absolute; top: 10px; left: 10px;
            width: 10px; height: 10px; background-color: blue">
</div>
```

Another way to specify the width of an element is to specify a value for both the `left` and `right` properties. Similarly, you can specify the height of an element by specifying both `top` and `bottom`. If you specify a value for `left`, `right`, and `width`, however, the `width` property overrides the `right` property; if the height of an element is overconstrained, `height` takes priority over `bottom`.

Bear in mind that it is not necessary to specify the size of every dynamic element. Some elements, such as images, have an intrinsic size. Furthermore, for dynamic elements that contain text or other flowed content, it is often sufficient to specify the desired width of the element and allow the height to be determined automatically by the layout of the element's content.

CSS requires position and dimension properties to be specified with a unit. In the examples above, the position and size properties were specified with the suffix "px," which stands for pixels. You can also use inches ("in"), centimeters ("cm"), points ("pt"), and ems ("em"; a measure of the line height for the current font).

Instead of specifying absolute positions and sizes using the units shown above, CSS also allows you to specify the position and size of an element as a percentage of the size of the containing element. For example, the following HTML creates an empty element with a black border that is half as wide and half as high as the containing element (or the browser window) and centered within that element:

```
<div style="position: absolute; left: 25%; top: 25%; width: 50%; height: 50%;
            border: 2px solid black">
</div>
```

16.2.1.1 The third dimension: z-index

You've seen that the `left`, `top`, `right`, and `bottom` properties can specify the X and Y coordinates of an element within the two-dimensional plane of the containing element. The `z-index` property defines a kind of third dimension: it allows you to specify the stacking order of elements and indicate which of two or more overlapping elements is drawn on top of the others. The `z-index` property is an integer. The default value is zero, but you may specify positive or negative values. When two or more elements overlap, they are drawn in order from lowest to highest `z-index`; the element with the highest `z-index` appears on top of all the others. If overlapping elements have the same `z-index`, they are drawn in the order in which they appear in the document so that the last overlapping element appears on top.

Note that `z-index` stacking applies only to sibling elements (i.e., elements that are children of the same container). If two elements that are not siblings overlap, setting their individual `z-index` properties does not allow you to specify which one is on top. Instead, you must specify the `z-index` property for the two sibling containers of the two overlapping elements.

Nonpositioned elements (i.e., elements with default `position:static` positioning) are always laid out in a way that prevents overlaps, so the `z-index` property does not apply to them. Nevertheless, they have a default `z-index` of zero, which means that positioned elements with a positive `z-index` appear on top of the normal document flow and positioned elements with a negative `z-index` appear beneath the normal document flow.

16.2.1.2 CSS positioning example: Shadowed text

The CSS3 specification includes a `text-shadow` property to produce drop-shadow effects under text. It is supported by a number of current browsers, but you can use CSS positioning properties to achieve a similar effect, as long as you are willing to repeat and restyle the text to produce a shadow:

```
<!-- The text-shadow property produces shadows automatically -->
<span style="text-shadow: 3px 3px 1px #888">Shadowed</span>

<!-- Here's how we can produce a similar effect with positioning. -->
<span style="position:relative;">
  Shadowed     <!-- This is the text that casts the shadow. -->
```

```
    <span style="position:absolute; top:3px; left:3px; z-index:-1; color: #888">
      Shadowed   <!-- This is the shadow -->
    </span>
  </span>
```

The text to be shadowed is enclosed in a relatively positioned `<span>` element. There are no position properties set, so the text appears at its normal position in the flow. The shadow is in an absolutely positioned `<span>` inside (and therefore positioned relatively to) the relatively positioned `<span>`. The `z-index` property ensures that the shadow appears underneath the text that produces it.

16.2.2 Borders, Margins and Padding

CSS allows you to specify borders, margins, and padding around any element. The border of an element is a rectangle (or rounded rectangle in CSS3) drawn around (or partially around) it. CSS properties allow you to specify the style, color, and thickness of the border:

```
border: solid black 1px; /* border is drawn with a solid, black 1-pixel line */
border: 3px dotted red;  /* border is drawn in 3-pixel red dots */
```

It is possible to specify the border width, style, and color using individual CSS properties, and it is also possible to specify the border for individual sides of an element. To draw a line beneath an element, for example, simply specify its `border-bottom` property. It is even possible to specify the width, style, or color of a single side of an element with properties such as `border-top-width` and `border-left-color`.

In CSS3, you can round all corners of a border with the `border-radius` property, and you can round individual corners with more explicit property names. For example:

```
border-top-right-radius: 50px;
```

The `margin` and `padding` properties both specify blank space around an element. The important difference is that `margin` specifies space outside the border, between the border and adjacent elements, and `padding` specifies space inside the border, between the border and the element content. A margin provides visual space between a (possibly bordered) element and its neighbors in the normal document flow. Padding keeps element content visually separated from its border. If an element has no border, padding is typically not necessary. If an element is dynamically positioned, it is not part of the normal document flow, and its margins are irrelevant.

You can specify the margin and padding of an element with the `margin` and `padding` properties:

```
margin: 5px; padding: 5px;
```

You can also specify margins and paddings for individual sides of an element:

```
margin-left: 25px;
padding-bottom: 5px;
```

Or you can specify margin and padding values for all four edges of an element with the `margin` and `padding` properties. You specify the top values first and then proceed

clockwise: top, right, bottom, and left. For example, the following code shows two equivalent ways to set different padding values for each of the four sides of an element:

```
padding: 1px 2px 3px 4px;
/* The previous line is equivalent to the following lines. */
padding-top: 1px;
padding-right: 2px;
padding-bottom: 3px;
padding-left: 4px;
```

The `margin` property works in the same way.

16.2.3 The CSS Box Model and Positioning Details

The `margin`, `border`, and `padding` style properties described above are not properties that you are likely to script very frequently. The reason that they are mentioned here is that margins, borders, and padding are part of the CSS *box model*, and you have to understand the box model in order to truly understand the CSS positioning properties.

Figure 16-2 illustrates the CSS box model and visually explains the meaning of the `top`, `left`, `width`, and `height` for elements that have borders and padding.

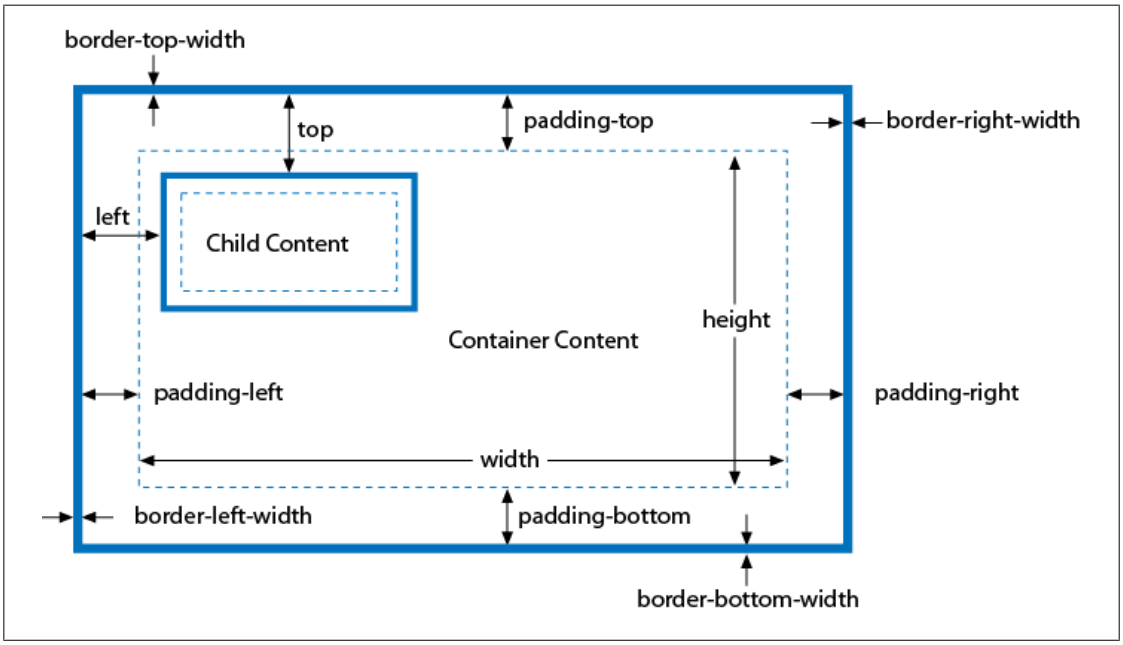

Figure 16-2. The CSS box model: borders, padding, and positioning properties

Figure 16-2 shows an absolutely positioned element nested inside a positioned container element. Both the container and the contained elements have borders and padding, and the figure illustrates the CSS properties that specify padding and border width for each side of the container element. Notice that no margin properties are shown: margins aren't relevant to absolutely positioned elements.

Figure 16-2 contains other, more important information as well. First, `width` and `height` specify the size of an element's content area only; they do not include any

additional space required for the element's padding or border (or margins). To determine the full on-screen size of an element with a border, you must add the left and right padding and left and right border widths to the element width, and you must add the top and bottom padding and top and bottom border widths to the element's height.

Second, the `left` and `top` properties specify the distance from the inside of the container's border to the outside of the positioned element's border. These properties do not measure from the upper-left corner of the content area of the container, but from the upper-left corner of the container's padding. Similarly, the `right` and `bottom` properties measure from the lower-right corner of the padding.

Here's an example that might make this clearer. Suppose you've created a dynamically positioned container element that has 10 pixels of padding all the way around its content area and a 5-pixel border all the way around the padding. Now suppose you dynamically position a child element inside this container. If you set the `left` property of the child to "0 px", you'll discover that the child is positioned with its left edge right up against the inner edge of the container's border. With this setting, the child overlaps the container's padding, which presumably was supposed to remain empty (since that is the purpose of padding). If you want to position the child element in the upper left corner of the container's content area, you should set both the `left` and `top` properties to "10px".

16.2.3.1 The border-box model and the box-sizing property

The standard CSS box model specifies that the `width` and `height` style properties give the size of the content area and do not include padding and borders. We might call this box model the "content-box model." There are exceptions to the content-box model in old versions of IE and also in new versions of CSS. Before IE6, and when IE6, 7, or 8 displays a page in "quirks mode" (when the page has no `<!DOCTYPE>` or has an insufficiently strict doctype), the `width` and `height` properties do include the padding and border widths.

IE's behavior is a bug, but IE's nonstandard box model is often quite useful. Recognizing this, CSS3 introduces a `box-sizing` property. The default value is `content-box`, which specifies the standard box model described above. If you instead specify `box-sizing:border-box`, the browser will use the IE box model for that element, and the `width` and `height` properties will include border and padding. The `border-box` model is particularly useful when you want to specify the overall size of an element as a percentage but also want to specify the border and padding size in pixels:

```
<div style="box-sizing:border-box; width: 50%;
            padding: 10px; border: solid black 2px;">
```

The box-sizing property is supported by all current browsers but is not yet universally implemented without a prefix. In Chrome and Safari, use -webkit-box-sizing. In Firefox, use -moz-box-sizing. In Opera and IE8 and later, you can use box-sizing without any prefix.

A future CSS3 alternative to the border-box model is the use of calculated values for box dimensions:

```
<div style="width: calc(50%-12px); padding: 10px; border: solid black 2px;">
```

Calculated CSS values with calc() are supported in IE9 and in Firefox 4 (as -moz-calc()).

16.2.4 Element Display and Visibility

Two CSS properties affect the visibility of a document element: visibility and display . The visibility property is simple: when the property is set to the value hidden, the element is not shown; when it is set to the value visible, the element is shown. The display property is more general and is used to specify the type of display an item receives. It specifies whether an element is a block element, an inline element, a list item, or so on. When display is set to none, however, the affected element is not displayed, or even laid out, at all.

The difference between the visibility and display style properties has to do with their effect on elements that use static or relative positioning. For an element that appears in the normal layout flow, setting visibility to hidden makes the element invisible but reserves space for it in the document layout. Such an element can be repeatedly hidden and shown without changing the document layout. If an element's display property is set to none, however, no space is allocated for it in the document layout; elements on either side of it close up as if it were not there. The display property is useful, for example, when creating expanding and collapsing outlines.

visibility and display have equivalent effects when used with absolute- or fixed-position elements because these elements are not part of the document layout. The visibility property is generally preferred for hiding and showing positioned elements, however.

Note that it doesn't make much sense to use visibility or display to make an element invisible unless you are going to use JavaScript to dynamically set these properties and make the element visible at some point! We'll see how to do this later in the chapter.

16.2.5 Color, Transparency, and Translucency

You can specify the color of text contained in a document element with the CSS color property. And you can specify the background color of any element with the background-color property. Earlier, we saw that you can specify the color of an element's border with border-color or with the shortcut property border.

The discussion of borders included examples that specified border colors using the English names of common colors such as "red" and "black". CSS supports a number of these English color names, but the more general syntax for specifying colors in CSS is to use hexadecimal digits to specify the red, green, and blue components of a color. You can use either one or two digits per component. For example:

```
#000000       /* black */
#fff          /* white */
#f00          /* bright red */
#404080       /* dark unsaturated blue */
#ccc          /* light gray */
```

CSS3 also defines syntaxes for specifying colors in the RGBA color space (red, green, and blue values plus an *alpha* value that specifies the transparency of the color). RGBA is supported by all modern browsers except IE, and support is expected in IE9. CSS3 also defines support for HSL (hue-saturation-value) and HSLA color specifications. Again, these are supported by Firefox, Safari, and Chrome, but not IE.

CSS allows you to specify the exact position, size, background color, and border color of elements; this gives you a rudimentary graphics capability for drawing rectangles and (when the height and width are reduced) horizontal and vertical lines. The last edition of this book included a bar chart example using CSS graphics, but it has been replaced in this book by extended coverage of the `<canvas>` element. (See Chapter 21 for more on scripted client-side graphics.)

In addition to the `background-color` property, you can also specify images to be used as the background of an element. The `background-image` property specifies the image to use, and the `background-attachment`, `background-position`, and `background-repeat` properties specify further details about how this image is drawn. The shortcut property `background` allows you to specify these properties together. You can use these background image properties to create interesting visual effects, but those are beyond the scope of this book.

It is important to understand that if you do not specify a background color or image for an element, that element's background is usually transparent. For example, if you absolutely position a `<div>` over some existing text in the normal document flow, that text will, by default, show through the `<div>` element. If the `<div>` contains its own text, the letters may overlap and become an illegible jumble. Not all elements are transparent by default, however. Form elements don't look right with a transparent background, for example, and elements such as `<button>` have a default background color. You can override this default with the `background-color` property, and you can even explicitly set it to "transparent" if you desire.

The transparency we've been discussing so far is all-or-none: an element either has a transparent background or an opaque background. It is also possible to specify that an element (both its background and its foreground content) is translucent. (See Figure 16-3 for an example.) You do this with the CSS3 `opacity` property. The value of this property is a number between 0 and 1, where 1 means 100 percent opaque (the

default) and 0 means 0% opaque (or 100% transparent). The `opacity` property is supported by all current browsers except IE. IE provides a work-alike alternative through its IE-specific `filter` property. To make an element 75 percent opaque, you can use the following CSS styles:

```
opacity: .75;              /* standard CSS3 style for transparency */
filter: alpha(opacity=75); /* transparency for IE; note no decimal point */
```

16.2.6 Partial Visibility: overflow and clip

The `visibility` property allows you to completely hide a document element. The `overflow` and `clip` properties allow you to display only part of an element. The `overflow` property specifies what happens when the content of an element exceeds the size specified (with the `width` and `height` style properties, for example) for the element. The allowed values and their meanings for this property are as follows:

`visible`
: Content may overflow and be drawn outside of the element's box if necessary. This is the default.

`hidden`
: Content that overflows is clipped and hidden so that no content is ever drawn outside the region defined by the size and positioning properties.

`scroll`
: The element's box has permanent horizontal and vertical scrollbars. If the content exceeds the size of the box, the scrollbars allow the user to scroll to view the extra content. This value is honored only when the document is displayed on a computer screen; when the document is printed on paper, for example, scrollbars obviously do not make sense.

`auto`
: Scrollbars are displayed only when content exceeds the element's size rather than being permanently displayed.

While the `overflow` property allows you to specify what happens when an element's content is bigger than the element's box, the `clip` property allows you to specify exactly which portion of an element should be displayed, whether or not the element overflows. This property is especially useful for scripted effects in which an element is progressively displayed or uncovered.

The value of the `clip` property specifies the clipping region for the element. In CSS2, clipping regions are rectangular, but the syntax of the `clip` property leaves open the possibility that future versions of the standard will support clipping shapes other than rectangles. The syntax of the `clip` property is:

```
rect(top right bottom left)
```

The *`top`*, *`right`*, *`bottom`*, and *`left`* values specify the boundaries of the clipping rectangle relative to the upper-left corner of the element's box. For example, to display only a 100 × 100-pixel portion of an element, you can give that element this `style` attribute:

```
style="clip: rect(0px 100px 100px 0px);"
```

Note that the four values within the parentheses are length values and must include a unit specification, such as `px` for pixels. Percentages are not allowed. Values can be negative to specify that the clipping region extends beyond the box specified for the element. You can also use the `auto` keyword for any of the four values to specify that the edge of the clipping region is the same as the corresponding edge of the element's box. For example, you can display just the leftmost 100 pixels of an element with this `style` attribute:

```
style="clip: rect(auto 100px auto auto);"
```

Note that there are no commas between the values, and the edges of the clipping region are specified in clockwise order from the top edge. To turn clipping off, set the `clip` property to `auto`.

16.2.7 Example: Overlapping Translucent Windows

This section concludes with an example that demonstrates many of the CSS properties discussed here. Example 16-2 uses CSS to create the visual effect of scrolling, overlapping, translucent windows within the browser window. Figure 16-3 shows how it looks.

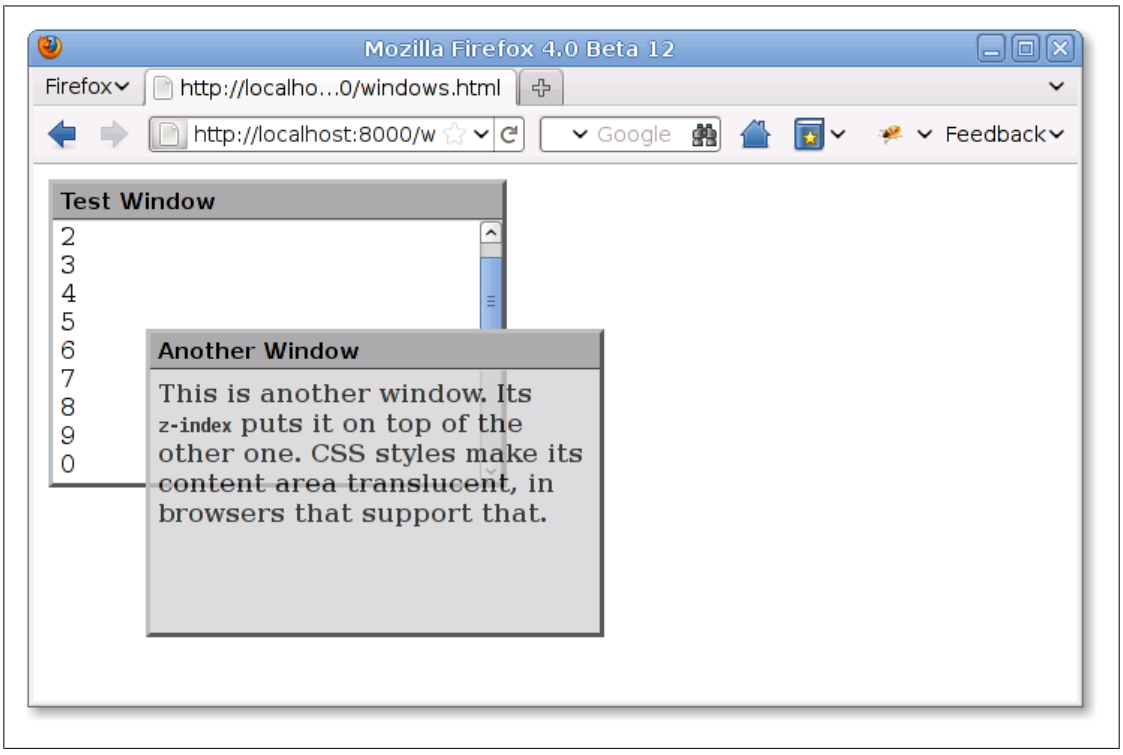

Figure 16-3. Windows created with CSS

The example contains no JavaScript code and no event handlers, so there is no way to interact with the windows (other than to scroll them), but it is a useful demonstration of the powerful effects that can be achieved with CSS.

Example 16-2. Displaying windows with CSS

```
<!DOCTYPE html>
<head>
<style type="text/css">
/**
 * This is a CSS stylesheet that defines three style rules that we use
 * in the body of the document to create a "window" visual effect.
 * The rules use positioning properties to set the overall size of the window
 * and the position of its components. Changing the size of the window
 * requires careful changes to positioning properties in all three rules.
 **/
div.window {  /* Specifies size and border of the window */
    position: absolute;         /* The position is specified elsewhere */
    width: 300px; height: 200px;/* Window size, not including borders */
    border: 3px outset gray;    /* Note 3D "outset" border effect */
}

div.titlebar {  /* Specifies position, size, and style of the titlebar */
    position: absolute;         /* It's a positioned element */
    top: 0px; height: 18px;     /* Titlebar is 18px + padding and borders */
    width: 290px;               /* 290 + 5px padding on left and right = 300 */
    background-color: #aaa;     /* Titlebar color */
    border-bottom: groove gray 2px;  /* Titlebar has border on bottom only */
    padding: 3px 5px 2px 5px;   /* Values clockwise: top, right, bottom, left */
    font: bold 11pt sans-serif; /* Title font */
}

div.content {  /* Specifies size, position and scrolling for window content */
    position: absolute;         /* It's a positioned element */
    top: 25px;                  /* 18px title+2px border+3px+2px padding */
    height: 165px;              /* 200px total - 25px titlebar - 10px padding*/
    width: 290px;               /* 300px width - 10px of padding */
    padding: 5px;               /* Allow space on all four sides */
    overflow: auto;             /* Give us scrollbars if we need them */
    background-color: #fff;     /* White background by default */
}

div.translucent { /* this class makes a window partially transparent */
    opacity: .75;               /* Standard style for transparency */
    filter: alpha(opacity=75);  /* Transparency for IE */
}
</style>
</head>

<body>
<!-- Here is how we define a window: a "window" div with a titlebar and -->
<!-- content div nested inside. Note how position is specified with -->
<!-- a style attribute that augments the styles from the stylesheet. -->
<div class="window" style="left: 10px; top: 10px; z-index: 10;">
<div class="titlebar">Test Window</div>
<div class="content">
1<br>2<br>3<br>4<br>5<br>6<br>7<br>8<br>9<br>0<br><!-- Lots of lines to -->
1<br>2<br>3<br>4<br>5<br>6<br>7<br>8<br>9<br>0<br><!-- demonstrate scrolling-->
</div>
</div>
```

```
<!-- Here's another window with different position, color, and font weight -->
<div class="window" style="left: 75px; top: 110px; z-index: 20;">
<div class="titlebar">Another Window</div>
<div class="content translucent"
     style="background-color:#ccc; font-weight:bold;">
This is another window. Its <tt>z-index</tt> puts it on top of the other one.
CSS styles make its content area translucent, in browsers that support that.
</div>
</div>
```


The major shortcoming of this example is that the stylesheet specifies a fixed size for all windows. Because the titlebar and content portions of the window must be precisely positioned within the overall window, changing the size of a window requires changing the value of various positioning properties in all three rules defined by the stylesheet. This is difficult to do in a static HTML document, but it would not be so difficult if you could use a script to set all the necessary properties. This topic is explored in the next section.

16.3 Scripting Inline Styles

The most straightforward way to script CSS is to alter the `style` attribute of individual document elements. Like most HTML attributes, `style` is also a property of the Element object, and you can manipulate it in JavaScript. The `style` property is unusual, however: its value is not a string, but a CSSStyleDeclaration object. The JavaScript properties of this style object represent the CSS properties specified by the HTML `style` attribute. To make the text of an element `e` big, bold, and blue, for example, you can use the following code to set the JavaScript properties that correspond to the `font-size` `font-weight` and `color` style properties:

```
e.style.fontSize = "24pt";
e.style.fontWeight = "bold";
e.style.color = "blue";
```

Naming Conventions: CSS Properties in JavaScript

Many CSS style properties, such as `font-size`, contain hyphens in their names. In JavaScript, a hyphen is interpreted as a minus sign, so it is not possible to write an expression like:

```
e.style.font-size = "24pt"; // Syntax error!
```

Therefore, the names of the properties of the CSSStyleDeclaration object are slightly different from the names of actual CSS properties. If a CSS property name contains one or more hyphens, the CSSStyleDeclaration property name is formed by removing the hyphens and capitalizing the letter immediately following each hyphen. Thus, the CSS property `border-left-width` is accessed through the JavaScript `borderLeftWidth` property, and you can access the CSS `font-family` property with code like this:

```
e.style.fontFamily = "sans-serif";
```