CHAPTER
8

Boosting Styles with CSS3

It would be almost ludicrous to build a modern website *without* CSS. The standard is fused into the fabric of the Web almost as tightly as HTML. Whether you're laying out pages, building interactive buttons and menus, or just making things look pretty, CSS is a fundamental tool. In fact, as HTML has increasingly shifted its focus to content and semantics (page 44), CSS has become the heart and soul of web *design*.

Along the way, CSS has become far more detailed and far more complex. When CSS evolved from its first version to CSS 2.1, it quintupled in size, reaching the size of a modest novel. Fortunately, the creators of the CSS standard had a better plan for future features. They carved the next generation of enhancements into a set of separate standards, called *modules*. That way, browser makers were free to implement the most exciting and popular parts of the standard first—which is what they were already doing, modules or not. All together, the new CSS modules fall under the catchall name *CSS3* (note the curious lack of a space, as with *HTML5*).

CSS3 has roughly 50 modules in various stages of maturity. They range from features that provide fancy eye candy (like rich fonts and animation) to ones that serve a more specialized, practical purpose (for example, speaking text aloud or varying styles based on the capabilities of the computer or mobile device). Altogether, they include features that are reliably supported in the most recent versions of all modern browsers and features so experimental that *no* browser yet supports them.

In this chapter, you'll tour some of the most important (and best supported) parts of CSS3. First, you'll learn how to jazz up your text with virtually any font. Then, you'll learn how to tailor your styles to suit different sized browser windows and different types of web-connected devices, like iPads and iPhones. Next, you'll see how to use shadows, rounded corners, and other refinements to make your boxes look better. Finally, you'll learn how you can use transitions to create subtle effects when the visitor hovers over an element, clicks on it, or tabs over to a control. (And you'll make these effects even better with two more CSS3 features: transforms and transparency.)

But first, before you get to any of these hot new features, it's time to consider how you can plug in the latest and most stylin' styling features without leaving a big chunk of your audience behind.

Using CSS3 Today

CSS3 is the unchallenged future of web styling, and it's not finished yet. Most modules are still being refined and revised, and no browser supports them all. This means CSS has all the same complications as HTML5. Website authors like yourself need to decide what to use, what to ignore, and how to bridge the gaping support gaps.

There are essentially three strategies you can use when you start incorporating CSS3 into a website. The following sections describe them.

Note: CSS3 is not part of HTML5. The standards were developed separately, by different people working at different times in different buildings. However, even the W3C encourages web developers to lump HTML5 and CSS3 together as part of the same new wave of modern web development. For example, if you check out the W3C's HTML5 logo-building page at *www.w3.org/html/logo*, you'll see that it encourages you to advertise CSS3 in its HTML5 logo strips.

Strategy 1: Use What You Can

It makes sense to use features that already have solid browser support across all browser brands. One example is the web font feature (page 244). With the right font formats, you can get it working with browsers all the way back to IE 6. Unfortunately, very few CSS3 features fall into this category. The word-wrap property works virtually everywhere, and older browsers can do transparency with a bit of fiddling, but just about every other feature leaves the still-popular IE 7 and IE 8 browsers in the dust.

Note: Unless otherwise noted, the features in this chapter work on the latest version of every modern browser, including Internet Explorer 9. However, they don't work on older versions of IE.

Strategy 2: Treat CSS3 Features as Enhancements

CSS3 fans have a rallying cry: "Websites don't need to look exactly the same on every browser." Which is certainly true. (They have a website, too—see *http://DoWebsitesNeedToBeExperiencedExactlyTheSameInEveryBrowser.com*.)

The idea behind this strategy is to use CSS3 to add fine touches that won't be missed by people using less-capable browsers. One example is the border-radius property that you can use to gently round the corners of a floating box. Here's an example:

```
header {
  background-color: #7695FE;
  border: thin #336699 solid;
  padding: 10px;
  margin: 10px;
  text-align: center;
  border-radius: 25px;
}
```

Browsers that recognize the border-radius property will know what to do. Older browsers will just ignore it, keeping the plain square corners (Figure 8-1).

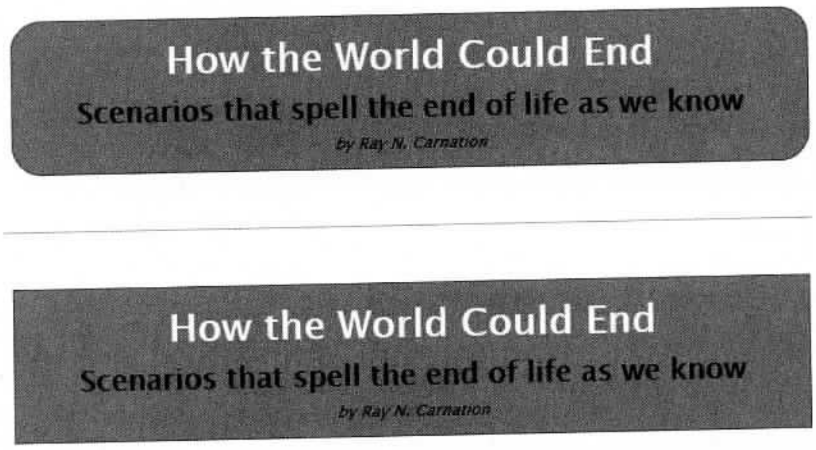

How the World Could End
Scenarios that spell the end of life as we know
by Ray N. Carnation
How the World Could End
Scenarios that spell the end of life as we know
by Ray N. Carnation

Figure 8-1: *On Internet Explorer 9, this header box has rounded corners (top). IE 8 ignores the border-radius property but applies the rest of the style sheet properties (bottom).*

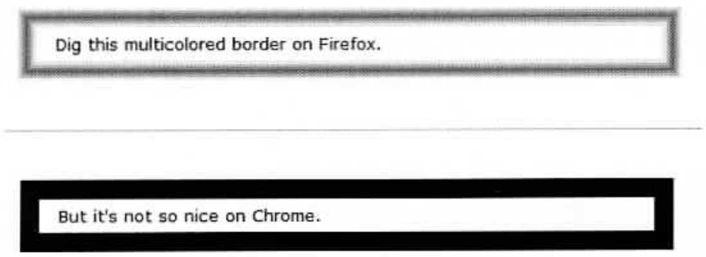
There's an obvious appeal to this strategy, because it gives web designers justification to play with all the latest technology toys. However, there's a definite downside if you go too far. No matter how good a website looks in the latest version of your favorite browser, it can be deeply deflating if you fire up an older browser that's used by a significant slice of your clientele and find that it looks distinctly less awesome. After all, you want your website to impress everyone, not just web nerds with the best browsers.

For this reason, you may want to approach some CSS3 enhancements with caution. Limit yourself to features that are already in multiple browsers (and are at least promised for IE 10). And don't use them in ways that change the experience of your website so dramatically that some people are getting second-rate status.

Tip: When it comes to CSS3, Internet Explorer is the straggler. There's a militant minority of web designers who believe that web designers should ignore IE and start using CSS3 features as soon as other browsers support them. Otherwise, who will keep pressure on Microsoft and encourage the Web to get better? This is all well and fine, if the primary purpose of your website is the political one of promoting advanced web standards. But otherwise, keep in mind that dismissing a large segment of the web world will reflect poorly on you—because no matter how much you dislike someone's browser, that person is still using it to look at *your* work.

Strategy 3: Add Fallbacks with Modernizr

Using a partially supported CSS3 feature is a great idea if the website still looks great without it. But sometimes, a vital part of your website design can go missing, or the downgraded version of your website just looks ugly. For example, consider what happens if you use the Firefox-only multicolored border settings, as shown in Figure 8-2.


Dig this multicolored border on Firefox.

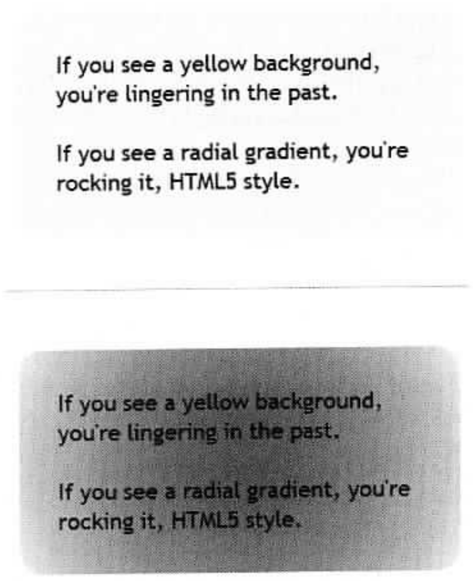
But it's not so nice on Chrome.

Figure 8-2:
This multicolored border looks snazzy in Firefox (top). But try the same thing out in Chrome, and you'll get a thick, plain black border (bottom)—and that never looks good.

Sometimes, you can solve the problem by stacking properties in the right order. The basic technique is to start with more general properties, followed by new properties that *override* these settings. When this works, it satisfies every browser—the old browsers get the standard settings, while the new browsers override these settings with newer ones. For example, you can use this technique to replace an ordinary background fill with a gradient:

```
.stylishBox  {
  ...
  background: yellow;
  background: radial-gradient(ellipse, red, yellow);
}
```

Figure 8-3 shows the result.



If you see a yellow background, you're lingering in the past.

If you see a radial gradient, you're rocking it, HTML5 style.

If you see a yellow background, you're lingering in the past.

If you see a radial gradient, you're rocking it, HTML5 style.

Figure 8-3:
Top: In browsers that don't understand CSS3, the stylishBox rule paints a yellow background.

Bottom: In browsers that do understand CSS3, the yellow background is replaced with a radial gradient that blends from a red center point to yellow at the edges. (At least, that's the plan. This example doesn't work exactly as written, because the radial-gradient standard is still being revised. To get the result shown here, you need vendor prefixes, as described on page 243.)

In some cases, overriding style properties doesn't work, because you need to set properties in *combination*. The multicolored border in Figure 8-2 is an example. The multicolored effect is set with the *border-colors* property, but appears only if the border is made thick with the *border-thickness* property. On browsers that don't support multicolored borders, the thick border is an eyesore, no matter what single color you use.

One way to address problems like these is with Modernizr, the JavaScript library that tests HTML5 feature support (page 38). It lets you define alternate style settings for browsers that don't support the style properties you really want. For example, imagine you want to create two versions of the header box shown in Figure 8-1. You want to use rounded corners if they're supported, but substitute a double-line border if they aren't. If you've added the Modernizr script reference to your page, then you can use a combination of style rules, like this:

```
/* Settings for all headers, no matter what level of CSS3 support. */
header {
  background-color: #7695FE;
  padding: 10px;
  margin: 10px;
  text-align: center;
}
```

```
/* Settings for browsers that support border-radius. */
.borderradius header {
  border: thin #336699 solid;
  border-radius: 25px;
}

/* Settings for browsers that don't support border-radius. */
.no-borderradius header {
  border: 5px #336699 double;
}
```

So how does this nifty trick work? When you use Modernizr in a page, you begin by adding the *class="no-js"* attribute to the root <html> element:

```
<html class="no-js">
```

When you load Modernizr on a page, it quickly checks if a range of HTML5, JavaScript, and CSS3 features are supported. It then applies a pile of classes to the root <html> element, separated by spaces, changing it into something like this:

```
<html class="js flexbox canvas canvastext webgl no-touch geolocation
postmessage no-websqldatabase indexeddb hashchange history draganddrop
no-websockets rgba hsla multiplebgs backgroundsize borderimage borderradius
boxshadow textshadow opacity no-cssanimations csscolumns cssgradients
no-cssreflections csstransforms no-csstransforms3d csstransitions fontface
generatedcontent video audio localstorage sessionstorage webworkers
applicationcache svg inlinesvg smil svgclippaths">
```

If a feature appears in the class list, that feature is supported. If a feature name is prefixed with the text "no-" then that feature is not supported. Thus, in the example shown here, JavaScript is supported (js) but web sockets are not (no-websockets). On the CSS3 side of things, the border-radius property works (borderradius) but CSS3 reflections do not (no-cssreflections).

You can incorporate these classes into your selectors to filter out style settings based on support. For example, a selectors like *.borderradius header* gets all the <header> elements inside the root <html> element—if the browser supports the border-radius property. Otherwise, there will be no .borderradius class, the selector won't match anything, and the rule won't be applied.

The catch is that Modernizr provides classes for only a subset of CSS3 features. This subset includes some of CSS3's most popular and mature features, but the border-color feature in Figure 8-2 doesn't qualify because it's still Firefox-only. For that reason, it's a good idea to hold off on using multicolored borders in your pages, at least for now.

Note: You can also use Modernizr to create JavaScript fallbacks. In this case, you simply need to check the appropriate property of the Modernizr object, as you do when checking for HTML5 support. You could use this technique to compensate if you're missing more advanced CSS3 features, like transitions or animations. However, there's so much work involved and the models are so different that it's usually best to stick with a JavaScript-only solution for essential website features.

Browser-Specific Styles

When the creators of CSS develop new features, they often run into a chicken-and-egg dilemma. In order to perfect the feature, they need feedback from browser makers and web designers. But in order to get this feedback, browser makers and web designers need to implement these new-and-imperfect features. The result is a cycle of trial and feedback that takes many revisions to settle down. While this process unfolds, the syntax and implementation of features change. This raises a very real danger—unknowing web developers might learn about a dazzling new feature and implement it in their real-life websites, not realizing that future versions of the standard could change the rules and break the websites.

To avoid this threat, browser makers use a system of *vendor prefixes* to change CSS property and function names while they're still under development. For example, consider the new radial-gradient property described on page 271. To use it with Firefox, you need to set the "in progress" version of this property called *-moz-radial-gradient*. That's because Firefox uses the vendor prefix *-moz-* (which is short for Mozilla, the organization that's behind the Firefox project).

Every browser engine has its own vendor prefix (Table 8-1). This complicates life horrendously, but it has a good reason. Different browser makers add support at different times, often using different draft versions of the same specification. Although all browsers will support the same syntax for final specification, the syntax of the vendor-specific properties and functions often varies.

Table 8-1. *Vendor prefixes*

Prefix	For Browsers
-moz-	Firefox
-webkit-	Chrome and Safari (the same rendering engine powers both browsers)
-ms-	Internet Explorer
-o-	Opera

So, if you want to use a radial gradient today and support all the browsers you can (including the forthcoming IE 10), you'll need to use a bloated CSS rule like this one:

```
.stylishBox {
  background: yellow;
  background-image: -moz-radial-gradient(circle, green, yellow);
  background-image: -webkit-radial-gradient(circle, green, yellow);
  background-image: -o-radial-gradient(circle, green, yellow);
  background-image: -ms-radial-gradient(circle, green, yellow);
}
```

In this example, each rule sets the radial gradient using the same syntax. This indicates that the standard is settling down, and browser makers may soon be able to drop the vendor prefix and support the radial-gradient property directly (as they currently support corner-radius). However, this is a fairly recent development, as old versions of Chrome used completely different gradient syntax.

Note: Using vendor prefixes is a messy business. Web developers are split on whether they're a necessary evil of getting the latest and greatest frills, or a big fat warning sign that should scare clear-thinking designers away. But one thing is certain: If you don't use the vendor prefixes, a fair bit of CSS3 will be off-limits for now.

Web Typography

With all its pizzazzy new features, it's hard to pick the best of CSS3. But if you had to single out just one feature that opens an avalanche of new possibilities and is ready to use *right now*, that feature may just be web fonts.

In the past, web designers had to work with a limited set of web-safe fonts. These are the few fonts that are known to work on different browsers and operating systems. But as every decent designer knows, type plays a huge role in setting the overall atmosphere of a document. With the right font, the same content can switch from coolly professional to whimsical, or from old-fashioned to futuristic.

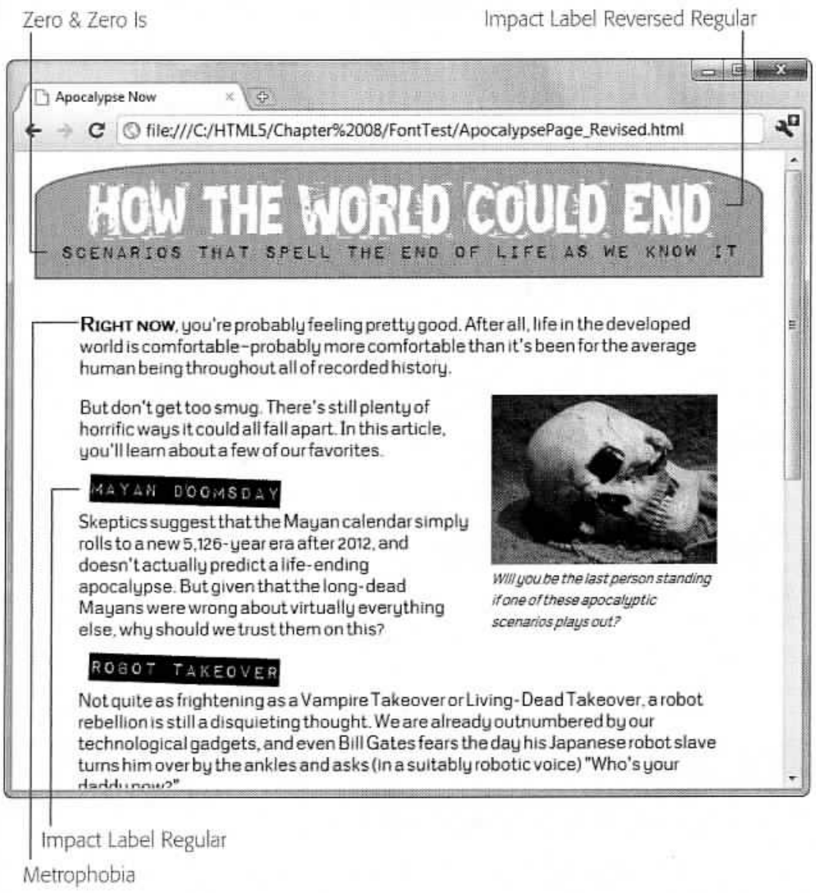
Note: There were good reasons why web browsers didn't rush to implement custom web fonts. First, there are optimization issues, because computer screens offer far less resolution than printed documents. If a web font isn't properly tweaked for onscreen viewing, it'll look like a blurry mess at small sizes. Second, most fonts aren't free. Big companies like Microsoft were understandably reluctant to add a feature that could encourage web developers to take the fonts installed on their computers and upload them to a website without proper permission. As you'll see in the next section, font companies now have good solutions for both problems.

CSS3 adds support for fancy fonts with the @font-face feature. Here's how it works:

1. You upload the font to your website (or, more likely, multiple versions of that font to support different browsers).
2. You register each font-face you want to use in your style sheet, using the @font-face command.
3. You use the registered font in your styles, by name, just as you use the web-safe fonts.
4. When a browser encounters a style sheet that uses a web font, it downloads the font to its temporary cache of pages and pictures. It then uses that font for just your page or website (Figure 8-4). If other web pages want to use the same font, they'll need to register it themselves and provide their own font files.

Note: Technically, @font-face isn't new. It was a part of CSS 2, but dropped in CSS 2.1 when the browser makers couldn't cooperate. Now, in CSS3, there's a new drive to make @font-face a universal standard.

The following sections walk you through these essential steps.



file:///C:/HTML5/Chapter%2008/FontTest/ApocalypsePage_Revised.html

HOW THE WORLD COULD END

SCENARIOS THAT SPELL THE END OF LIFE AS WE KNOW IT

RIGHT NOW, you're probably feeling pretty good. After all, life in the developed world is comfortable–probably more comfortable than it's been for the average human being throughout all of recorded history.

But don't get too smug. There's still plenty of horrific ways it could all fall apart. In this article, you'll learn about a few of our favorites.

MAYAN DOOMSDAY

Skeptics suggest that the Mayan calendar simply rolls to a new 5,126-year era after 2012, and doesn't actually predict a life-ending apocalypse. But given that the long-dead Mayans were wrong about virtually everything else, why should we trust them on this?

Will you be the last person standing if one of these apocalyptic scenarios plays out?

ROBOT TAKEOVER

Not quite as frightening as a Vampire Takeover or Living-Dead Takeover, a robot rebellion is still a disquieting thought. We are already outnumbered by our technological gadgets, and even Bill Gates fears the day his Japanese robot slave turns him over by the ankles and asks (in a suitably robotic voice) "Who's your daddy now?"

Figure 8-4: *This revised version of the apocalyptic page combines a hodgepodge of four fonts. All of these fonts are free, and you'll learn how to get them for yourself from Font Squirrel on page 247.*

Web Font Formats

Although all current browsers support @font-face, they don't all support the same *types* of font files. Internet Explorer, which has supported @font-face for years, supports only EOT (Embedded OpenType) font files. This format has a number of advantages—for example, it uses compression to reduce the size of the font file, and it allows strict website licensing so a font can't be stolen from one website and used on another. However, the .eot format never caught on, and no other browser uses it. Instead, other browsers have (until recently) stuck with the more familiar font standards used in desktop computer applications—that's TTF (TrueType) and OTF (OpenType PostScript). But the story's still not complete without two more acronyms—SVG and WOFF. Table 8-2 puts all the font formats in perspective.

Table 8-2. *Embedded font formats*

Format	Description	Use with
TTF (TrueType) OTF (OpenType PostScript)	Your font will probably begin in one of these common desktop formats.	Firefox (before version 3.6), Chrome (before version 6), Safari, and Opera
EOT (Embedded Open Type)	A Microsoft-specific format that never caught on with browsers except Internet Explorer.	Internet Explorer (before IE 9)
SVG (Scalable Vector Graphics)	An all-purpose graphics format you can use for fonts, with good but not great results (it's slower to display and produces lower-quality text).	Safari Mobile (on the iPhone and iPad before iOS 4.2), and mobile devices using the Android operating system.
WOFF (Web Open Font Format)	The single format of the future, probably. Newer browsers support it.	Any browser that supports it, starting with Internet Explorer 9, Firefox 3.6, and Chrome 6.

Bottom line: If you want to use the @font-face feature and support a wide range of browsers, you need to distribute your font in several different formats. At a minimum, you need to supply your font in TTF or OTF format (either one is fine), the EOT format, and the SVG format. It's a good idea (but not essential) to also supply a WOFF font, which is likely to become more popular and better supported in the future. (Among its advantages, WOFF files are compressed, which minimizes the download time.)

TROUBLESHOOTING MOMENT

Ironing Out the Quirks

Even if you follow the rules and supply all the required font formats, expect a few quirks. Here are some problems that occasionally crop up with web fonts:

- Many fonts look bad on the still-popular Windows XP operating system, because Windows XP computers often have the anti-aliasing display setting turned off. (And fonts without anti-aliasing look as attractive as mascara on a mule.)
- Some people have reported that some browsers (or some operating systems) have trouble printing certain embedded fonts.
- Some browsers suffer from a problem known as FOUT (which stands for Flash of Unstyled Text). This phenomenon occurs when an embedded font takes a few seconds to download, and the page is rendered first using a fallback font, and then re-rendered using the embedded font. This problem is most noticeable on old builds of Firefox. If it really bothers you, Google provides a JavaScript library that lets you define fallback styles that kick in for unloaded fonts, giving you complete control over the rendering of your text at all times (see *http://code.google.com/apis/webfonts/docs/webfont_loader.html*).

Although these quirks are occasionally annoying, most are being steadily ironed out in new browser builds. For example, Firefox now minimizes FOUT by waiting for up to 3 seconds to download an embedded font before using the fallback font.

Using a Font Kit

At this point, you're probably wondering where you can get the many font files you need. The easiest option is to download a ready-made font kit from the Web. That way, you get all the font files you need. The disadvantage is that your selection is limited to whatever you can find online. One of the best places to find web font kits is the Font Squirrel website; you can see its handpicked selection at *www.fontsquirrel.com/fontface* (see Figure 8-5).

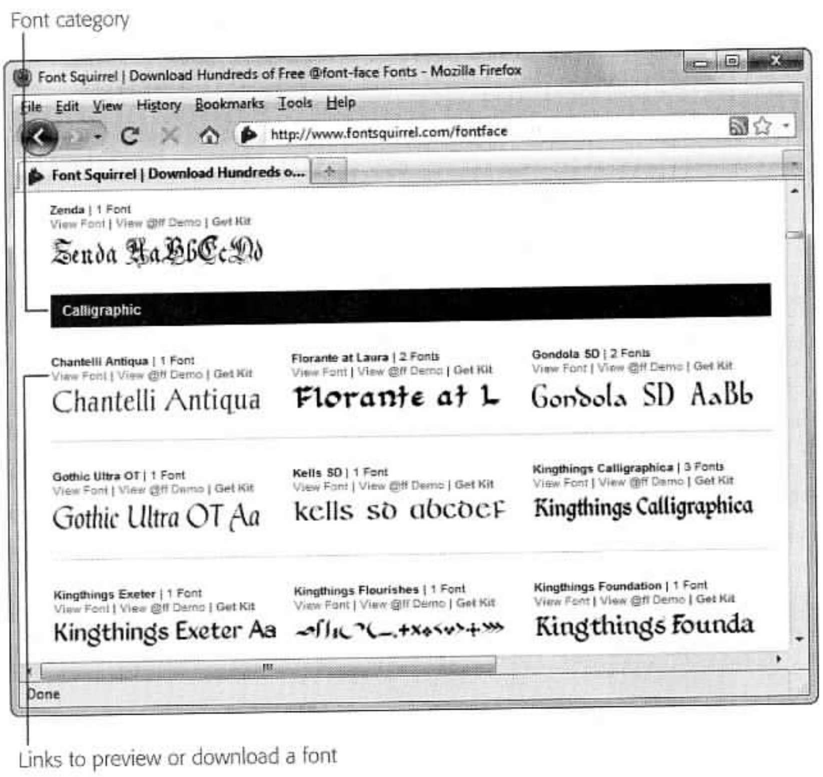

Font category
Font Squirrel | Download Hundreds of Free @font-face Fonts - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.fontsquirrel.com/fontface
Font Squirrel | Download Hundreds o...
Zenda | 1 Font
View Font | View @ff Demo | Get Kit
Calligraphic
Chantelli Antiqua | 1 Font
Chantelli Antiqua
Florante at Laura | 2 Fonts
Florante at L
Gondola SD | 2 Fonts
Gondola SD AaBb
Gothic Ultra OT | 1 Font
Gothic Ultra OT Aa
Kells SD | 1 Font
kells sd abcdef
Kingthings Calligraphica | 3 Fonts
Kingthings Calligraphica
Kingthings Exeter | 1 Font
Kingthings Exeter Aa
Kingthings Flourishes | 1 Font
Kingthings Foundation | 1 Font
Kingthings Founda
Done
Links to preview or download a font

Figure 8-5: *Font Squirrel provides a few hundred high-quality fonts, organized into sections by type ("Calligraphic," "Grunge," and "Retro," for example). Best of all, every font is free to use wherever you want—on your personal computer to create documents, or on the Web to build web pages. When you find a font you like, click View Font to get a closer look at every letter, "View @ff Demo" to see the font in a sample web page via the @font-face feature, and "Get Kit" to download the font to your computer.*

When you download a font kit, you get a compressed Zip file that contains a number of files. For example, download the Chantelli Antiqua font shown in Figure 8-5, and you get these files:

```
Bernd Montag License.txt
Chantelli_Antiqua-webfont.eot
Chantelli_Antiqua-webfont.svg
Chantelli_Antiqua-webfont.ttf
Chantelli_Antiqua-webfont.woff
demo.html
stylesheet.css
```

The text file (Bernd Montag License.txt) provides licensing information that basically says you can use the font freely, but never sell it. The Chantelli_Antiqua-webfont files provide the font in four different file formats. (Depending on the font you pick, you may get additional files for different variations of that font—for example, in bold, italic, and extra-dark styles.) Finally, the *stylesheet.css* file contains the style sheet rule you need to apply the font to your web page, and *demo.html* displays the font in a sample web page.

To use the Chantelli Antiqua font, you need to copy all the Chantelli_Antiqua-webfont files to the same folder as your web page. Then you need to register the font so that it's available for use in your style sheet. To do that, you use a complex @font-face rule at the beginning of your style sheet, which looks like this (with the lines numbered for easy reference):

```
1   @font-face {
2     font-family: 'ChantelliAntiquaRegular';
3     src: url('Chantelli_Antiqua-webfont.eot');
4     src: local('Chantelli Antiqua'),
5       url('Chantelli_Antiqua-webfont.woff') format('woff'),
6       url('Chantelli_Antiqua-webfont.ttf') format('truetype'),
7       url('Chantelli_Antiqua-webfont.svg') format('svg');
8   }
```

To understand what's going on in this rule, it helps to break it down line by line:

- **Line 1:** @font-face is the tool you use to officially register a font so you can use it elsewhere in your style sheet.
- **Line 2:** You can give the font any name you want. This is the name you'll use later, when you apply the font.
- **Line 3:** The first format you specify has to be the file name of the EOT file. That's because Internet Explorer gets confused by the rest of the rule, and ignores the other formats. The url() function is a style sheet technique that tells a browser to download another file at the location you specify. If you put the font in the same folder as your web page, then you can simply provide the file name here.
- **Line 4:** The next step is to use the local() function. This function tells the browser the font name, and if that font just happens to be installed on the visitor's computer, the browser uses it. However, in rare cases this can cause a problem (for example, it could cause Mac OS X to show a security dialog box, depending on where your visitor has installed the font, or it could load a different font that has the same name). For these reasons, web designers sometimes use an obviously fake name to ensure that the browser finds no local font. One common choice is to use a meaningless symbol like *local('☺')*.
- **Lines 5 to 7:** The final step is to tell the browser about the other font files it can use. If you have a WOFF font file, suggest that first, as it offers the best quality. Next, tell the browser about the TTF or OTF file, and finally about the SVG file.

Tip: Of course, you don't need to type the @font-face rule by hand (and you definitely don't need to understand all the technical underpinnings described above). You can simply copy the rule from the *stylesheet.css* file that's included in the web font kit.

Once you register an embedded font using the @font-face feature, you can use it in any style sheet. Simply use the familiar font-family property, and refer to the font family name you specified with @font-face (in line 2). Here's an example that leaves out the full @font-face details:

```
@font-face {
  font-family: 'ChantelliAntiquaRegular';
  ...
}

body {
  font-family: 'ChantelliAntiquaRegular';
}
```

This rule applies the font to the entire web page, although you could certainly restrict it to certain elements or use classes. However, you must register the font with @font-face *before* you use it in a style rule. Reverse the order of these two steps, and the font won't work properly.

Tip: The Font Squirrel website provides more fonts beyond its prepackaged font kits. You can also find more fonts by looking at the list of most popular fonts (click Most Downloaded) and most recent (click Newly Added). In these lists, you'll find the web font kits along with other free font files that don't have the support files or need to be downloaded from another website. You can convert your font to the other formats you need using Font Squirrel's font-kit generator, as described on page 252.

FREQUENTLY ASKED QUESTION

Using a Font on Your Computer

Can I use the same font for web work and printed documents?

If you find a hot new font to use in your website, you can probably put it to good use on your computer, too. For example, you might want to use it in an illustration program to create a logo. Or, your business might want to use it for other print work, like ads, fliers, product manuals, and financial reports.

Modern Windows and Mac computers support TrueType (.ttf) and OpenType (.otf) fonts. Every font package includes a font in one of these formats–usually, TrueType. To install it in Windows, make sure you've pulled it out of the ZIP download file. Then, right-click it and choose Install. (You can do this with multiple font files at once.) On a Mac, double-click the font file to open the Font Book utility. Then, click the Install Font button.

Using Google Web Fonts

If you want a simpler way to use a fancy font on your website, Google has got you covered. It provides a service called Google Web Fonts, which hosts free fonts that anyone can use. The beauty of Google Web Fonts is that you don't need to worry about font formats, because Google detects the browser and automatically sends the right font file.

To use a Google font in your pages, follow these steps:

1. **Go to *www.google.com/webfonts*.**

   Google shows you a long list of available fonts (Figure 8-6).

2. **At the top of the page, click a tab title (Word, Sentence, or Paragraph) to choose how you preview fonts.**

   For example, if you're hunting for a font to use in a heading, you'll probably choose Word or Sentence to take a close up look at a single word or line of text. But if you're looking for a font to use in your body text, you'll probably choose Paragraph to study a whole paragraph of text at once. No matter what option you choose, you can type in your own preview text and set an exact font size for your previews.

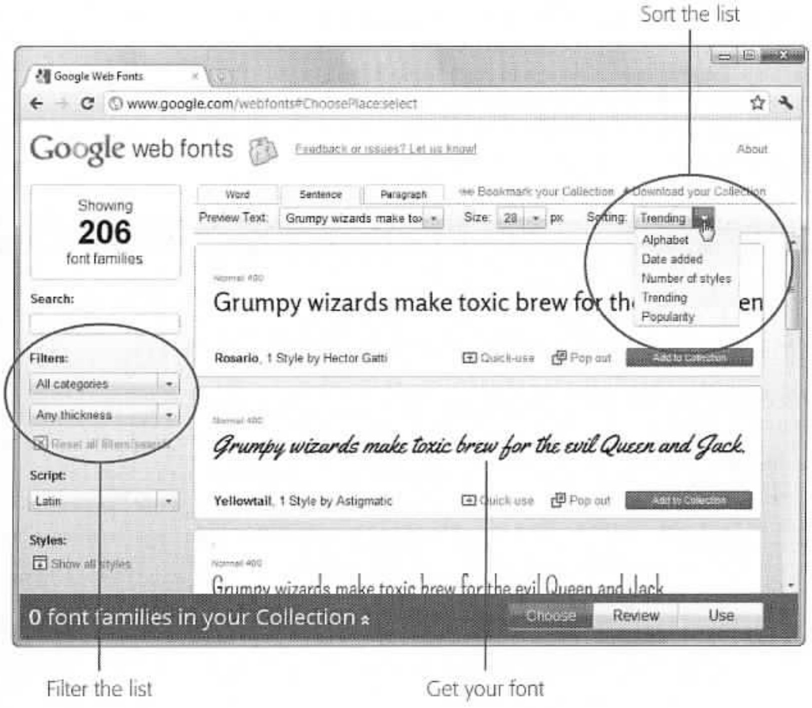


Figure 8-6: *Google has a relentlessly expanding selection of fonts. When you're looking for a font, you'll probably want to tweak the font list's sorting and filtering options (circled). For example, you can sort alphabetically or put the most popular fonts first, and you can filter out just serif, sans-serif, or handwritten (cursive) fonts.*

3. **Set your search options.**

   If you have a specific font in mind, type it into the search box. Otherwise, you'll need to scroll down, and that could take ages. To help you get what you want more quickly, start by setting a sort order and some filtering options, if they apply (for example, you might want to find the most popular bold sans-serif fonts). Figure 8-6 shows you where to find these options.

4. **When you see a font that's a candidate for your site, click "Pop out."**

   Google pops open an informative window that describes the font and shows each of its characters.

5. **If you like the font, click "Quick-use" to get the information you need to use it.**

   Google shows you the code you need to use this font. It consists of a style sheet link (which you must add to your web page) and an example of a style sheet rule that uses the font.

6. **Add a style sheet link to your web page.**

   For example, if you picked the Metrophobic font, Google wants you to place this link in the <head> section of your page:

   ```
   <link href="http://fonts.googleapis.com/css?family=Metrophobic"
   rel="stylesheet">
   ```

   This style sheet registers the font, using @font-face, so you don't have to. Best of all, Google provides the font files, so you don't need to upload anything extra to your website.

Note: Remember to put the link for the Google font style sheet before your other style sheet links. That way, your other style sheets can use the Google font.

7. **Use the font, by name, wherever you want.**

   For example, here's how you could use the newly registered Metrophobic font in a heading, with fallbacks in case the browser fails to download the font file:

   ```
   h1 {
     font-family: 'Metrophobic', arial, serif;
   }
   ```

POWER USER'S CLINIC

Creating a Font Collection

These steps show the fastest way to get the markup you need for a font. However, you can get more options by creating a *font collection*.

A font collection is a way to package up multiple fonts. To start creating one, you simply click the "Add to Collection" button next to a font you like. As you add fonts to your collection, each one appears in the fat blue footer at the bottom of the page.

When you're finished picking the fonts you want, click the footer's big Use button. Google then shows a page that's similar to the "Quick-use" page, except it allows you to create a single style sheet reference that supports *all* the fonts from your custom-picked collection.

When you create a font collection, you can also use two links that appear at the top-right of the page. Click "Bookmark your Collection" to create a browser bookmark that lets you load up the same collection at some point in the future, so you can tweak it. Choose "Download your collection" to download copies of the fonts to your computer, so you can install the fonts and use them for print work.

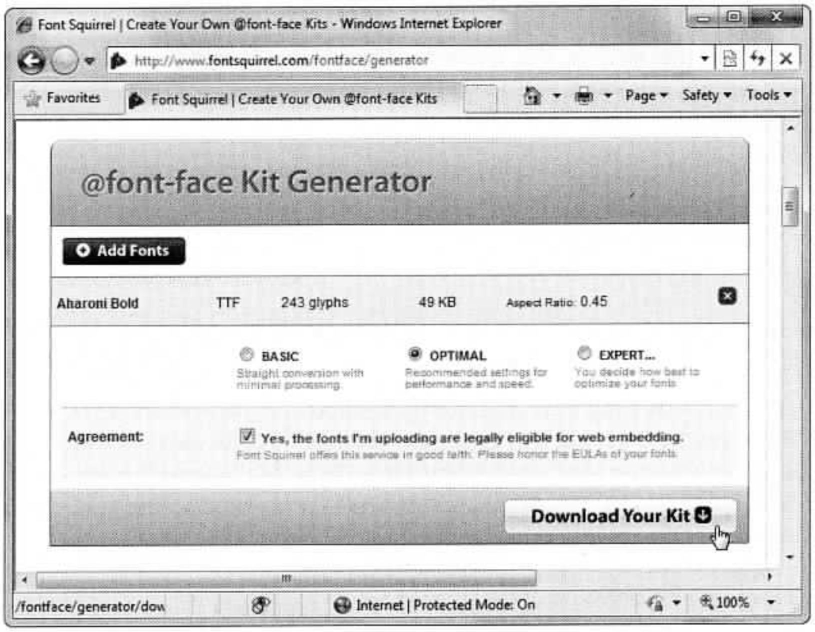
Using Your Own Fonts

Font fanatics are notoriously picky about their typefaces. If you have a specific font in mind for your web pages, even the biggest free font library isn't enough. Fortunately, there's a fairly easy way to adapt any font for the Web. Using the right tool, you can take a TTF or OTF font file you already have and create the other formats you need (EOT, SVG, and WOFF).

But before you take this road, it's important to get one issue out of the way: Ordinary fonts aren't free. That means it's not kosher to take a font you have on your computer and use it on your website, unless you have explicit permission from the font's creator. For example, Microsoft and Apple pay to include certain fonts with their operating systems and applications so you can use them to, say, create a newsletter in a word processor. However, this license doesn't allow you to put these fonts on a web server and use them in your pages.

Tip: If you have a favorite font, the only way to know whether you need to pay for it is to contact the company or individual that made it. Some font makers charge licensing fees based on the amount of traffic your website receives. Other font creators may let you use their fonts for a nominal amount or for free, provided you meet certain criteria (for example, you include some small-print note about the font you're using, or you have a noncommercial website that isn't out to make boatloads of money). There's also a side benefit to reaching out: Skilled font makers often provide display-optimized versions of their creations.

Once you know that you're allowed to use a specific font, you can convert it using a handy tool from Font Squirrel (the same website that offers the nifty free web font kits). To do so, surf to *www.fontsquirrel.com/fontface/generator*. Figure 8-7 shows you the three-step process you need to follow.


Font Squirrel | Create Your Own @font-face Kits - Windows Internet Explorer
http://www.fontsquirrel.com/fontface/generator
Favorites
Font Squirrel | Create Your Own @font-face Kits
Page ▾ Safety ▾ Tools ▾
@font-face Kit Generator
Add Fonts
Download Your Kit
/fontface/generator/dow
Internet | Protected Mode: On
100%

Aharoni Bold TTF 243 glyphs 49 KB Aspect Ratio: 0.45

BASIC — Straight conversion with minimal processing.
[✓ OPTIMAL] — Recommended settings for performance and speed.
EXPERT... — You decide how best to optimize your fonts.

Agreement: ☑ Yes, the fonts I'm uploading are legally eligible for web embedding.
Font Squirrel offers this service in good faith. Please honor the EULAs of your fonts.

Figure 8-7: *First, click Add Fonts to upload a font file from your computer. Then, add a checkmark to the setting "Yes, the fonts I'm uploading are legally eligible for web embedding" (assuming you've reviewed their license requirements, as described on page 252). Finally, click Download Your Kit.*

The web kit that Font Squirrel generates is just like the free ones described earlier. It even includes a style sheet that has the @font-face section you need, along with a test web page.

Tip: Still looking for more places to get fonts? If you haven't found that perfect typeface yet, spend some time at *http://webfonts.info*. There you'll find links to other websites that provide free fonts, as well as professional font foundries like the legendary Monotype. Although font-pricing models are changing fast, most font foundries offer some free fonts, and paid subscriptions can give website developers dozens of ultra-high-quality typefaces for as little as $10 per year. Some provide fonts through a hosting service like Google Fonts, so there's no need to upload font files to your web server.

Putting Text in Multiple Columns

Fancy fonts aren't the only innovation CSS3 has for displaying text. It also adds an entirely new module for multicolumn text, which gives you a flexible, readable way to deal with lengthy content.

Using multiple columns is almost effortless, and you have two ways to create them. Your first option is to set the number of columns you want using the *column-count* property, like this:

```
article {
  text-align: justify;
  column-count: 3;
}
```

At the time of this writing, this works for Opera only. To get the same support in Firefox, Chrome, and Safari, you need to add vendor-prefixed versions of the column-count property, like this:

```
article {
  text-align: justify;
  -moz-column-count: 3;
  -webkit-column-count: 3;
  column-count: 3;
}
```

You won't find any support in Internet Explorer 9 (although IE 10 is likely to join the party).

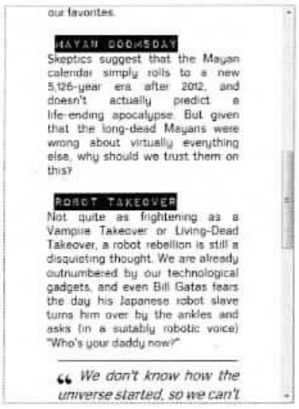
This approach—creating a set number of columns—works well for fixed-size layouts. But if you have a space that grows and shrinks with the browser window, your columns may grow too wide and become unreadable. In this situation, it's better *not* to set the exact number of columns. Instead, tell the browser how big each column should be using the column-width property:

```
article {
  text-align: justify;
  -moz-column-width: 10em;
  -webkit-column-width: 10em;
  column-width: 10em;
}
```

The browser can then create as many columns as it needs to fill up the available space (see Figure 8-8).

Note: You can use pixel units to size a column, but em units make more sense. That's because em units adapt to the current font size. So if a web page visitor ratchets up the text size settings in her browser, the column width grows proportionately to match. To get a sense of size, 1 em is equal to the two times the current font size. So if you have a 12 pixel font, 1 em works out to 24 pixels.

You can also adjust the size of the spacing between columns (with column-gap) and even add a vertical line that separates them (with column-rule). For more information about all your column-creating options, including ways to control where text breaks into columns and tricks that let figures and other elements span columns, refer to the full multicolumn standard at *www.w3.org/TR/css3-multicol*. Unfortunately, at this writing no browser supports these advanced features.



our favorites.

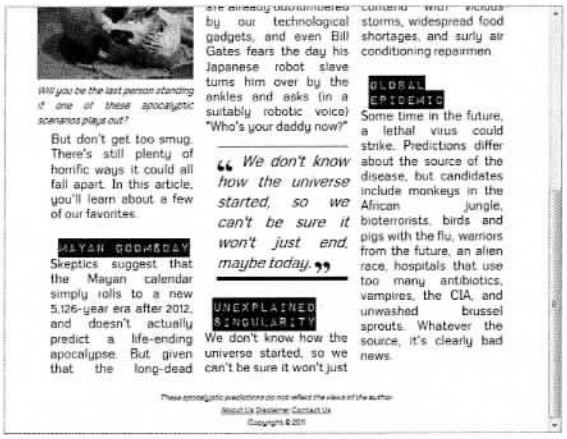
MAYAN DOOMSDAY

Skeptics suggest that the Mayan calendar simply rolls to a new 5,126-year era after 2012, and doesn't actually predict a life-ending apocalypse. But given that the long-dead Mayans were wrong about virtually everything else, why should we trust them on this?

ROBOT TAKEOVER

Not quite as frightening as a Vampire Takeover or Living-Dead Takeover, a robot rebellion is still a disquieting thought. We are already outnumbered by our technological gadgets, and even Bill Gates fears the day his Japanese robot slave turns him over by the ankles and asks (in a suitably robotic voice) "Who's your daddy now?"

❝ We don't know how the universe started, so we can't



Will you be the last person standing if one of these apocalyptic scenarios plays out?

But don't get too smug. There's still plenty of horrific ways it could all fall apart. In this article, you'll learn about a few of our favorites.

MAYAN DOOMSDAY

Skeptics suggest that the Mayan calendar simply rolls to a new 5,126-year era after 2012, and doesn't actually predict a life-ending apocalypse. But given that the long-dead

are already outnumbered by our technological gadgets, and even Bill Gates fears the day his Japanese robot slave turns him over by the ankles and asks (in a suitably robotic voice) "Who's your daddy now?"

❝ We don't know how the universe started, so we can't be sure it won't just end, maybe today.❞

UNEXPLAINED SINGULARITY

We don't know how the universe started, so we can't be sure it won't just

contend with vicious storms, widespread food shortages, and surly air conditioning repairmen.

GLOBAL EPIDEMIC

Some time in the future, a lethal virus could strike. Predictions differ about the source of the disease, but candidates include monkeys in the African jungle, bioterrorists, birds and pigs with the flu, warriors from the future, an alien race, hospitals that use too many antibiotics, vampires, the CIA, and unwashed brussel sprouts. Whatever the source, it's clearly bad news.

These apocalyptic predictions do not reflect the views of the author.

About Us Disclaimer Contact Us

Copyright © 2011

Figure 8-8:
In a narrow window (top), Firefox can accommodate just one column. But widen the window, and you'll get as many more as can fit (bottom).

Adapting to Different Devices

If you've ever gone on an extended web surfing spree using a mobile device (and odds are you have), you've discovered that a tiny screen isn't the best window onto the Web. Sure, you can scroll and zoom your way around virtually any website, but the process is often laborious. Life gets much better if you find a specifically designed mobile site that scales down its content to fit your device.

Today, it's not unusual to find developers creating custom versions of the same website for specific devices, like iPhones and iPads. These sites are often hosted on different web domains (like *http://m.nytimes.com* for the mobile version of the New York Times). But here's the rub: As mobile browsing becomes more popular, and mobile devices become increasingly numerous and varied, web developers like you can end up with big headaches managing all those device-specific sites.

Of course, separate sites aren't the only way to deal with different devices. You can also write web server code that checks every request, figures out what web browser is on the other end, and sends the appropriate type of content. This sort of solution is great, if you have the time and skills. But wouldn't it be nice to have a simple mechanism that tweaks your styles for different types of devices, with no web application framework or server-side code required?

Enter *media queries*. This CSS3 feature gives you a simple way to vary styles for different devices and viewing settings. Used carefully, they can help you serve everything from an ultra-widescreen desktop computer to an iPhone—without altering a single line of HTML.

Media Queries

Media queries work by latching onto a key detail about the device that's viewing your page (like its size, resolution, color capabilities, and so on). Based on that information, you can apply different styles, or even swap in a completely different style sheet. Figure 8-9 shows a media query in action.

NOSTALGIA CORNER

CSS Media Types

Interestingly, the creators of CSS took a crack at the multiple-device problem in CSS 2.1, using a feature called *media types*. You might already be using this standard to supply a separate style sheet for printouts:

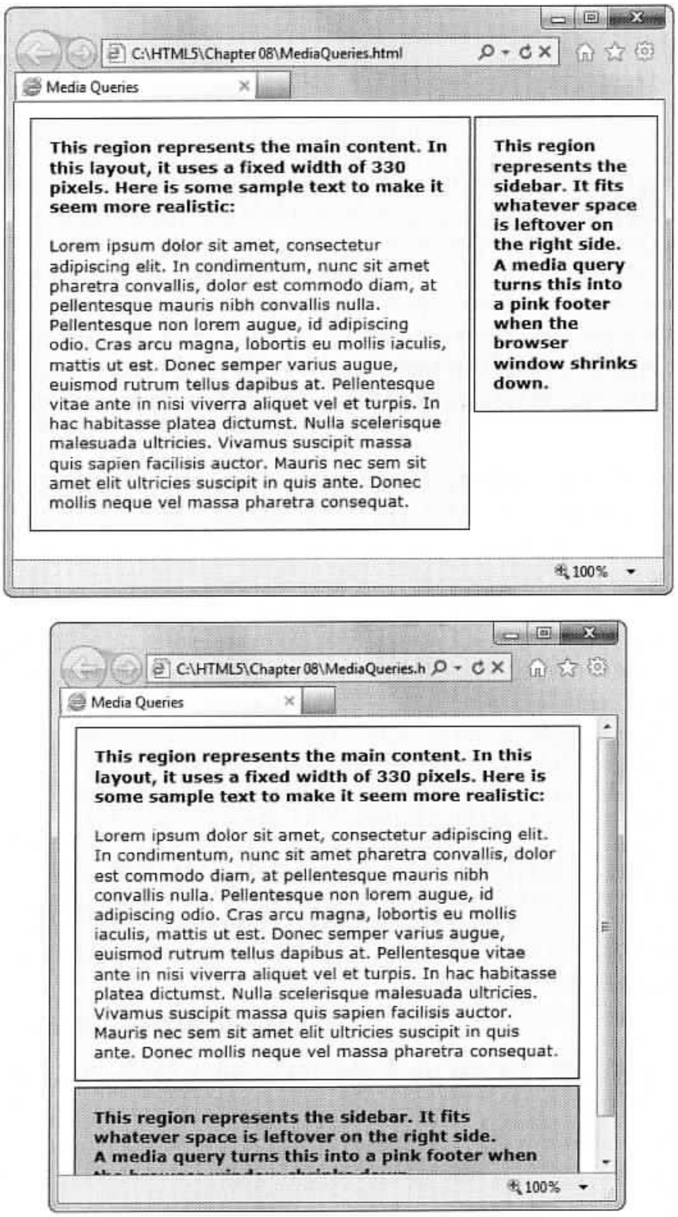
```
<head>
  ...
  <!-- Use this stylesheet to display the
       page on-screen. -->
  <link  rel="stylesheet" media="screen"
    href="styles.css">

  <!-- Use  this  stylesheet  to  print  the
       page. -->
  <link rel="stylesheet" media="print"
    href="print_styles.css">
</head>
```

The media attribute also accepts the value *handheld*, which is meant for low-bandwidth, small-screen mobile devices. Most mobile devices make some attempt to pay attention to the media attribute and use the handheld style sheet, if it exists. But there are quirks aplenty, and the media attribute is woefully inadequate for dealing with the wide range of web-connected devices that exists today. (However, it's still a good way to clean up printouts.)

To use media queries, you must first choose the property you want to examine. In Figure 8-9, that key detail is *max-width*, which gets the current size of the page, in the browser window. Even more useful is *max-device-width*, which checks the maximum screen width. If this value is small, it's clear that you're working with a webphone or a similarly tiny device.

The easiest way to use media queries is to start with the standard version of your website, and then override it selectively. In the example in Figure 8-9, the content is broken into two sections:

```
<article>
  ...
</article>
<aside>
  ...
</aside>
```



This region represents the main content. In this layout, it uses a fixed width of 330 pixels. Here is some sample text to make it seem more realistic:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In condimentum, nunc sit amet pharetra convallis, dolor est commodo diam, at pellentesque mauris nibh convallis nulla. Pellentesque non lorem augue, id adipiscing odio. Cras arcu magna, lobortis eu mollis iaculis, mattis ut est. Donec semper varius augue, euismod rutrum tellus dapibus at. Pellentesque vitae ante in nisi viverra aliquet vel et turpis. In hac habitasse platea dictumst. Nulla scelerisque malesuada ultricies. Vivamus suscipit massa quis sapien facilisis auctor. Mauris nec sem sit amet elit ultricies suscipit in quis ante. Donec mollis neque vel massa pharetra consequat.

This region represents the sidebar. It fits whatever space is leftover on the right side. A media query turns this into a pink footer when the browser window shrinks down.

This region represents the main content. In this layout, it uses a fixed width of 330 pixels. Here is some sample text to make it seem more realistic:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In condimentum, nunc sit amet pharetra convallis, dolor est commodo diam, at pellentesque mauris nibh convallis nulla. Pellentesque non lorem augue, id adipiscing odio. Cras arcu magna, lobortis eu mollis iaculis, mattis ut est. Donec semper varius augue, euismod rutrum tellus dapibus at. Pellentesque vitae ante in nisi viverra aliquet vel et turpis. In hac habitasse platea dictumst. Nulla scelerisque malesuada ultricies. Vivamus suscipit massa quis sapien facilisis auctor. Mauris nec sem sit amet elit ultricies suscipit in quis ante. Donec mollis neque vel massa pharetra consequat.

This region represents the sidebar. It fits whatever space is leftover on the right side. A media query turns this into a pink footer when

Figure 8-9:
Here's the same page viewed in a wide browser window (top) and in a narrow browser window (bottom). The media query magic automatically switches a portion of the page's style rules when the window shrinks, turning a sidebar into a footer. You don't even need to refresh the page.

And the style sheet starts with two rules, one for each section:

```
article {
  border: solid 1px black;
  padding: 15px;
  margin: 5px;
  background: yellow;
  float: left;
  width: 330px;
}

aside {
  border: solid 1px black;
  padding: 15px;
  margin: 5px;
  background: yellow;
  position: absolute;
  float: left;
  margin-left: 370px;
}
```

These rules implement a standard two-column layout, with a fixed 330-pixel column on the left, and a sidebar on the right that expands to fill up the remaining space. (Of course, you're free to use any sort of CSS-powered layout you want in your examples.)

The magic happens when you define a separate part of your style sheet that comes into effect for a given media-query value. The syntax takes this form:

```
@media (media-query-property-name: value) {
  /* New styles go here. */
}
```

In the current example, this new set of styles comes into effect when the width of the browser window is 480 pixels or less. That means you need a section in your style sheet that looks like this:

```
@media (max-width: 480px) {
  ...
}
```

Tip: Right now, the most popular media queries are max-device-width (for creating mobile versions of your pages), max-width (for varying styles based on the current size of the browser window), and orientation (for changing your layout based on whether an iPad is turned horizontally or vertically). But the media queries specification defines a handful of other details you can examine. For the full list, check out the standard at *www.w3.org/TR/css3-mediaqueries*.

Although you're free to put anything inside the media query block, this example simply adds new style rules for the <article> and <aside> elements:

```
@media (max-width: 480px) {
  article {
    float: none;
    width: auto;
  }
```

```
    aside {
      position: static;
      float: none;
      background: pink;
      margin-left: 5px;
    }
  }
```

These styles are applied in addition to the normal styles you've already defined. Thus, you may need to reset properties you've already changed to their default values. In this example, the media query styles reset the position property to *static*, the float property to *none*, and the width property to *auto*. These are the default values, but the original sidebar style changed them.

Note: Browsers that don't understand media queries, like Internet Explorer 8, will simply ignore these new styles and keep applying the original styles, no matter how big or small the browser window becomes.

If you want, you could add another media query section that overrides these rules at a still-smaller size. For example, this section would apply new rules when the browser width creeps under 250 pixels:

```
@media (max-width: 250px) {
  ...
}
```

Just remember that these rules are overriding everything that's been applied so far—in other words, the cumulative set of properties that have been set by the normal styles and the media query section for under 450 pixels. If this seems too confusing, don't worry—you'll learn to work around it with more tightly defined media queries in the next section.

Tip: When trying to identify mobile devices like webphones, you need to use the *max-device-width* property, not max-width. That's because the max-width property uses the size of the phone's viewport—the segment of the web page that the phone user can scroll around. A typical viewport is twice as wide as the actual device width. For the full scoop (and some pictures that illustrate how viewports work), see the Quirksmode article at *http://tinyurl.com/yyec93n*. And you'll learn more about media queries for mobile devices on page 261.

More Advanced Media Queries

Sometimes you might want your styles even more specific, so that they depend on multiple conditions. Here's an example:

```
@media (min-width: 400px) and (max-width: 700px) {
  /* These styles apply to windows from 400 to 700 pixels wide. */
}
```

This comes in handy if you want to apply several sets of mutually exclusive styles, but you don't want the headaches of several layers of overlapping rules. Here's an example:

```
/* Normal styles here */

@media (min-width: 600px) and (max-width: 700px) {
  /* Override the styles for 600-700 pixel windows. */
}

@media (min-width: 400px) and (max-width: 599.99px) {
  /* Override the styles for 400-600 pixel windows. */
}

@media (max-width: 399.99px) {
  /* Override the styles for sub-400 pixel windows. */
}
```

Now, if the browser window is 380 pixels, exactly two sets of style will apply: the standard styles and the styles in the final @media block. Whether this approach simplifies your life or complicates it depends on exactly what you're trying to accomplish. If you're using complex styles and changing them a lot, the no-overlap approach shown here is often the simplest way to go.

Notice that you have to take care that your rules don't unexpectedly overlap. For example, if you set the maximum width of one rule to 400 pixels and the minimum width of another rule to 400 pixels, you'll have one spot where both style settings suddenly combine. The slightly awkward solution is to use fractional values, like the 399.99 pixel measurement used in this example.

Another option is to use the *not* keyword. There's really no functional difference, but if the following style sheet makes more sense to you, feel free to use this approach:

```
/* Normal styles here */

@media (not max-width: 600px) and (max-width: 700px) {
  /* Override the styles for 600-700 pixel windows. */
}

@media (not max-width: 400px) and (max-width: 600px) {
  /* Override the styles for 400-600 pixel windows. */
}

@media (max-width: 400px) {
  /* Override the styles for sub-400 pixel windows. */
}
```

In these examples, there's still one level of style overriding to think about. That's because every @media section starts off with the standard, no-media-query style rules. Depending on this situation, you might prefer to separate your style logic completely (for example, so a mobile device gets its own, completely independent set of styles). To do so, you need to use media queries with external style sheets, as described next.

Replacing an Entire Style Sheet

If you have simple tweaks to make, the @media block is handy, because it lets you keep all your styles together in one file. But if the changes are more significant, you may decide that it's just easier to create a whole separate style sheet. You can then use a media query to create a link to that style sheet:

```
<head>
  <link rel="stylesheet" href="standard_styles">
  <link rel="stylesheet" media="(max-width: 480px)" href="small_styles.css">
  ...
</head>
```

The browser will download the second style sheet (*small_styles.css*) with the page but won't actually apply it unless the browser width falls under the maximum.

As in the previous example, the new styles will override the styles you already have in this place. In some cases, what you really want is completely separate, independent style sheets. First, you need to add a media query to your standard style sheet, to make sure it kicks in only for large sizes:

```
<link rel="stylesheet" media="(min-width: 480.01px)" href="standard_styles">
<link rel="stylesheet" media="(max-width: 480px)" href="small_styles.css">
```

The problem with this approach is that browsers that don't understand media queries will ignore both style sheets. You can fix this up for old versions of Internet Explorer by adding your main style sheet again, but with conditional comments:

```
<link rel="stylesheet" media="(min-width: 480.01px)" href="standard_styles">
<link rel="stylesheet" media="(max-width: 480px)" href="small_styles.css">
<!--[if lt IE 9]>
  <link rel="stylesheet" href="standard_styles">
<![endif]-->
```

This example still has one small blind spot. Old versions of Firefox (earlier than 3.5) don't understand media queries and don't use the conditionally commented IE section. You could solve the problem by detecting the browser in your code, and then using JavaScript to swap in a new page, but it's messy. Fortunately, old versions of Firefox are becoming increasingly rare.

Incidentally, you can combine media queries with the media types described in the box on page 256. When doing this, always start with the media type, and don't put it in parentheses. For example, here's how you could create a print-only style sheet for a specific page width:

```
<link rel="stylesheet" media="print and (min-width: 25cm)"
 href="NormalPrintStyles.css" >
<link rel="stylesheet" media="print and (not min-width: 25cm)"
 href="NarrowPrintStyles.css" >
```

Recognizing Mobile Devices

As you've already learned, you can distinguish between normal computers and mobile devices by writing a media query that uses max-device-width. But what widths should you use?

If you're looking for mobile phones, check for a max-device-width of 480 pixels. This is the best, more general rule. It catches the iPhone and the Android phones that exist today:

```
<link rel="stylesheet" media="(max-device-width: 480px)"
  href="mobile_styles.css">
```

If you're a hardware geek, this rule may have raised a red flag. After all, the current crop of mobile devices uses tiny, super-high-resolution screens. For example, the iPhone 4 crams a grid of 960 × 640 pixels into view at once. You might think you'd need larger device widths for these devices. Surprisingly, though, that isn't the case. Most web devices continue reporting that they have 480 pixels of width, even when they have a fancy high-resolution display. These devices add in a fudge factor called the *pixel ratio*. In the iPhone 4, for instance, every CSS pixel is two physical pixels wide, so the pixel ratio is 2. In fact, you can create a media query that matches the iPhone 4, but ignores older iPhones, using the following media query:

```
<link rel="stylesheet"
 media="(max-device-width: 480px) and (-webkit-min-device-pixel-ratio: 2)"
 href="iphone4.css">
```

The iPad poses a special challenge: Users can turn it around to show content vertically or horizontally. And although this changes the max-width, it doesn't alter the max-device-width. In both portrait and landscape orientation, the iPad reports a device width of 768 pixels. Fortunately, you can combine the max-device-width property with the orientation property if you want to vary styles based on the iPad's orientation:

```
<link rel="stylesheet"
 media="(max-device-width: 768px) and (orientation: portrait)"
 href="iPad_portrait.css">

<link rel="stylesheet"
 media="(max-device-width: 768px) and (orientation: landscape)"
 href="iPad_landscape.css">
```

Of course, this rule isn't limited to iPads. Other devices that have similar screens sizes (in this case, 768 pixels or less) will get the same style rules.

Note: On their own, media queries probably aren't enough to turn a normal website into a mobile-friendly one. You'll also need to think about bandwidth and the user experience. On the bandwidth side, you'll want to use smaller, lightweight images. (You can do this by giving elements background images, and setting these images in your styles. However, this approach is a nightmare for websites with lots of pictures.) On the user experience side, you need to think about breaking content down into smaller pieces (so less scrolling is required) and avoiding effects and interactions that are difficult to navigate with a touch interface (like pop-up menus).

GEM IN THE ROUGH

Media Queries for Video

One obvious difference between desktop websites and mobile websites is the way they use video. A mobile website may still include video, but it will typically use a smaller video window and a smaller media file. The reasons are obvious–not only do mobile browsers have slower, more expensive network connections to download video, they also have less powerful hardware to play it back.

Using the media query techniques you've just learned, you can easily change the size of a <video> element to suit a mobile user. However, it's not as easy to take care of the crucial second step, and link to a slimmed-down video file.

HTML5 has a solution: It adds a *media* attribute directly the <source> element. As you learned in Chapter 5, the <source> element specifies the media file a <video> element should play. By adding the media attribute, you can limit certain media files to certain device types.

Here's an example that hands the *butterfly_mobile.mp4* file out to small-screened devices. Other devices get *butterfly.mp4* or *butterfly.ogv*, depending on which video format they support.

```
<video controls width="400" height="300">
  <source src="butterfly_mobile.mp4"
   type="video/mp4"
   media="(max-device-width: 480px)">
  <source src="butterfly.mp4" type="video/mp4">
  <source src="butterfly.ogv" type="video/ogg">
</video>
```

Of course, it's still up to you to encode a separate copy of your video for mobile users. Encoding tools usually have device-specific profiles that can help you out. For example, they might have an option for encoding "iPad video." It's also still up to you to make sure that you use the right media format for your device (usually, that will be H.264), and supply video formats for every other browser.

Building Better Boxes

From the earliest days of CSS, web designers were using it to format boxes of content. As CSS became more powerful, the boxes became more impressive, creating everything from nicely shaded headers to floating, captioned figures. And when CSS cracked the hovering problem, floating boxes were even turned into rich, glowy buttons, taking over from the awkward JavaScript-based approaches of yore. With this is mind, it's no surprise that some of the most popular and best-supported CSS3 features can make your boxes look even prettier, no matter what they hold.

Transparency

The ability to make partially transparent pictures and colors is one of the most basic building blocks in CSS3. There are two ways to do it.

Your first option is to use the rgba() color function, which accepts four numbers. The first three values are the red, green, and blue components of the color, from 0 to 255. The final value is the *alpha*, a fractional value number from 0 (fully transparent) to 1 (fully opaque).

Here's an example that creates a 50 percent transparent lime green color:

```
.semitransparentBox {
  background: rgba(170,240,0,0.5);
}
```

Browsers that don't support rgba() will just ignore this rule, and the element will keep its default, completely transparent background. So the second, and better, approach is to start by declaring a solid fallback color, and then replace that color with a semitransparent one:

```
.semitransparentBox {
  background: rgb(170,240,0);
  background: rgba(170,240,0,0.5);
}
```

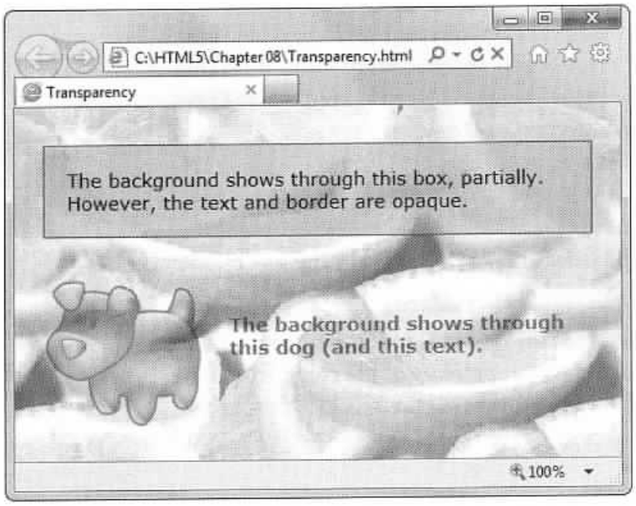
This way, browsers that don't support the rgba() function will still color the element's background, just without the transparency.

Tip: To make this fallback better, you should use a color that more accurately reflects the semitransparent effect. For example, if you're putting a semitransparent lime green color over a mostly white background, the color will look lighter because the white shows through. Your fallback color should reflect this fact, if possible.

CSS3 also adds a style property named *opacity*, which works just like the alpha value. You can set opacity to a value from 0 to 1 to make any element partially transparent:

```
.semitransparentBox {
  background: rgb(170,240,0);
  opacity: 0.5;
}
```

Figure 8-10 shows two example of semitransparency, one that uses the rgba() function and one that uses the opacity property.



The background shows through this box, partially. However, the text and border are opaque.

The background shows through this dog (and this text).

Figure 8-10: *This page serves up semitransparency two different ways: to fade out a picture (using the opacity property) and to let the background show through a box (using a background color created with the rgba() function).*

The opacity property is a better tool than the rgba() function if you want to do any of the following:

- Make more than one color semitransparent. With opacity, the background color, text color, and border color of an element can become transparent.
- Make something semitransparent, even if you don't know its color (for example, because it might be set by another style sheet or in JavaScript code).
- Make an image semitransparent.
- Use a transition, an animated effect that can make an element fade away or reappear (page 271).

Rounded Corners

You've already learned about the straightforward border-radius property, which lets you shave the hard corners off your boxes. But what you haven't yet seen is the way you can tweak this setting to get the curve you want.

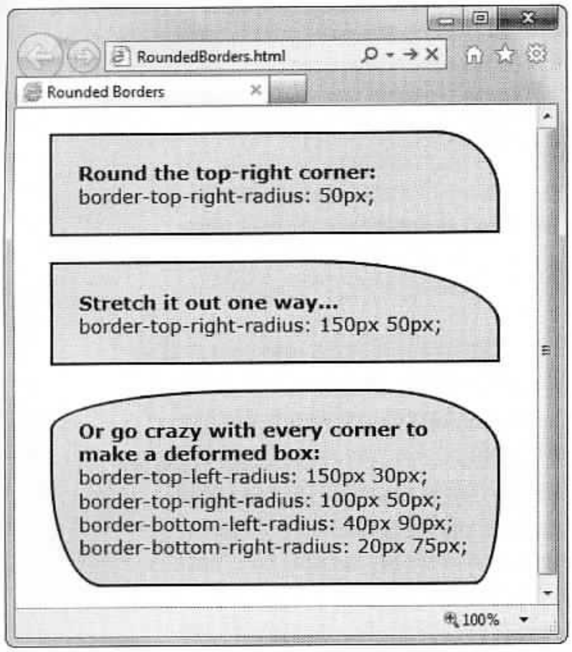
First, you can choose a different, single value for the border-radius property. The border radius is the radius of the circle that's used to draw the rounded edge. Of course, you don't see the entire circle—just enough to connect the vertical and horizontal sides of the box. Set a bigger border-radius value, and you'll get a bigger curve and a more gently rounded corner. As with most measurements in CSS, you can use a variety of units, including pixels and percentages. You can also adjust each corner separately by supplying four values:

```
.roundedBox {
  background: yellow;
  border-radius: 25px 50px 25px 85px;
}
```

But that's not all—you can also stretch the circle into an ellipse, creating a curve that stretches longer in one direction. To do this, you need to target each corner separately (using properties like *border-top-left-radius*) and then supply two numbers: one for the horizontal radius and one for the vertical radius:

```
.roundedBox {
  background: yellow;
  border-top-left-radius: 150px 30px;
  border-top-right-radius: 150px 30px;
}
```

Figure 8-11 shows some examples.



Round the top-right corner:
border-top-right-radius: 50px;

Stretch it out one way...
border-top-right-radius: 150px 50px;

Or go crazy with every corner to make a deformed box:
border-top-left-radius: 150px 30px;
border-top-right-radius: 100px 50px;
border-bottom-left-radius: 40px 90px;
border-bottom-right-radius: 20px 75px;

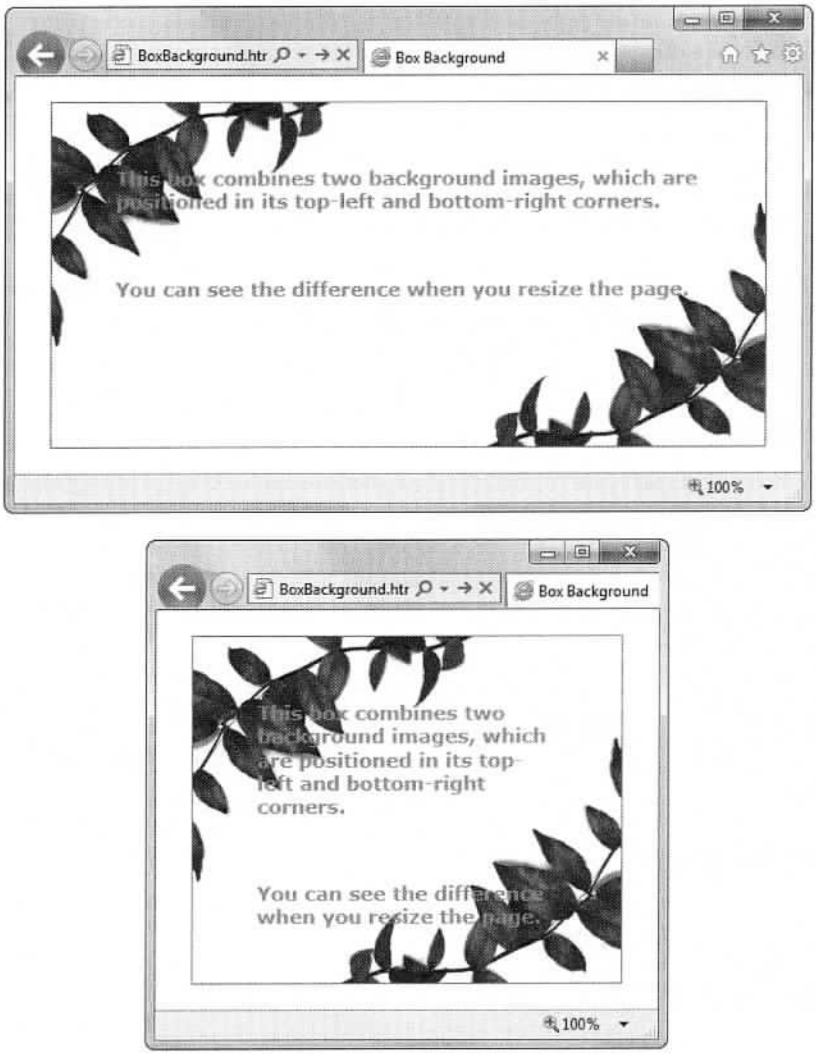
Figure 8-11:
A clever application of border-radius can create virtually any sort of curve.

Backgrounds

One shortcut to attractive backgrounds and borders is to use images. CSS3 introduces two new features to help out here. First is multiple background support, which lets you combine two or more images in a single element's background. Here's an example that uses two backgrounds to embellish the top-left and bottom-right corner of a box:

```
.decoratedBox {
  margin: 50px;
  padding: 20px;
  background-image: url('top-left.png'), url('bottom-right.png');
  background-position: left top, right bottom;
  background-repeat: no-repeat, no-repeat;
}
```

This first step is to supply a list with any number of images, which you use to set the *background-image* property. You can then position each image, and control whether it repeats, using the *background-position* and *background-repeat* properties. The trick is to make sure that the order matches, so the first image is positioned with the first background-position value, the second image with the second background-position value, and so on. Figure 8-12 shows the result.



This box combines two background images, which are positioned in its top-left and bottom-right corners.

You can see the difference when you resize the page.

Figure 8-12: *It doesn't matter how big this box grows—the two background images stay fixed at either corner.*

Note: If browsers don't support multiple backgrounds, they'll completely ignore your attempt to set the background. To avoid this problem, start by setting the background or background-image property with a fallback color or picture. Then, attempt to set multiple backgrounds by setting background-image with a list of pictures.

And here's a revised example that uses the *sliding doors* technique, a time-honored web design pattern that creates a resizable graphic out of three pieces: an image for the left, an image for the right, and an extremely thin sliver that's tiled through the middle:

```
.decoratedBox {
  margin: 50px;
  padding: 20px;
  background-image: url('left.png'), url('middle.png'), url('right.png');
  background-position: left top, left top, right bottom;
  background-repeat: no-repeat, repeat-x, no-repeat;
}
```

You could use markup like this to draw a background for a button. Of course, with all of CSS3's fancy new features, you'll probably prefer to create those using shadows, gradients, and other image-free effects.

Shadows

CSS3 introduces two types of shadows: box shadows and text shadows. Of the two, box shadows are generally more useful and better supported, while text shadows don't work in any version of Internet Explorer. You can use a box shadow to throw a rectangular shadow behind any <div> (but don't forget your border, so it still looks like a box). Shadows even follow the contours of boxes with rounded corners (see Figure 8-13).

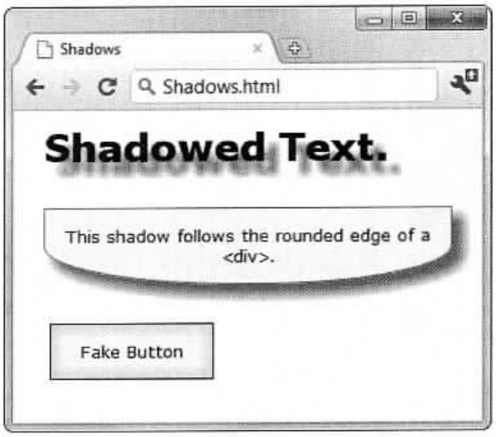


Figure 8-13:
Shadows can make text float (top), boxes pop out (middle), or buttons look glowy (bottom).

The two properties that make shadows work are *box-shadow* and *text-shadow*. Here's a basic box shadow example:

```
.shadowedBox {
  border: thin #336699 solid;
  border-radius: 25px;
  box-shadow: 5px 5px 10px gray;
}
```

The first two values set the horizontal and vertical offset of the shadow. Using positive values (like 5 pixels for both, in the above example) displaces the shadow down and to the right. The next value sets the *blur* distance (in this example, 10 pixels), which increases the fuzziness of the shadow. At the end is the shadow color. If there's any content underneath the box, consider using the rgba() function (page 185) to supply a semitransparent shadow.

If you want to tweak your shadow, you can tack on two details. You can add another number between the blur and the color to set the shadow *spread*, which expands the shadow by thickening the solid part before the blurred edge starts:

```
box-shadow: 5px 5px 10px 5px gray;
```

And you can add the word *inset* on the end to create a shadow that reflects inside an element, instead of outside. This works best if you use a shadow that's directly on top of the element, with no horizontal or vertical offset:

```
box-shadow: 0px 0px 20px lime inset;
```

This creates the bottom example in Figure 8-13. You can use inset shadows to add hover effects to a button (page 272).

Note: Shadow-crazy developers can even supply multiple shadows, by separating each one with a comma. But this is usually a waste of developer effort and computing power.

The text-shadow property requires a similar set of values, but in a different order. The color comes first, followed by the horizontal and vertical offsets, followed by the blur:

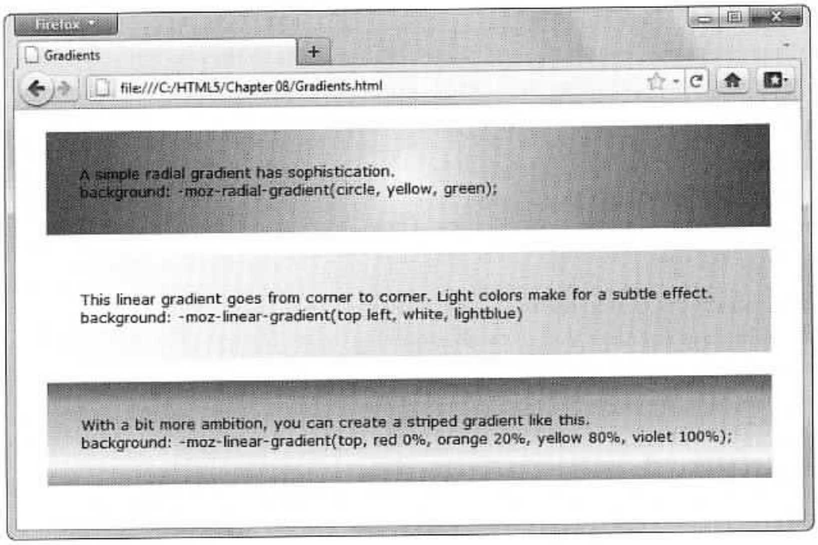
```
.textShadow  {
  font-size: 30px;
  font-weight: bold;
  text-shadow: gray 10px 10px 7px;
}
```

Gradients

Gradients are blends of color that can create a range of effects, from the subtle shading behind a menu bar to a psychedelic button that's colored like a 1960s revival party. Figure 8-14 shows some examples.

Note: Many web gradients are faked with background images. But CSS3 lets you define the gradient you want, and gets the browser to do the work. The advantage is fewer image files to schlep around and the ability to create gradients that seamlessly resize themselves to fill any amount of space.

You've already had some gradient-building experience with the canvas (page 208), and CSS3 gradients are similar. As with the canvas, CSS supports two types of gradients: linear gradients that blend from one band of color to another, and radial gradients that blend from a central point to the outer edges of your region.



A simple radial gradient has sophistication.
background: -moz-radial-gradient(circle, yellow, green);

This linear gradient goes from corner to corner. Light colors make for a subtle effect.
background: -moz-linear-gradient(top left, white, lightblue)

With a bit more ambition, you can create a striped gradient like this.
background: -moz-linear-gradient(top, red 0%, orange 20%, yellow 80%, violet 100%);

Figure 8-14: *At the heart of it, gradients are nothing more than blends of two or more colors. But that simple recipe cooks up into plenty of different dishes.*

There aren't any special CSS properties for creating gradients. Instead, you can use a gradient function to set the background property. Just remember to set the property to a solid color first to create a fallback for browsers that don't support gradients. (That includes Internet Explorer, which won't add gradient support until IE 10.)

There are four gradient functions, and they all need the awkward vendor prefixes you learned about on page 243. In this section, you'll look at examples that work on Firefox (and use the -moz- prefix). You'll need to add identical -webkit- and -o- versions to support Chrome, Safari, and Opera.

The first function is *linear-gradient()*. Here it is in one of its simpler forms, shading a region from white at the top to blue at the bottom:

```
.colorBlendBox {
  background: -moz-linear-gradient(top, white, blue);
}
```

Replace *top* with *left* to go from one side to another. Or, use a corner to blend diagonally:

```
  background: -moz-linear-gradient(top left, white, lightblue)
```

If you want multiple color bands, you simply need to supply a list of colors. Here's how you create a series of three horizontal color stripes:

```
  background: -moz-linear-gradient(top, red, orange, yellow);
```

Finally, you can control where each color starts (bumping some together or off to one side), using *gradient stops*. Each gradient stop is a percentage, with 0 percent being at the very start of the gradient and 100 percent being at the very end. Here's an example that extends the orangey-yellow section in the middle:

```
background: -moz-linear-gradient(top, red 0%, orange 20%, yellow 80%,
 violet 100%);
```

To get a radial gradient, you use the *radial-gradient()* function. You need to supply a color for the center of the circle and a color for the outer edge of the circle, where it meets the boundaries of the element. Here's a radial gradient that places a white point in the center and fades out to blue on the edges:

```
background: -moz-radial-gradient(circle, white, lightblue)
```

There are a pile more options that let you move the center point of the circle, stretch it into an ellipse, and change exactly where the colors fade. Different browser makers are still nailing down simple, consistent syntax that they can all use. For more gradient examples, and to see the two gradient functions not discussed here—that's repeating-linear-gradient() and repeating-radial-gradient()—check out the short Safari blog post at *www.webkit.org/blog/1424/css3-gradients*. Or, try an online Microsoft tool that lets you click-and-pick your way to the gradient you want, and then gives you markup that works with every browser, including IE 10. (Try it at *http://tinyurl.com/5rzocsk*.)

Tip: In all these examples, gradients were used with the background property. However, you can also use gradient functions to set the background-image property in exactly the same way. The advantage here is that background-image lets you use an image fallback. First, set background-image to a suitable fallback image for less-equipped browsers, and then set it again using a gradient function. Most browsers are smart enough that they won't download the gradient image unless they need it, which saves bandwidth.

Creating Effects with Transitions

CSS made every web developer's life a whole lot easier when it added *pseudoclasses* (page 389). Suddenly, with the help of *:hover* and *:focus*, developers could create interactive effects without writing any JavaScript code. For example, to create a hover button, you simply supply a set of new style properties for the *:hover* pseudoclass. These styles kick in automatically when the visitor moves the mouse pointer over your button.

Tip: If you're the last web developer on earth who hasn't rolled your own hover button, you can find a detailed tutorial in Creating a Website: The Missing Manual (O'Reilly), or in an online article at *www.elated.com/articles/css-rollover-buttons*.

Great as they are, pseudoclasses aren't cutting edge any longer. The problem is their all-or-nothing nature. For example, if you use the *:hover* pseudoclass, then your style settings spring into action immediately when someone hovers over an element. But in Flash application or in desktop programs, the effect is usually more refined. The hovered-over button may shift its color, move, or begin to glow, using a subtle animation that takes a fraction of a second to complete.

Some web developers have begun to add effects like these to their pages, but it usually requires the help of someone else's JavaScript animation framework. But CSS3 has a simpler solution—a new *transitions* feature that lets you smoothly switch from one group of settings to another.

A Basic Color Transition

To understand how transitions work, you need to see a real example. Figure 8-15 shows a color-changing button that's bolstered with some CSS3 transition magic.

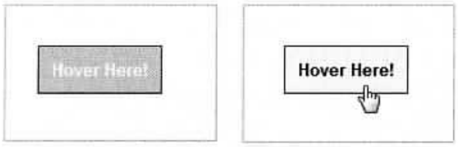


Figure 8-15:
If this were an ordinary rollover button, its background would jump from green to yellow in one step. But with transitions, the green blends into yellow, taking half a second to make the change. Move the mouse off, and the same transition plays out in reverse, returning the button to its normal state. The result is a button that just feels more polished.

Here's the nontransition way to style this button:

```
.slickButton {
  color: white;
  font-weight: bold;
  padding: 10px;
  border: solid 1px black;
  background: lightgreen;
  cursor: pointer;
}

.slickButton:hover {
  color: black;
  background: yellow;
}
```

Here's the button this markup formats:

```
<button class="slickButton">Hover Here!</a>
```

To smooth this change out with a transition, you need to set the *transition* property. You do this in the normal slickButton style (not the :hover pseudoclass).

At a minimum, every transition needs two pieces of information: the CSS property that you want to animate and the time the browser should take to make the change. In this example, the transition acts on the background property, and the duration is 0.5 seconds:

```
.slickButton {
  color: white;
  font-weight: bold;
  padding: 10px;
  border: solid 1px black;
  background: lightgreen;
  cursor: pointer;
  -webkit-transition: background 0.5s;
  -moz-transition: background 0.5s;
  -o-transition: background 0.5s;
}

.slickButton:hover {
  color: black;
  background: yellow;
}
```

As you'll no doubt notice, this example adds three transition properties instead of the promised one. That's because the CSS3 transitions standard is still under development, and the browsers that support it do it using vendor prefixes. To get your transition to work in Chrome, Safari, Firefox, and Opera, you need to set three versions of the same property—and you'll need to add another one with the -ms- prefix if Internet Explorer 10 supports transitions (as it's expected to do). Sadly, using experimental properties can make for some messy style sheets.

There's one quirk in this example. The hovered-over button changes two details: its background color and its text color. But the transition applies to the background color only. As a result, the text blinks from white to black in an instant, while the new background color fades in slowly.

There are two ways to patch this up. Your first option is to set the transition property with a comma-separated list of transitions, like this:

```
.slickButton {
  ...
  -webkit-transition: background 0.5s, color 0.5s;
  -moz-transition: background 0.5s, color 0.5s;
  -o-transition: background 0.5s, color 0.5s;
  ...
}
```

But there's a shortcut if you want to set transitions for all the properties that change, and you want to use the same duration for each one. In this case, you can simply add a single transition and use *all* for the property name:

```
  -webkit-transition: all 0.5s;
  -moz-transition: all 0.5s;
  -o-transition: all 0.5s;
```

Note: There are a few more details that can fine-tune a transition. First, you can choose a *timing function* that controls how the transition effect flows—for example, whether it starts slow and then speeds up or starts fast and then decelerates. In a short transition, the timing function you choose doesn't make much of a difference. But in a longer, more complex animation, it can change the overall feel of the effect. Second, you can also add a delay that holds off the start of the transition for some period of time. For more information about both, check out the official specification at *www.w3.org/TR/css3-transitions*.

Right now, transitions work in Opera 10.5, Firefox 4, and any version of Safari or Chrome you'll ever meet. They aren't supported in Internet Explorer (although they're planned for IE 10). However, this lack of support isn't the problem it seems.

Even if a browser ignores the transition property, it still applies the effect. It just makes the change immediately, rather than smoothly fading it in. That's good news—it means a website can use transitions and keep the essentials of its visual style intact on old browsers.

More Transition Ideas

It's gratifying to see that CSS transitions can make a simple color change look good. But if you're planning to build a slick rollover effect for your buttons or menus, there are plenty of other properties you can use with a transition. Here are some first-rate ideas:

- **Transparency.** By modifying the opacity property, you can make an image fade away into the background. Just remember not to make the picture completely transparent, or the visitor won't know where to hover.
- **Shadow.** Earlier, you learned how the box-shadow property can add a shadow behind any box (page 268). But the right shadow can also make a good hover effect. In particular, consider shadows with no offset and lots of blur, which create more of a traditional glow effect. You can also use an inset shadow to put the effect inside the box.
- **Gradients.** Change up a linear gradient or add a radial one—either way, it's hard to miss this effect.
- **Transforms.** As you'll learn in the next section, transforms can move, resize, and warp any element. That makes them a perfect tool for transitions.

On the flip side, it's usually not a good idea to use transitions with padding, margins, and font size. These operations take more processing power (because the browser needs to recalculate layout or text hinting), which can make them slow and jerky. If you're trying to make something move, grow, or shrink, you're better off using a transform, as described next.

Transforms

When you explored the canvas, you learned about *transforms*—ways to move, scale, skew, and rotate content. In the canvas, you can use transforms to change the things you draw. With CSS3 transforms, you use them to change the appearance of an element. Like transitions, transforms are a new and experimental feature. To use them, you need to stack up the vendor-prefixed versions of the *transform* property. Here's an example that rotates an element and all its contents:

```
.rotatedElement {
  -moz-transform: rotate(45deg);
  -webkit-transform: rotate(45deg);
  -o-transform: rotate(45deg);
}
```

WORD TO THE WISE

Don't Leave Old Browsers Behind

As you know, browsers that don't support transitions switch between states immediately, which is usually a good thing. However, if you use CSS3 glitter to make your states look different (for example, you're adding a shadow or a gradient to a hovered-over button), old browsers ignore that too. That's not so good. It means that visitors with less capable browsers get *no* hover-over effect at all.

To solve this problem, use a fallback that older browsers understand. For example, you might create a hover state that sets a different background color and *then* sets a gradient. This way, older browsers will see the background change to a new solid color when the button is hovered over. More capable browsers will see the background change to a gradient fill. For even more customizing power, you can use Modernizr, which lets you define completely different styles for older browsers (page 240).

In the previous example, the rotate() function does the work, twisting an element 45 degrees around its center. However, there are many more transform functions that you can use, separately or at the same time. For example, the following style chains three transforms together. It enlarges an element by half (using the *scale* transform), moves it to 10 pixels to the left (using the *scaleX* transform), and skews it for effect (using the *skew* transform):

```
.rotatedElement {
  -moz-transform: scale(1.5) scaleX(10px) skew(10deg);
  -webkit-transform: scale(1.5) scaleX(10px) skew(10deg);
  -o-transform: scale(1.5) scaleX(10px) skew(10deg);
}
```

Note: A skew twists an element out of shape. For example, imagine pushing the top edge of a box out to the side, while the bottom edge stays fixed (so it looks like a parallelogram). To learn more about the technical details of transform functions, check out Firefox's helpful documentation at *http://tinyurl.com/6ger2wp*, but don't forget to add the other vendor prefixes if you want it to work in Chrome and Opera.

Transforms don't affect other elements or the layout of your web page. For example, if you enlarge an element with a transform, it simply overlaps the adjacent content.

Transforms and transitions make a natural pair. For example, imagine you want to create an image gallery, like the one shown in Figure 8-16.

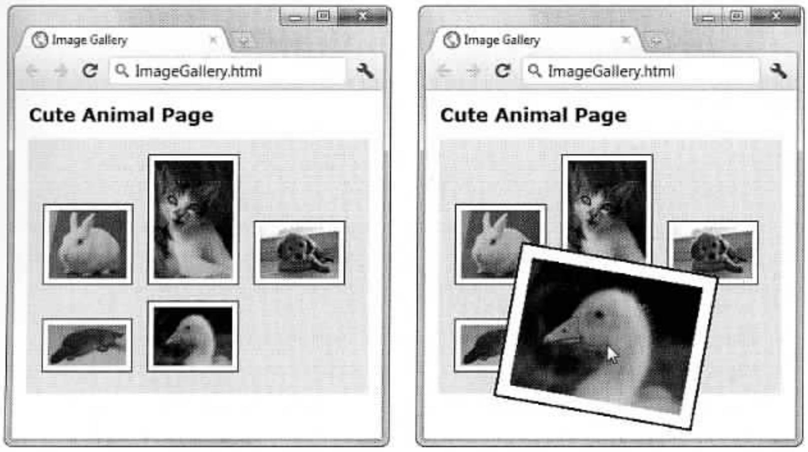


Figure 8-16:
Here, a transform makes the hovered-over image stand out.

This example starts out simple enough, with a bunch of images wrapped in a <div> container:

```
<div class="gallery">
  <img src="bunny.jpg">
  <img src="cat.jpg">
  <img src="dog.jpg">
  <img src="platypus.jpg">
  <img src="goose.jpg">
</div>
```

Here's the style for the <div> that holds all the images:

```
.gallery {
  margin: 0px 30px 0px 30px;
  background: #D8EEFE;
  padding: 10px;
}
```

And here's how each <img> element starts off:

```
.gallery img {
  margin: 5px;
  padding: 5px;
  width: 75px;
  border: solid 1px black;
  background: white;
}
```

You'll notice that all the images are given explicit sizes with the width property. That's because this example uses slightly bigger pictures that are downsized when they're shown on the page. This technique is deliberate: It makes sure the browser has all the picture data it needs to enlarge the image with a transform. If you didn't take this step, and used thumbnail-sized picture files, the enlarged version would be blurry.

Now for the hover effect. When the user moves the mouse over an image, the page uses a transform to rotate and expand the image slightly:

```
.gallery img:hover {
  -webkit-transform: scale(2.2) rotate(10deg);
  -moz-transform: scale(2.2) rotate(10deg);
  -o-transform: scale(2.2) rotate(10deg);
}
```

Right now, this transform snaps the picture to its new size and position in one step. But to make this effect look more fluid and natural, you can define an all-encompassing transition in the normal state:

```
.gallery img {
  margin: 5px;
  padding: 5px;
  width: 75px;
  border: solid 1px black;
  -webkit-transition: all 1s;
  -moz-transition: all 1s;
  -o-transition: all 1s;
  background: white;
}
```

Now the picture rotates and grows itself over 1 second. Move the mouse away, and it takes another second to shrink back to its original position.

POWER USERS' CLINIC

The Future of CSS-Powered Effects

The examples on these pages just scratch the surface of what transforms and transitions can do. Although these features are far from being finalized, you can use several experimental features to extend them further:

- **3-D transforms.** When you get tired of moving an element around in two dimensions, you can use 3-D transforms to move, rotate, and warp it in three-dimensional space. The creators of Safari have a brief walkthrough at *www.webkit.org/blog/386/3d-transforms*.
- **Animations.** Right now, transitions are limited to fairly simple interactions–mostly, that's when someone hovers with the mouse (using the *:hover* pseudoclass) or changes focus to an input control (using *:focus*). The animation feature extends where you can use transitions, allowing you to apply them dynamically, in response to a JavaScript event. For example, you might create a rotation effect that kicks in when you click a button. You can read the specification at *www.w3.org/TR/css3-animations*.
- **A sprinkling of JavaScript.** Add a little bit of code to dynamically turn styles on and off, and you can build complex chunks of user interface, like a 3-D image carousel or a collapsible group of panels (often called an accordion control). To see some examples in action, visit *http://css3.bradshawenterprises.com*.

Right now, none of these features are worth the trouble. First, they require lots of messy vendor-specific prefixes, which makes mistakes all too easy and forces you to test the page on every mainstream browser. Second, these features aren't available on many of today's browsers. Animations, for instance, aren't supported on any version of IE or Opera (at the time of this writing), or on Firefox 4. And trying to add a workaround is more work than using a different approach from the start.

Today, the best practical solution for animated effects is a JavaScript library like jQuery UI or MooTools. But CSS3 is the clear future of web effects, once the standards settle down and modern browsers have colonized the computers of the world.