```
border-radius: 10px 5px;
```

The following three parameters set the radius of the top-left corner to 10 pixels, the top-right and bottom-left corners to 5 pixels, and the bottom-right corner to 3 pixels:

```
border-radius: 10px 5px 3px;
```

The following four parameters set the radius of the top-left corner to 10 pixels, the top-right corner to 5 pixels, the bottom-right corner to 3 pixels, and the bottom-left corner to 1 pixel:

```
border-radius: 10px 5px 3px 1px;
```

When you're able to set the radius of each individual corner or pairs of corners, you can create quite a few different special effects, such as dialog bubbles or ellipses.

You can also set the individual corner radii values independently from one another with a few additional properties:

- `border-top-left-radius`
- `border-top-right-radius`
- `border-bottom-left-radius`
- `border-bottom-right-radius`

Each one sets the corresponding border radius value in the element.

The `border-radius` properties all use a size value to set the circle radius for the corner. You can specify the size using any of the standard CSS size unit measurements, such as inches, pixels, or em units.

Using Border Images

The default border line that HTML5 places around objects is pretty dull. How about adding some more elaborate borders around objects? You can, thanks to another interesting feature added to CSS3. It provides the ability to use images for the border around elements instead of just a line. This feature allows you to use any type of image to create a flourish around your elements.

You apply a border image to an element by adding the `border-image` property:

```
border-image: url(file) slice repeat
```

The `url()` function defines the location of the image file used for the border. The path can be either an absolute value pointing directly to the image file or a relative path (relative to the location of the CSS script).

The *`slice`* value defines what parts of the border image to use for the border. This part can get somewhat complicated. By default, the browser slices the border image into nine sections, as shown in Figure 4-2. The nine border image sections are

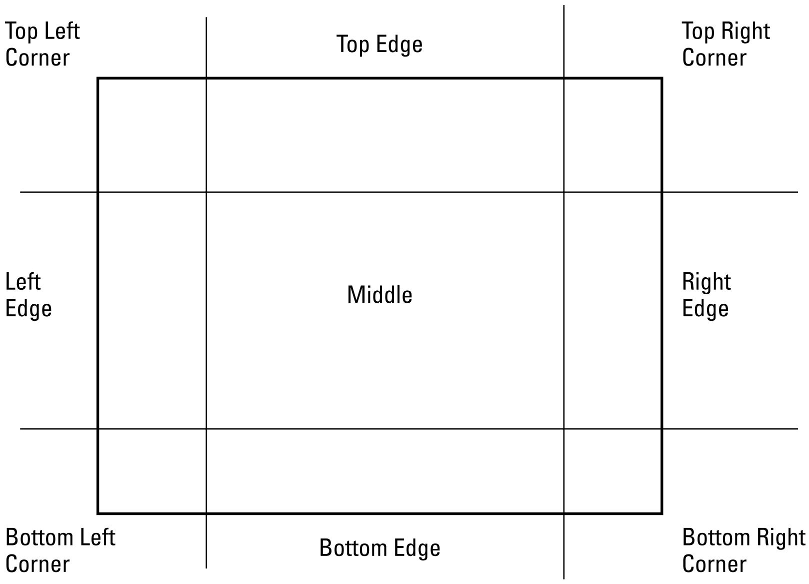


FIGURE 4-2: Slicing a border image to retrieve the pieces.

- The four corner pieces (top left, top right, bottom right, and bottom left)
- The four edge pieces (top, right, bottom, and left)
- The middle section

For the `slice` value, you specify the size of the image pieces to use for the individual border images. You can specify that as either a percentage of the entire image size, or a pixel value to represent how much of the image edges to use for the border edges. You have the option to specify the `slice` as one, two, or four separate values:

- **One value:** Cuts the same size of the image for the four corners and the four edges
- **Two values:** One size for the top and bottom, and another size for the left and right sides
- **Four values:** One size each for the top, right side, bottom, and left side

The *`repeat`* parameter defines how the browser should make the image fit the space required to create the border edges. There are four ways to do that:

- `repeat`: Repeats the image to fill the entire edge
- `round`: Repeats the image, but if the image doesn't fit the area as a whole number of repeats, rescales the image so it fits
- `space`: Repeats the image, but if the image doesn't fit the area as a whole number of repeats, adds spaces between the images so it fits
- `stretch`: Stretches the image to fill the edge

So, for example, to define a border image that uses 10-pixel slices from all the sides, and stretches them to fit the border area, you'd use the following:

```
border-image: url("myimage.jpg") 10 stretch;
```

Note that you don't use the units for the `slice` value. If you specify the value as a percentage of the entire image, add the percent sign, but if it's in pixels, leave off the `px`.

Instead of using one property statement for all the features, if you prefer, you can define these values in separate properties. There are five separate properties used to define the border image, and how the browser should use it (see Table 4-1).

TABLE 4-1 **The CSS4 Border Image Properties**

Property	Description
`border-image-outset`	Specifies the amount the image extends beyond the normal border box area
`border-image-repeat`	Specifies how the image should be extended to fit the entire border area
`border-image-slice`	Specifies what piece of the image to use as the border
`border-image-source`	Specifies the path to the image used for the border
`border-image-width`	Specifies the widths of the border image sides

Figure 4-3 shows what the border image looks like around an element. That's quite a bit better than the standard border line.

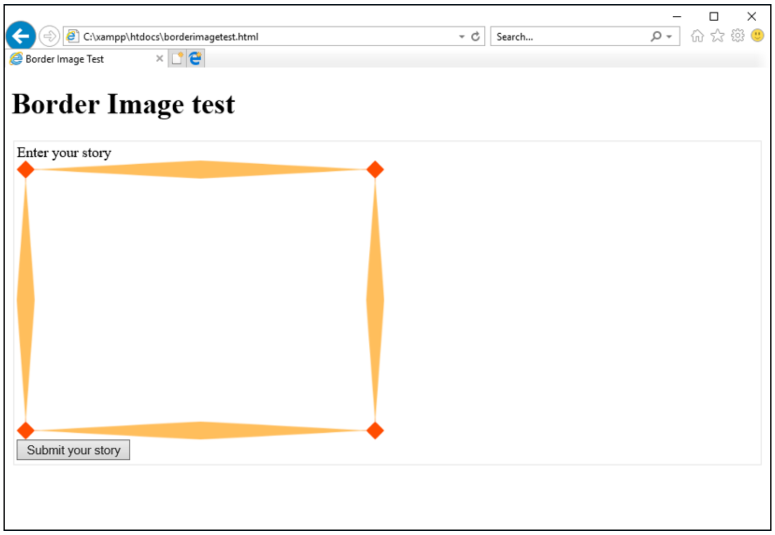


FIGURE 4-3: Using a border image around an element.

TIP

The Mozilla Foundation developers' website includes a handy border image generator tool: `https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Background_and_Borders/Border-image_generator`. With this tool, you can upload an image or use one of their standard images, and the tool will automatically generate the CSS3 code necessary to extract the border image properties.

Looking at the CSS3 Colors

In Book 2, Chapter 2, I show you the three formats that the original CSS standard defines for setting colors in the web page:

- Using a color name
- Using an RGB hexadecimal value
- Using the `rgb( )` function with decimal values

The CSS3 standard extends the options you have available for defining colors by adding the hue, saturation, and lightness (HSL) method. The HSL method of defining colors uses three values:

- **Hue:** The degree of color on the color wheel. The color wheel concept has been around in the art world since the early 1800s. It places the colors around a circle with the primary colors — red, yellow, and blue — positioned on the wheel at 0, 60, and 240 degrees, respectively. The secondary colors — orange, green, and violet — are placed in between the primary colors, in locations based on their shades at 30, 120, and 260 degrees, respectively. From there, the different shades of color combinations are arranged appropriately on the wheel. To specify an individual color hue, you must know its location on the color wheel. Fortunately there are plenty of charts online to help out with that.
- **Saturation:** The percentage of the color used. The saturation value is a percentage that specifies the grayness shade of the color, from 0 percent for no color (all gray) to 100 percent for full color saturation.
- **Lightness:** The percentage of lightness added to the color. The lightness value is a percentage that specifies how dark (0 percent) or light (100 percent) the color should be. The 50 percent value creates the color at its normal shade. Larger percentages create darker shades of the color, while smaller percentages create lighter shades of the color.

To use the HSL method to specify a color, use the `hsl()` format. For example, the following property specifies the red color at position 0 of the color wheel, shown at 100 percent saturation, with 50 percent lightness:

```
color: hsl(0, 100%, 50%);
```

The CSS3 standard also adds the opacity feature to HSL, creating the *HSLA color method.* With HSLA, you add a fourth parameter to specify the opaqueness of the color, from 0 to 1. The following example uses the red color, but at 50 percent transparency:

```
color: hsla(0, 100%, 50%, 0.5)
```

TIP

The beauty of using the HSL values comes when you're choosing a color scheme for your website. If you want to use a single color for the website scheme, you can modify the saturation and lightness levels to make different shades of the color. If you want to create a two-color scheme, you may want to choose hues that are 180 degrees apart — those are considered complementary. For a three-color scheme, hues that are 120 degrees apart create a triad. In a four-color scheme, select hues that are 90 degrees apart to create a nice offset. By sticking with the color wheel rules, just about anyone can create a tasteful color scheme for a website.

Playing with Color Gradients

While using individual colors are a great way to liven up the website, even colors can get somewhat boring when you use them all the time. To help make things more interesting, the CSS3 standard adds color gradients to the mix. A *color gradient* slowly fades from one color into a second color, producing a warm transition effect. These transition colors are often used for backgrounds, creating an effect that helps the website visitor follow the content as the color gradient morphs into a different color.

There are two types of color gradients defined in the CSS3 standard:

- **Linear gradients:** Fade using a side-to-side or top-to-bottom direction
- **Radial gradients:** Use a center point and fade outward (radiate) from there, much like a tie-dyed T-shirt.

This section discusses how to use each of these methods in your web pages.

Linear gradients

A linear gradient fades between two colors in a linear manner — that is, from side to side, or from top to bottom. Use the `linear-gradient()` function to define the direction of the fade and the transition colors:

```
linear-gradient(direction, color1, color2);
```

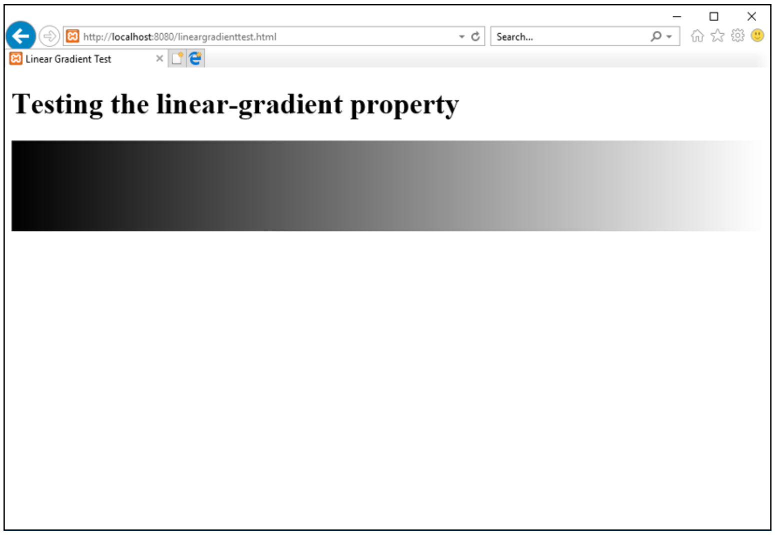
The `direction` parameter defines which way the gradient should go. If you omit the direction, the browser will create the gradient from top to bottom, a common effect for backgrounds. If you want to change the direction, specify it by the direction that the gradient should fade from *`color1`* to *`color2`*, like this:

```
linear-gradient(to right, black, white)
```

You can use `to top`, `to bottom`, or `to right` to specify the direction of the gradient. This example starts with the black color on the left side and fades to the white color on the right side, as shown in Figure 4-4.

To use the linear gradient, just add it anywhere you'd use a color value:

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



Testing the linear-gradient property

FIGURE 4-4: A left-to-right color gradient.

Linear gradients can create quite the stunning effect for web page backgrounds. If you want to get fancy, you can specify more colors in the `linear-gradient()` list as intermediate points between the two endpoints:

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

This takes the color transition from a red to an orange first and then finally to the yellow destination.

Radial gradients

The `radial-gradient()` does the same thing as the linear gradient, but in a circular pattern radiating from a central point. If you have fond memories of the days when tie-dyed T-shirts were popular, you may love the radial gradients!

Here's the format for the `radial-gradient()` function:

```
radial-gradient(shape size, color1, color2, ...)
```

The keys to creating the radial gradient are the *`shape`* parameter, which defines the shape of the gradient, and the *`size`* parameter. By default the radial gradient is drawn as an ellipse, but you can instead specify a circle. The size determines where the radial gradient stops. Usually this is a location, such as `closest-corner`, `closest-side`, `farthest-corner`, or `farthest-side`.

You'll also want to define two or more colors to create the gradient effect in the image. The simplest way to define a radial gradient is to just define the colors:

```
background: radial-gradient(red, orange, yellow);
```

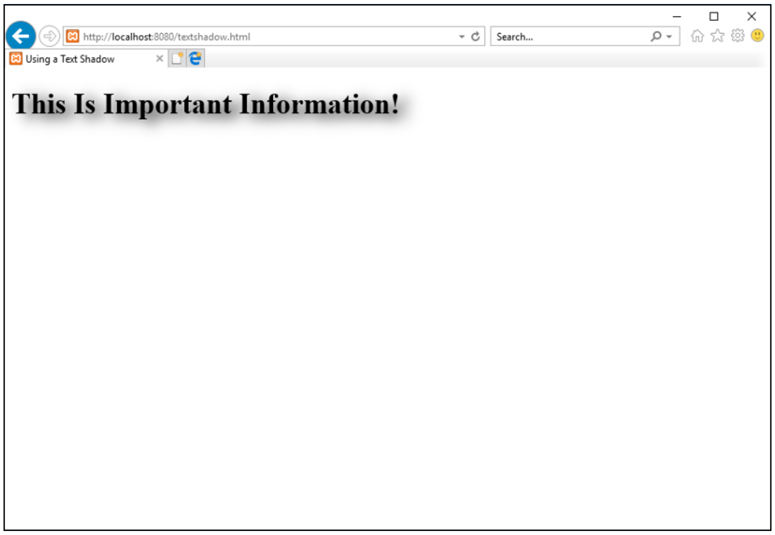
This creates an elliptical radial gradient, centered in the element, radiating outward toward the farthest corner.

Adding Shadows

Yet another cool feature added in CSS3 is the ability to create shadows of elements on the web page. Shadows allow you to produce the effect of a light shining down on the web page. You can place shadows behind both text and box elements.

Text shadows

Placing shadows behind text on a web page can create a startling effect to draw attention to headings. Figure 4-5 shows how the text shadow effect can make the heading stand out on the web page.



This Is Important Information!

FIGURE 4-5: Adding the text shadow effect to a heading element.

The CSS3 `text-shadow` style property allows you to define just how the shadow should look. Here's the format of the `text-shadow` style property:

```
text-shadow: color offsetx offsety blur;
```

The *`color`* parameter defines the color to use for the shadow. The *`offsetx`* and *`offsety`* parameters define the distance of the shadow from the text. You can use either positive or negative values to represent the offset values. Positive values move the shadow down and to the right of the text. Negative values move the shadow up and to the left of the text. The *`blur`* parameter defines the amount of space the shadow uses. The larger the space, the more stretched looking the shadow appears.

Here's an example of a CSS3 rule that sets a shadow for all `h1` elements:

```
h1 {
    text-shadow: black, 10px, 5px, 15px;
}
```

This produces a black shadow to the right and below the text.

You can apply more than one shadow to a text element. Just list the different shadow definitions on the same `text-shadow` line, separated by commas:

```
text-shadow: shadow1, shadow2, ...;
```

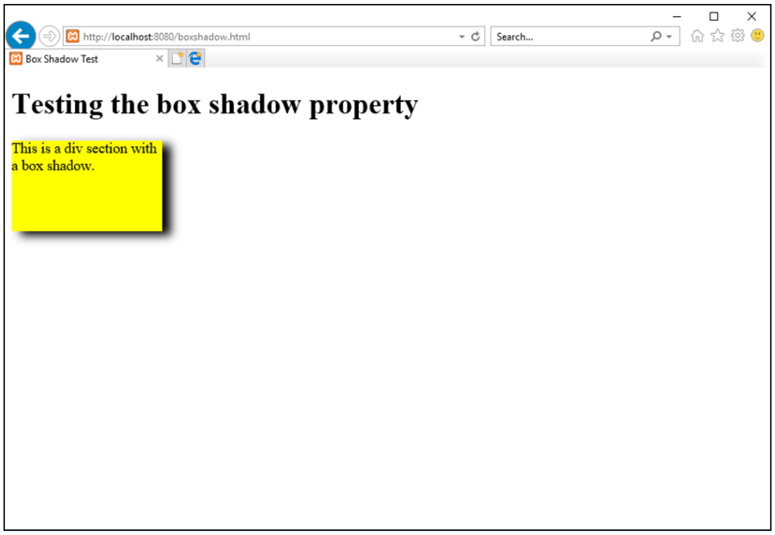
The browser displays the shadows in the order you define them, with each shadow placed on top of the previous shadows.

Box shadows

The box shadow helps the element stand out with almost a 3-D effect on the web page. Box shadows work the same way as text shadows, but you apply them to box elements, such as individual form input fields, text areas, or even entire div blocks. Figure 4-6 shows an example of applying a simple box shadow to a div section on the web page.

The format for the `box-shadow` property is similar to the `text-shadow` property, with a couple of added things:

```
box-shadow: [inset] color offsetx offsety blur [spread];
```



Testing the box shadow property

This is a div section with a box shadow.

FIGURE 4-6: Using a box shadow on a div element.

The `inset` keyword is optional. It determines whether the browser should display the shadow inside the element. By default, the size of the shadow is the same as the object; by adding the *`spread`* value, you can increase or decrease the size of the shadow.

Creating Fonts

In Chapter 2 of this minibook, I mention the problem with fonts on a web page. In the past, browsers were only able to use fonts that were already installed on the workstation. Finding fonts that are available on all workstations is somewhat of a challenge.

The CSS3 standard has attempted to remedy this situation by providing a way for web designers to create their own fonts and deliver them to their site visitors as part of the web page download. The `@font-face` rule provides a way to specify a font file that the client browser must download as part of the style definitions. When the browser downloads the font file, your web application can use that font to style text in the web pages. These fonts are known as *web fonts*.

The following sections describe the different types of web fonts and how to use them in your web applications.

Focusing on font files

The key to using web fonts is the ability to define a font in a file that every site visitor's browser can download and use. The *font files* contain detailed information on how the workstation should display individual characters and symbols.

The problem with font files, though, is that, over the years, lots of different font file formats have appeared. Table 4-2 shows the popular font file formats you may run into.

TABLE 4-2 Font File Formats

Font	Description
TrueType	A font created in the 1980s by Microsoft and Apple. This font type is still commonly used by both operating systems.
OpenType	Created by Microsoft and built to extend TrueType fonts. The most common font type used.
Embedded OpenType	A font format created by Microsoft for use only in the Internet Explorer web browser.
Scalable Vector Graphics (SVG)	Primarily used for graphics on mobile devices, but can be used to display text.
Web Open Font Format (WOFF)	A font created by the W3C standards group, intended for web pages.

The TrueType and WOFF font file formats are currently the only two supported by all browsers. It's best to stick with one of these types of fonts when creating your web fonts.

WARNING

Font files can often be found and downloaded from the Internet. However, beware of licensing restrictions on font files. Most font files are not free, or are free only for personal use.

Working with web fonts

CSS3 allows you to define a web font file for client browsers to download using the `@font-face` rule. You may notice that the `@font-face` rule doesn't follow any of the standard style rule-naming conventions that I discuss in Chapter 2 of this minibook. There's a reason for that. The `@font-face` rule defines a *CSS command.* CSS commands are directives to the browser to perform some action while loading the styles. CSS commands start with the at symbol (`@`) and should be placed at the start of the CSS stylesheet area.

Here's the format of the `@font-face` rule:

```
@font-face {
    font-family: name;
    src: url(location);
    [descriptor:value];
}
```

The `font-family` property defines a unique name for the font in your stylesheet. The `src` property defines the location of the font file on your server, either as an absolute or relative path.

Following those two properties, you can add *descriptors* that indicate when the font should be used (such as for bold text or for text in italics).

An example of defining a web font would be:

```
@font-face {
    font-family: myfont;
    src: url(myfont.woff);
}
```

This defines a font family named `myfont` from the `myfont.woff` font file that the client workstation should download. Then, to use the new font in your web pages, just define the `font-family` name in a style rule:

```
div {
    font-family: myfont;
}
```

There are three descriptors that you can define for the web font:

- `font-stretch`: Specifies how the font should be stretched to fill a space. The default is normal, but other values are condensed or expanded.
- `font-style`: Specifies how the font should be styled. The values are normal, italic, or oblique.
- `font-weight`: Specifies the boldness of the font. The values are normal, bold, or numeric values from 100 to 900.

By specifying different `font-style` and `font-weight` values, you can specify more than one font file, depending on how you use the font in the web page.

Handling Media Queries

These days, it's likely that your web applications will be viewed by site visitors using a myriad of devices. Whether it's on a large monitor connected to a desktop workstation or a small mobile device that fits in the palm of your hand, your web application will need to be presentable to all your website visitors.

The CSS3 standard has some tricks that you can use to help determine just when you need to alter the style and layout of your web pages, based on how your site visitor is viewing the application. This section covers just how to use those tricks.

Using the @media command

The CSS2 standard defined the `@media` CSS command to help you detect what type of device the web page is being viewed on. You can then create styles based on the media type. This allows you to style the web page one way when your site visitor is displaying it on a monitor screen and another way when the site visitor prints it out.

The CSS2 standard defined several different media types to use in the `@media` rule, as shown in Table 4-3.

TABLE 4-3 **The CSS2 @media Types**

Type	Description
all	All types of output devices
braille	Devices that produce Braille
embossed	Braille printers
handheld	Mobile devices with small screens
print	Printers
projection	Large-screen projectors
screen	Standard computer monitors
speech	Text-to-speech readers
tty	Teletype terminals
tv	Television

You use the `@media` command in your standard style sheet to define styles used for that specific type of device:

```
@media screen {
    body {
        font-family: sans-serif;
        font-size: 12pt;
    }
    h1 {
        font-family: sans-serif;
        font-size: 20pt;
    }
}

@media print {
    body {
        font-family: serif;
        font-size: 10pt;
    }
    h1 {
        font-family: serif;
        font-size: 18pt;
    }
}
```

These two `@media` commands define two sets of style rules — one for when the web page appears on a monitor, and one for when the web page is printed. It's up to the browser to determine which situation dictates which `@media` command set to use.

Dealing with CSS3 media queries

The CSS2 `@media` command went a long way toward helping you determine what types of devices your site visitors are using to display your web application, but it didn't go quite far enough. For example, whether your site visitor is viewing your web application on a big monitor or a small mobile device, the device evaluates to the `screen` media type by the `@media` command.

The CSS3 standard solves that problem by adding *media queries* to the standard `@media` commands. Media queries allow you to query the features supported by the client browser and the device the browser is running on. You can add the media queries to the standard `@media` commands to produce a customized rule set for just about any type of circumstance.

Here's the format of the media query:

```
@media type and feature
```

The *type* parameter defines the media type, similar to the CSS2 media types, but now limits them to only four (`all`, `print`, `screen`, and `speech`). The *feature* parameter defines new features available to query. The CSS3 media features available are shown in Table 4-4.

TABLE 4-4 The CSS3 Media Features

Feature	Description
`any-hover`	Whether the device supports hovering a pointer over elements
`any-pointer`	Whether the device supports a pointing device
`aspect-ratio`	The height and width ratio of the viewing device
`color`	The number of bits of color supported by the viewing device
`color-index`	The number of colors the device can display
`grid`	Whether the device supports a grid or a bitmap
`height`	The height of the viewing area of the device
`hover`	Whether the device supports hovering a pointer over elements
`inverted-colors`	Whether the browser is capable of inverting colors
`light-level`	The current ambient light level
`max-aspect-ratio`	The maximum ratio between the width and height of the viewing area
`max-color`	The maximum number of bits of color supported by the viewing area
`max-color-index`	The maximum number of colors the device supports
`max-device-aspect-ratio`	The maximum ratio between the width and height of the device
`max-device-height`	The maximum height of the device
`max-device-width`	The maximum width of the device
`max-height`	The maximum height of the device viewing area
`max-monochrome`	The maximum number of bits in a monochrome setting
`max-resolution`	The maximum resolution of the device

(continued)

TABLE 4-4 ***(continued)***

Feature	Description
`max-width`	The maximum width of the device
`min-aspect-ratio`	The minimum ratio between the width and height of the viewing area
`min-color`	The minimum number of bits of color supported by the viewing area
`min-color-index`	The minimum number of colors the device supports
`min-device-aspect-ratio`	The minimum ratio between the width and height of the device
`min-device-height`	The minimum height of the device
`min-device-width`	The minimum width of the device
`min-height`	The minimum height of the device viewing area
`min-monochrome`	The minimum number of bits in a monochrome setting
`min-resolution`	The minimum resolution of the device
`min-width`	The minimum width of the device
`monochrome`	The number of bits of color in a monochrome setting
`orientation`	The orientation (landscape or portrait) of the device
`overflow-block`	How the device handles overflowing block elements
`overflow-inline`	How the device handles overflowing inline elements
`pointer`	Whether the device supports a pointing device
`resolution`	The resolution of the device
`scan`	Whether the device uses progressive or interlaced scanning
`scripting`	Whether the device supports client-side scripting languages
`update-frequency`	How quickly the device can update the viewing area
`width`	The width of the device viewing area

Table 4-4 shows lots of different device features you can test to customize the styles you apply to your web page. An example looks like this:

```
@media screen and (max-width: 1000px) {
    font-size: 12px;
}
```

```
@media screen and (max-width: 500px) {
    font-size: 10px;
}
```

The first rule only applies to devices that have a maximum viewing area width of 100 pixels. It uses the 12-pixel font size for the text on the web page. The second rule only applies to devices that have a maximum viewing area of 500 pixels (such as a mobile device). It uses the 10-pixel font size for the text on the web page to make it smaller.

Applying multiple style sheets

You can also use the media types and features queries in the `<link>` tag to reference specific external style sheets depending on the media features. This allows you to apply entirely different style sheets to the web page based on the device your site visitor is using to view it. Here's the format for doing that:

```
<link rel="stylesheet" href="desktop.css" media="screen and (max-width:500px)">
```

Now the browser will apply the `desktop.css` external style sheet only if the device has a maximum viewing area width of 500 pixels.

TIP

It's always a good idea to have separate style sheets for mobile devices for your web application. Usually, you'll need to change the layout of navigation buttons to make them easily accessible on the mobile device, as well as limit the content that you display in the web page.