Holzschlag_03.qxd 3/30/05 9:13 AM Page 31

CHAPTER 3

Adding Images, Media, and Scripts

Images, media, and scripting help a site become "dynamic and rich." This is called *dynamic* because many of these features offer the opportunity for the site visitor to interact in an active way with the site. It's *rich* because the site becomes richer visually and in terms of content. *Images* in this chapter refers specifically to images you'll be adding to your page as part of the design itself or as a means of enhancing the content, such as with photos. Images must be processed in a specific way for the Web, using a good image editor; you can quickly learn the details.

NOTE
Web graphics can be created by a wide range of programs, but typically you're going to want to have a decent image editor, such as Photoshop (if your wallet is a little smaller, you can try Jasc's Paint Shop Pro). There are numerous other web graphic programs; you can find them by searching for "web graphics software" at your favorite search engine.

Two primary types of web graphic formats exist: GIF and JPEG. The GIF file format is best for images with few, flat colors and line drawings; JPEGs are best for images with many colors and color gradients, such as photos. A third type of web graphic format is PNG, but the lack of support for PNG in some browsers makes it a less stable choice.

Multimedia on the Web can refer to a number of things, including animated GIFs, Flash animations, audio, video, and Java applets. *Scripts* in this chapter refers to JavaScript and DHTML effects that you can add to your documents, creating a richer user experience.

NOTE
Although images, media, and scripting can bring more options to your site, they also can add unnecessary clutter and download time. I like to think of most content of this nature to be decorative. Just as you wouldn't want to overdecorate a house, think about how less can be more when it comes to your page.

Holzschlag_03.qxd 3/30/05 9:13 AM Page 32

The *img* Element

When working with images, the element you'll be using is `img`. This is an empty element—in other words, it does not contain any text content. As a result, it doesn't require a closing tag. It's written in XHTML as follows:

```
<img />
```

Placed all by itself within the body of your document, this will do nothing at all. So along with the `img` element itself, you'll need to point to the location of the image. This is done using the `src` attribute, which stands for "source."

In the value of the source attribute, you'll add the location and the name of the actual web graphic file, along with its extension. Example 3-1 shows a complete document with the image source included.

EXAMPLE 3-1 Adding the image into the document body

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

<head>

<title>Chapter 3</title>

</head>

<body>

<img src="images/photo.jpg" />

</body>
</html>
```

NOTE
You'll notice that I've used a subdirectory called images in which to store my web graphics. It's conventional to place images in a subdirectory beneath the documents that use them or, if you have a fairly small site, in one image location off the root directory.

Holzschlag_03.qxd 3/30/05 9:13 AM Page 33

This causes the image to appear in your browser window (see Figure 3-1).

FIGURE 3-1 The image appears within the browser window.

Without any content in the document, the image automatically is placed in the upper-left corner. You'll ultimately be doing a lot more with this image to make it more useful:

- Assist browsers with better rendering
- Provide helpful information to those who might not be able to view the image
- Link the image

NOTE
You'll explore how to do these things using XHTML first, but in Chapter 8, "Working with Color and Images Using CSS," you'll learn more advanced methods to control an image's presentation using CSS.

Holzschlag_03.qxd 3/30/05 9:13 AM Page 34

Adding *width* and *height* Values

The next thing you'll want to do for your image is add `width` and `height` values. This actually assists browsers in rendering images more efficiently, so it's always a good idea to add this information.

You can find the image's width and height in a couple ways. The first way is that you can look for it in your imaging editor (see Figure 3-2).

Another way to find width and height is to open the image itself in your browser.

Figure 3-3 depicts the image itself (not the HTML file), and you can see in the title bar that the width and height are displayed.

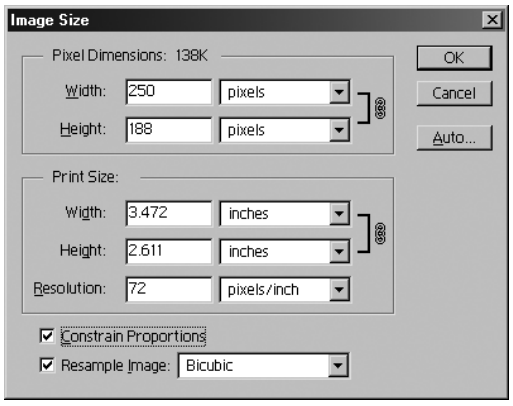

FIGURE 3-2 Look for the width and height in pixels for your image—here, Photoshop displays the width and height at the top of the Image Size dialog box.

FIGURE 3-3 I opened the image in my browser, and the image width and height appear within the browser's title bar.

NOTE
Not all browsers have this feature, but most common ones do.

When you've got the image dimensions—in this case, 250 pixels wide by 188 pixels high—you can place it into your image markup:

```
<img src="photo.jpg" width="250" height="188" />
```

NOTE
You should always include the correct width and height. If the `width` and `height` values are larger than the actual image, the browser will stretch the image to make it fit. If you note smaller values, the browser will squeeze the image into the smaller size, scrunching it up.

Holzschlag_03.qxd 3/30/05 9:13 AM Page 35

Providing Alternative Text

Some people surf the Web without images turned on. Sounds strange in today's world of high bandwidth, but some people still don't have high-speed access, so images can slow down their access to your page's content, and they will disable the images. Another important concern is that many people visiting the Web are visually impaired or blind. In all these cases, it's helpful to provide some clues to your visitor as to what the image represents.

This is done using *alternative text*, which uses the `alt` attribute and a description, as shown in Example 3-2.

EXAMPLE 3-2 Adding an `alt` text description

```
<img src="photo.jpg" width="250" height="188" alt="photograph of a delicious
Vietnamese noodle dish from restaurant Pho 88" />
```

Alternative text descriptions appear in two ways on the site. First, they appear in the location of the image before the image load and when images are disabled (see Figure 3-4).

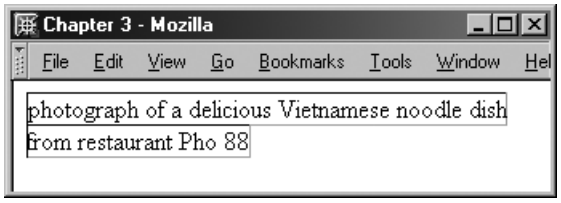

FIGURE 3-4 Alternative text in a browser where images are disabled.

The second way alternative text appears is upon mouseover of an image. This assists everyone because it provides more contextual clues on the image's purpose (see Figure 3-5).

FIGURE 3-5 Alternative text in a ToolTip as the mouse passes over the image.

Holzschlag_03.qxd 3/30/05 9:13 AM Page 36

Linking the Image

Many times you will want to link an image to either another HTML document or a detailed version of the image. In either case, linking an image works the same way as linking text. You surround the image code with the anchor element and the reference to where the image is linking, just as if it were the text content (see Example 3-3).

EXAMPLE 3-3 Linking the image

```
<a href="detail.html">
<img src="images/photo.jpg" width="250" height="188" alt="photograph of a
delicious Vietnamese noodle dish from restaurant Pho 88" />
</a>
```

The image is now linked, and when clicked on, it will take the visitor to the `detail.html` page. You can even add a `title` attribute to the link if you want further details about the link to be available to your visitors. By default, browsers place a border around the image to highlight the fact that it is a linked image, and the hand cursor appears upon mouseover, too (see Figure 3-6).

FIGURE 3-6 A linked image.

Holzschlag_03.qxd 3/30/05 9:13 AM Page 37

If the image link is followed, the browser will use the default visited link color around the image. Of course, many people find the link border unsightly. If you'd like to get rid of your border immediately, you can do so by turning it off directly in the HTML, as shown in Example 3-4.

EXAMPLE 3-4 Using the `border` attribute

```
<a href="detail.html">
<img src="images/photo.jpg" width="250" height="188" alt="photograph of a
delicious Vietnamese noodle dish from restaurant Pho 88" border="0" />
</a>
```

The image, while still linked, now displays no border (see Figure 3-7).

FIGURE 3-7 The image, while still linked, has no visible border.

Beware: Borders Are Presentational

The `border` attribute is considered presentational because it can be used decoratively. By providing a value greater than `0`, the border size changes, whether the image is linked or not. Ideally, you will use CSS instead of the `border` attribute to modify your borders. CSS will also be used to position or float the image within its content. You'll learn more about this in Chapter 11, "Margins, Borders, and Padding," and Chapter 12, "Positioning, Floats, and Z-index."

Holzschlag_03.qxd 3/30/05 9:13 AM Page 38

Linking to an Audio or Video File

If you'd like to provide links to media on your site, you can do so just as easily as linking an image. Many file types exist for audio and video, the most popular these days being the MP3, QuickTime, Real, and Windows Media files.

You first place your media file into a subdirectory. As with images, this is a convention that helps you keep all your various files organized. In this case, I've named the subdirectory media (how's that for brilliant?) and placed two files in it, one an MP3 audio file and the other an `.avi` video file. Example 3-5 shows my document and how I've linked to my media files.

EXAMPLE 3-5 Linking to audio and video

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<title>Chapter 3</title>
</head>

<body>

<a href="media/audio-sample.mp3">Link to Audio Sample</a><br />
<a href="media/video-sample.avi">Link to Video Sample</a>

</body>
</html>
```

I've added text within the links and placed a break between the two links so they appear on top of one another rather than side by side, for the sake of clarity. This results in the links as shown in Figure 3-8.

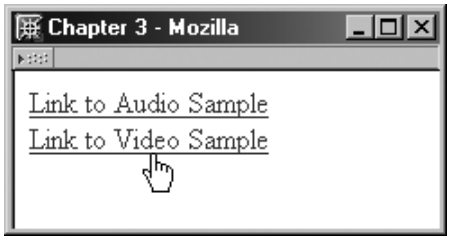

FIGURE 3-8 Links to audio and video samples.

Holzschlag_03.qxd 3/30/05 9:13 AM Page 39

So far, pretty easy, right? Well, there are a few more things to do with links to give visitors an easier time managing the audio and video.

Beware Browser Behavior Differs

Different browsers and different browser configurations influence the way that linked audio and video is displayed. As the link is selected, some browsers automatically download the file into an external media player and play the audio or video clip. Other browsers provide a pop-up option asking whether you'd like to open the file with the appropriate application, or download it and save it to your hard drive. Because of these differences in behavior, it's helpful to let your visitors know as much about the link they're about to click on as possible.

Because most audio and video clips are quite large, it's helpful to provide the file sizes on the page so visitors are prepared. You can do this by simply typing the details into the link or directly after it.

NOTE
Some folks even go so far as to provide a range of file sizes for their low-, medium-, and high-bandwidth visitors.

Another way to assist is to place a description of the file into the `title` attribute of the link (see Figure 3-9).

This helps provide more detail to all and also alerts those folks who can't see or hear to understand what the link is for (see Example 3-6).

EXAMPLE 3-6 Adding details for your visitors

```
<a href="media/audio-sample.mp3" title="audio of molly singing">Link to Audio
Sample</a>. Size: 1,300KB<br />
<a href="media/video-sample.avi" title="video of sarah's dance class">Link to
Video Sample</a>. Size: 29,000KB
```

FIGURE 3-9 Providing helpful details for audio and video links.

Holzschlag_03.qxd 3/30/05 9:13 AM Page 40

Embedding Files Using the *object* Element

Another means of providing audio, video, and other multimedia such as Flash animations and Java applets is to embed them directly into the page. This means that the software plug-in automatically loads with the page.

All external files are considered *objects*. This includes images as well as multimedia files. In contemporary HTML and XHTML specifications, the proper way to include all multimedia is to use the `object` element to embed a file directly:

```
<object data="media/video-sample.avi" type="video/avi" />
```

This results in the player application appearing on the page. The video can then be played (see Figure 3-10).

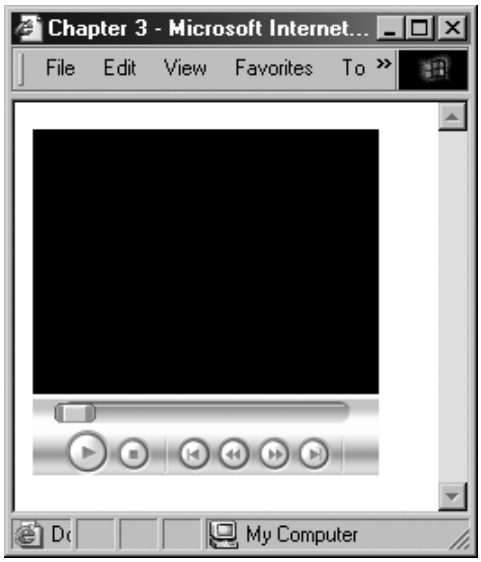

FIGURE 3-10 The embedded player loaded into Internet Explorer.

NOTE
Of course, you can do a lot more in terms of providing advanced settings. To learn more about the many available settings for multimedia, see the excellent tutorial at http://www.w3schools.com/media/default.asp.

Holzschlag_03.qxd 3/30/05 9:13 AM Page 41

In instances with Flash files, you use the `object` element to achieve inline results, as you can see in Example 3-7.

EXAMPLE 3-7 Embedding a Flash movie file (SWF) into a page using `<object>`

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" width="100"
height="100" codebase="http://active.macromedia.com/flash6/cabs/
swflash.cab#version=6,0,0,0">
<param name="movie" value="media/ava.swf" />
<param name="play" value="true" />
<param name="loop" value="true" />
<param name="quality" value="high" />
</object>
```

In most standards-compliant browsers that also have Flash enabled, the file should play directly upon loading, as shown in Figure 3-11.

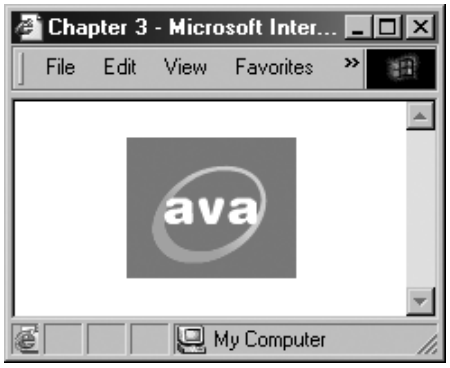

FIGURE 3-11 The Flash animation of the AVA logo plays inline in Internet Explorer.

NOTE

As you can see, quite a bit of information has to go along with objects, including `codebase` information and use of the `parameter` element to define not only the location of the file, but also aspects about how it's to be played. These options are all generated by Macromedia Flash when you create the Flash file. For more information on Flash, see http://www.macromedia.com/software/flash/.

You can use the `object` element for audio and Java applets, too. Simply add the correct `codebase` information and desired parameters, and you'll be good to go.

Holzschlag_03.qxd 3/30/05 9:13 AM Page 42

But Your Honor, I Object!

Okay, the entire last section was filled with lies. Not that I'm trying to steer you wrong—I'm trying to do quite the opposite. All external media should be addressed using the `object` element if you want to be using markup that exists in the valid world of HTML and XHTML.

However, there's a big stumbling block with using the `object` element, and that is that support for it is inconsistent across browsers and platforms. This is very disturbing from a purist's point of view because there's no other alternative *within the specifications.*

QUANTUM LEAP: OBJECT HANDLING IN XHTML 2.0

In XHTML 2.0, the `object` element becomes ubiquitous. In other words, any other elements for external objects, including the `img` element, are made obsolete. Obviously, it's preliminary to use XHTML 2.0 because your results will be limited to those very few browsers that support `object` for `img`. But it gives you a good idea of the direction XHTML is taking.

Here's what it boils down to, in simple terms: If you want your multimedia to be as consistent as possible across browsers, you have to turn to a proprietary element, the `embed` element. This element has never existed in any of the formal specs, but most all browsers support it; although your pages with the `embed` element will cause validation errors, they're going to work.

Conventional wisdom mixes both `object` and `embed`. So if you were to use this approach with the same Flash file just described, you'd end up with the markup shown in Example 3-8.

EXAMPLE 3-8 Embedding a Flash movie file (SWF) into a page using `<object>` and `<embed>`

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" width="100"
height="100" codebase="http://active.macromedia.com/flash6/cabs/
swflash.cab#version=6,0,0,0">
<param name="movie" value="media/ava.swf" />
<param name="play" value="true" />
<param name="loop" value="true" />
<param name="quality" value="high" />
<embed src="media/ava.swf" width="100" height="100" play="true"
loop="true" quality="high"></embed>
</object>
```

The Flash movie will now play inline in almost every browser.

Holzschlag_03.qxd 3/30/05 9:13 AM Page 43

Adding Scripts

Another means of bringing interactivity and interest to your pages is adding scripts to them. Typically, this refers to JavaScript or what is known as *Dynamic HTML* or DHTML, which is a combination of technologies, including HTML, CSS, JavaScript, and the Document Object Model. Combining these technologies gives you rich features such as drop-down menus and interactive games.

QUANTUM LEAP: THE DOCUMENT OBJECT MODEL

The Document Object Model, also referred to as the DOM, is the interface within browsers that enables you to attach scripting to specific elements. Part of the reason DHTML has been controversial and problematic is that browsers have implemented nonstandard DOMs, which have resulted in poor consistency. When you are looking for DHTML scripts, be sure that you're using those scripts that offer the most cross-browser support. The DOM is standardized, and all contemporary, standards-based browsers are working to implement DOM standards efficiently.

You can add scripts to your document in two primary ways. One is to place the script into the head portion of your document. This is referred to as an *embedded* script. The other way is to place your script external to the document, which is referred to as a *linked* script.

Embedding a Script

To embed a JavaScript in the head portion of your document, you use the `script` element to contain the script (see Example 3-9).

EXAMPLE 3-9 Embedding a script into the head of a document

```
<head>
          <script type="text/javascript">
          function newWindow() {foodWindow = window.open("images/photo.jpg",
          "foodWin", "width=250,height=188")}
          </script>
</head>
```

The purpose of this script is to set up the document to open the image `photo.jpg` in a new window when a specific link is clicked. You also need a bit of script in the actual link found in the body of the document, as follows:

```
<a href="javascript:newWindow()">Delicious Vietnamese Lunch</a>
```

Holzschlag_03.qxd 3/30/05 9:13 AM Page 44

Figure 3-12 demonstrates how clicking on the link makes the pop-up window appear with the image intact.

FIGURE 3-12 The results of the embedded script.

Linking to a Script

In terms of best practices, the more you can get out of your document and into external files in terms of scripting and style, the better. You can have many pages pointing to one script, and if you require changes to the script, you can make them to the one script file instead of to many documents with embedded scripts.

To link to the script, first place the script code (without any HTML) into a separate file, and name the file with a `.js` extension, as in `popup.js`. You can place that file into a subdirectory named scripts (just as you did with images and media), and then use the `script` element to link it to the document (see Example 3-10).

EXAMPLE 3-10 Linking to the script

```
<head>
<script src="scripts/popup.js" type="text/javascript"></script>
</head>
```

Leave the link code as is within the body of the document, and the results will be exactly the same as demonstrated in Figure 3-12.

Holzschlag_03.qxd 3/30/05 9:13 AM Page 45

Scripting and Browser Concerns

In some instances, people are using old browsers with no JavaScript support or poor support for the `script` element itself, or have JavaScript disabled. Working around these issues requires some additional markup.

NOTE
Most contemporary web designers do not use the workarounds here unless they absolutely know that they have to support older browsers. However, you might want to use them. At the very least, it's important that you see these techniques in action so you'll recognize them when viewing HTML from other sources.

Hiding Scripts from Older Browsers

If you're using embedded JavaScript, some older browsers attempt to display whatever is contained within the `script` element as body text.

To avoid this, many people got into the habit of "commenting out" their scripts—in other words, using comment syntax to prevent the script from being displayed (see Example 3-11).

EXAMPLE 3-11 Hiding a script with comments

```
<head>
          <script type="text/javascript">
          <!-- this hides the script from older browsers
           function newWindow() {foodWindow = window.open("images/photo.jpg",
           "foodWin", "width=250,height=188")}
           // End hiding script from old browsers -->
           </script>
</head>
```

Note the //. This is JavaScript syntax that enables you to write in a comment after that point that won't be displayed, either. The commenting used here will *not* prevent the script from operating normally in any browser that supports it.

Using the *noscript* Element

If you'd like to add some text so supporting browsers will display a message regarding script support, you can do so using the `noscript` element (see Example 3-12).

Holzschlag_03.qxd 3/30/05 9:13 AM Page 46

EXAMPLE 3-12 Using the `noscript` element

```
<head>
          <script type="text/javascript">
          <!-- this hides the script from older browsers
          function newWindow() {foodWindow = window.open("images/photo.jpg",
          "foodWin", "width=250,height=188")}
          // End hiding script from old browsers -->
          </script>

          <noscript>Attention: Your browser does not support JavaScript or you
          have disabled JavaScript.
          </noscript>
</head>
```

Browsers that support scripting and do not have scripting disabled will *not* see the contents of the `noscript` element.

However, those browsers without JavaScript or, as in the case with Figure 3-13, browsers with JavaScript purposely turned off will display the text within the `noscript` element.

FIGURE 3-13 The `noscript` text within a browser with disabled JavaScript.

As you can see, the link is still intact, but the pop-up script will not work if JavaScript is disabled or unavailable.

Finding Scripts Online

One of the great things about JavaScript and DHTML is that so many free scripts are available. Of course, the downside of having so many free scripts available means that many of those scripts might be substandard or that newer, better scripts have come along since. Because of that, use discretion and read the fine print. A few sites that I like include http://javascriptkit.com/, http://simplythebest.net/scripts/DHTML_scripts/, http://www.javascripts.com/, and http://www.dynamicdrive.com/.

Holzschlag_03.qxd 3/30/05 9:13 AM Page 47

Imagine That!

From structure to well-formatted text, to great imagery and interactive features, you've sure come a long way in three short chapters.

Of course, if you're getting frustrated because all of this seems very much like building a house while you're imagining how it's going to be decorated, that's understandable. It's important to keep in mind that building great web pages in today's world means taking the extra time to organize your materials, have clear goals, and take pride in the crafting of your documents.

As with a home, the better the foundation, the more well-built and finely crafted the structure, the easier it will be to make aesthetic modifications. This is really what we're after by taking the time to build our pages correctly. Just something as simple as placing all your images in an image directory, scripts in a script directory, and additional media in corresponding directories means having an internal site structure that will grow with you instead of causing collapse as your site grows and changes to meet your needs.

Imagine if you hadn't taken the time early on to build the structure well. Take the advice of professional web developers who have learned the hard way: Not building the infrastructure well can lead to all kinds of expensive, time-consuming, and downright frustrating problems along the way.

Now that you're a bit more organized in terms of your document, text, and image and media management, it's time to get fancier. In the next chapter, you'll be learning how to build effective tables. Once the holy grail of how websites were laid out visually, tables are being revisited for their structural integrity.

What we're learning is that CSS is a lot more efficient and flexible for the presentational aspects of our site, but tables can be extremely useful for displaying a range of information in effective ways. Depending upon your needs, you might find tables an excellent way of managing data, further assisting your site visitors to easily get to and understand the information you're sharing with them.

Holzschlag_03.qxd 3/30/05 9:13 AM Page 48