
5

Images

``
Strict / Transitional / Frameset

Though I remember the days of the Web without graphics—I saw my first copy of Mosaic in 1994 at John Marshall Law School in Chicago—Web pages without images today are few and far between. The `` element is the one that directs the browser to “place an image here.” Before I explain this element and all its associated attributes, a brief introduction to graphic file formats for the Web is in order.

WHAT KIND OF GRAPHICS CAN YOU USE?

Today’s browsers support three different graphic file formats: .gif, .jpg, and .png. Which you use depends on several factors. Let’s take a look at the benefits of each.

.GIF¹

The Graphics Interchange Format was originally developed for CompuServe back in the 1980s. A longtime standard, it was quickly adopted by the Web design community as the first file format to be used in Web pages. .GIFs had several benefits over other existing formats of the time. Primarily, they were relatively small, and size was a factor when transmitting images over slow connections. .GIF files also had some additional features that Web developers could take advantage of.

- GIFs could be saved in an interlaced format. This is not nearly as important today with the advent of broadband connections in the home, but you’ve probably seen

an interlaced GIF. Like all other image file formats, GIFs load topdown, line by line. An interlaced GIF, however, loads in four passes, looking fuzzy at first and becoming clearer with each pass. This tricks your mind into thinking the image is loading faster even though it's not. It's just an optical illusion, but when people were connecting to the Internet with 14.4 or 28.8 kbps connections, this illusion was very effective.

- .GIFs could be used as imagemaps. An imagemap is a single image with multiple hyperlinked “hotspots.” Depending on which hotspot on the graphic you clicked, you received a different result. For example, you've probably visited a site in the past with a map of the United States. You clicked on the appropriate state to view that state's page. (Click on Colorado, you get a Colorado page. Click on Kansas, you get a Kansas page.) We'll be covering the creation of imagemaps later in this chapter.
- .GIFs can be transparent. When I tell my students that all graphics on all Web pages are rectangles (i.e., have four sides), many don't believe me. After all, we've all seen graphics that look like circles. Well, that's the catch: They just look like circles. A transparent .GIF has one of color (usually the background color) that is “transparent.” In other words, it automatically matches itself to the background of the page it's on, thus giving the appearance of having more or fewer than four sides.
- .GIFs can be animated. With additional software, you can create a series of .GIFs that, when saved as a single file and played back by the browser appears as a cartoon or a digital flip-book, giving the image the illusion of movement.

.JPG (OR .JPEG)

The Joint Pictures Engineering Group in the mid-1990s devised an alternative to the .GIF format. Though the .JPG format cannot support some of the features that .GIF offers, its smaller file size is a measurable advantage over .GIFs.

The .JPG file format has a built-in method of compressing an image file, resulting in files as much as 50 percent smaller than the same file saved in the .GIF format. With a smaller file, the image will load more quickly. There is, however, a potentially significant drawback to this bandwidth savings.

The .JPG format uses a form of “lossy” compression. In other words, the smaller file size is a result of data being removed (“lost”) from the image. In most cases, this lost data is not noticeable to the average user due to limitations both in their monitors and in their own eyes.

Most authors tend to rely on .GIFs for imagemaps since some older browsers did not support JPG-based imagemaps. Today's browser's do support .JPG-based imagemaps.

.PNG

The Portable Network Graphics format was created in the late 1990s as a replacement for both .JPGs and .GIFs. This format advertises itself as a “best-of-both-worlds” solution. .PNGs support all the features contained within .GIFs but have the smaller file sizes of .JPGs with lossless compression (compression without losing any data from the image).

Though all of today's browsers support the display of .PNGs, not all of those browsers support them well. For example, Internet Explorer does display .PNGs but does not support transparency in .PNGs. For this reason, many Web authors have still not started using .PNGs in their pages.

CHOOSING A FILE FORMAT FOR YOUR GRAPHICS

Since .PNGs still have some issues that prevent Web authors from considering them as a serious option, I'll focus this discussion on choosing between the .GIF and .JPG formats.

Potentially, your choice is simple. If you want to use a transparent or interlaced image, .GIF is your format. However, if you don't want either of these features as part of your image, the choice does not automatically become .JPG. The main reason for this is the minor problem of lossy compression. Certain types of images work better and some suffer noticeably from lossy compression. Take a look at the following two graphics saved in both the .GIF (Figures 5.1 and 5.2) and .JPG (Figures 5.3 and 5.4) formats.

Figure 5.1. .gif photo, 640x480, 159k



Figure 5.2. .jpg photo, 640x480, 38.1k



Figure 5.3. .gif line art, 200x315, 18.6k

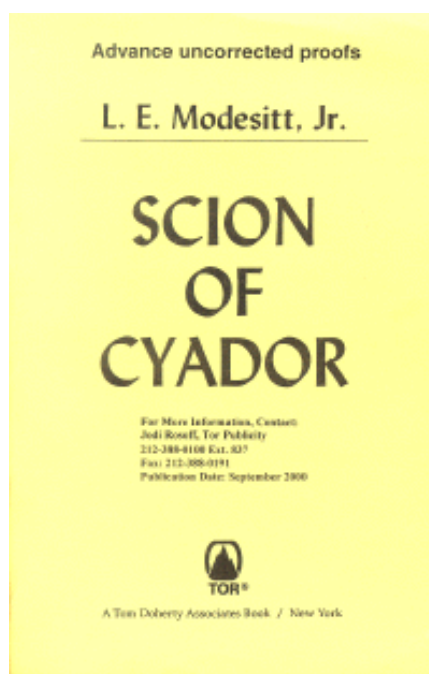
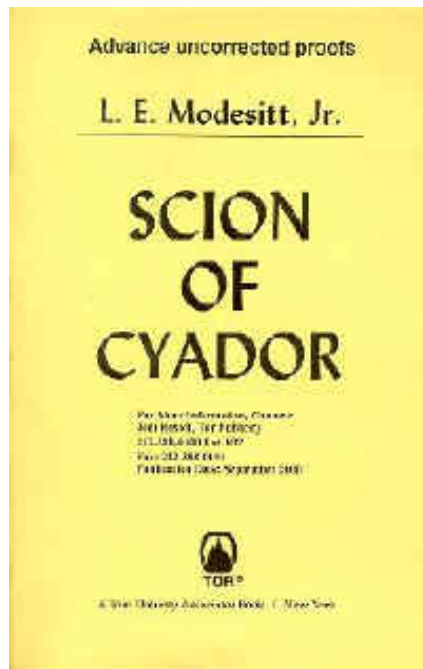


Figure 5.4. .jpg line art, 200x315, 7.4k

If you examine the photo in both the .GIF and .JPG formats, you shouldn't see much of a difference, if any at all. Complex images such as photographs lend themselves well to .JPG compression because of their level of detail. The more detail you have, the more areas where colors are blended or less distinct, and the more places where the compression process can remove data unnoticed.

On the other hand, looking at the drawing, the .GIF version is crisp where the .JPG version is fuzzy around the edges. In images such as drawings, cartoons, or line art, where there is less detail and color separation is distinct, there are fewer places where data loss can go unnoticed. The result is fuzziness around the edges. You do end up with a faster-loading file, but the image suffers in its quality—a compromise you may not be willing to make.

The decision comes down to the type of image you're placing within your document. If it's a simple image, stick with .GIF. The .JPG format is better suited for photographs and other complex images.

IMAGE ELEMENT BASICS

Images are inline elements. They will be placed exactly where they are specified inline with any surrounding text. To place an image into a document you first need to add the `` element. (Please note that `` is an empty element and therefore requires the trailing slash.)

For example, let's place a library logo at the top of our Web page. Those with previous HTML experience would envision something like the following code.

```
<body>
<img />
<h1>Welcome to the Acme Public Library</h1>
```

As acceptable as this code would be to longtime Web designers, it has a problem that will prevent this code from validating. An image must be placed within something. At the moment, it is a child of `<body>`, which is an invalid code. For our purposes, we'll place our image within its own `<div>`. This will have no effect on our layout and will create valid XHTML code. (`<div>` is similar to `<p>` in that it's a container for other items. For now, just think of `<div>` as a different form of `<p>`. We'll discuss `<div>`s further in Chapter 13.)

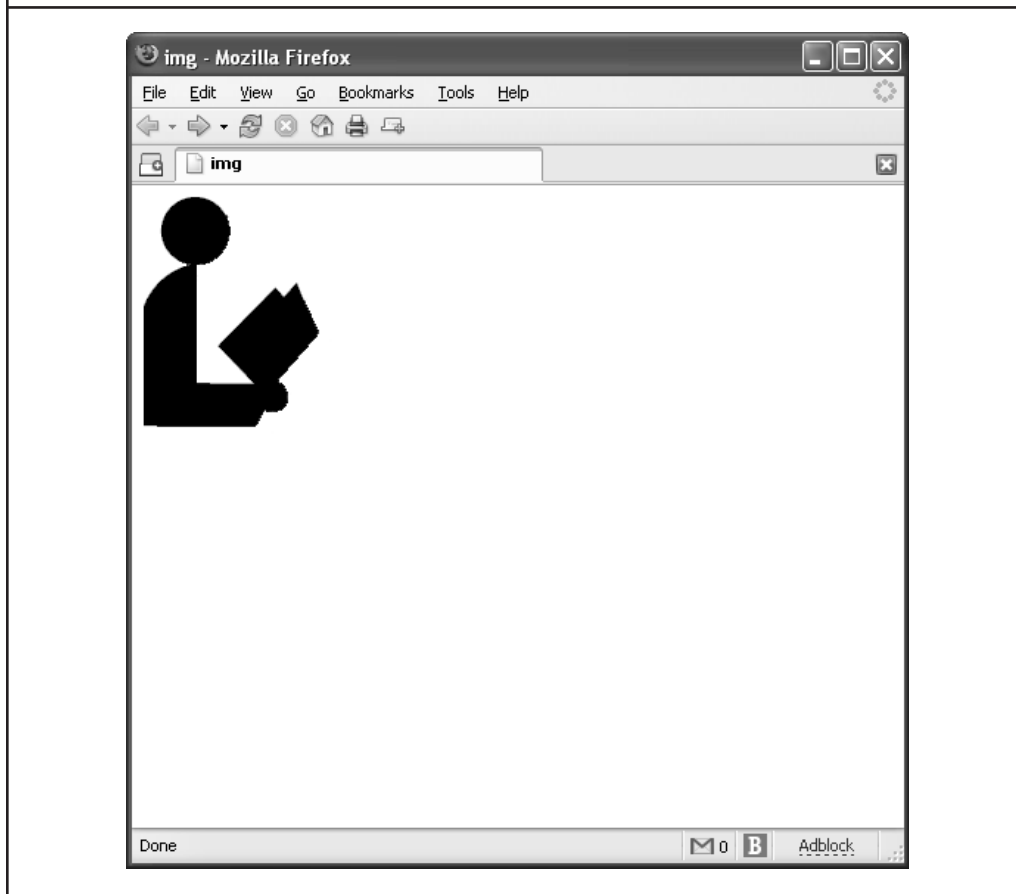
```
<body>
  <div>
    <img />
  <div>
<h1>Welcome to the Mallville Public Library</h1>
```

You're welcome to look at the results of this code in your browser but there's a reason I haven't included a screenshot of the results at this point. As written, our `` element accomplishes absolutely nothing beyond putting a place holder for the image in our code. In order for the image to appear we need to specify the name and location of the image we wish to have displayed.

NOTE: For the purposes of this discussion, we'll be assuming that our XHTML and image file are in the same directory on our site. Once your site starts to contain many images, you may want to put all your graphic files in a separate "images" directory and indicate that the images are elsewhere on the service by specifying the path to the appropriate image file. For more information on pathing, see the section on directory structures in Appendix I.

The attribute that we need to add to specify which image we'd like to display is the source attribute, abbreviated as `src=". . ."` (Figure 5.5).

```
<div>
  
<div>
```

Figure 5.5. Inserting an Image into Your Document

It might go without saying but I like to be specific: in XHTML, `src=" . . . "` is a required attribute. Without the appearance of this attribute on your `` element, the document will not validate.

One more attribute on the `` element is required for your document to validate—the `alt=" . . . "` attribute.

NOTE: The `alt` attribute is not required in HTML, though all good Web designers were sure to include it for other reasons which I'll be covering shortly.

The `alt` attribute specifies text that should be displayed when the image, for one reason or another, is not. For example, users with text-only browsers will see, or hear, the alternative text instead of the image (Figure 5.6). Today's graphical browsers will show the alternative text as tip-text (Figure 5.7).

Alternative text should serve one of two purposes: either to say what the picture is, as with a photograph, or explain what the image does, in the case of a "back" button.

```
<div>

</div>
```

Figure 5.6. Alt-text as Displayed in Lynx

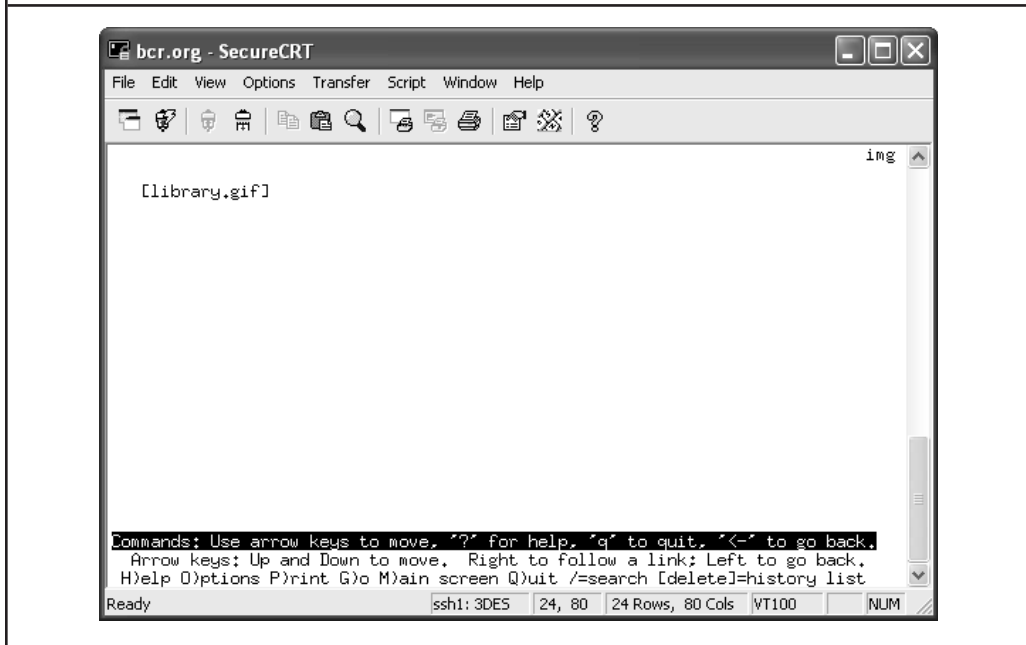
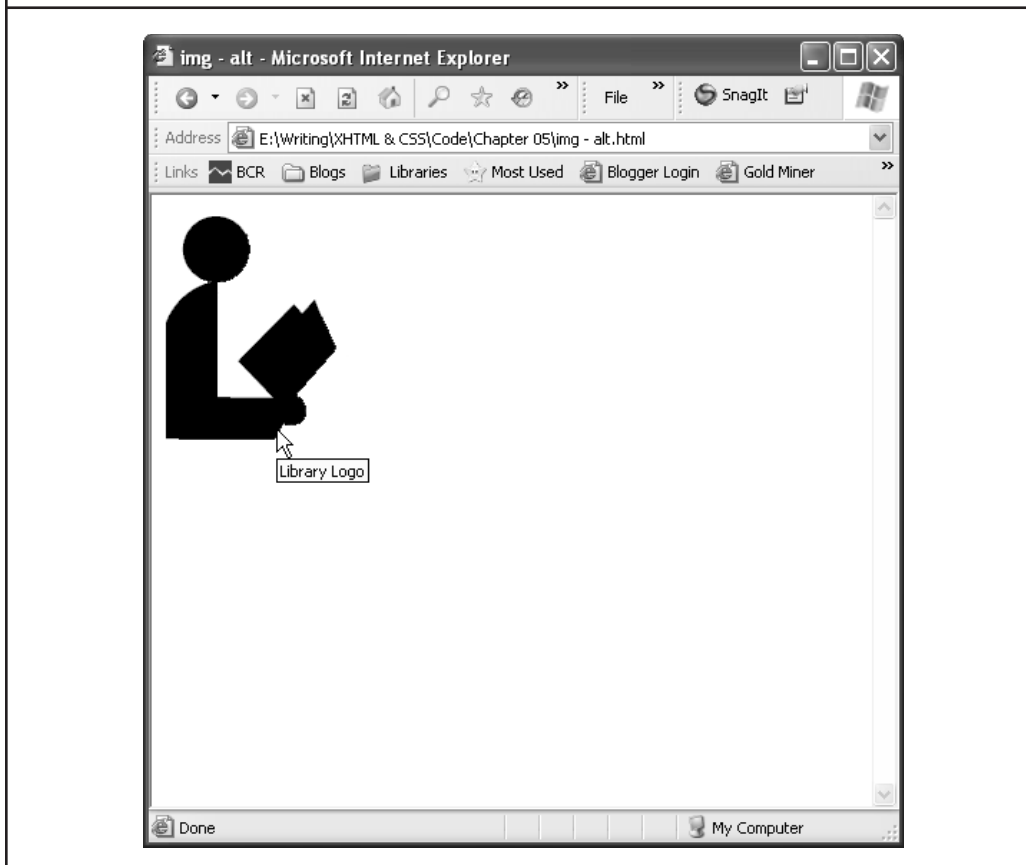


Figure 5.7. Alt-text as Displayed in IE



ADDITIONAL IMAGE ATTRIBUTES

All three of the following attributes are in the strict, transitional, and frameset DTDs. However, the first two do have preferred CSS replacements.

height **AND** width

The `height` and `width` attributes specify the dimensions of the image in pixels. Though these are not required attributes, they are useful pieces of information to add. The addition of this information does not increase the speed at which the image is loaded or displayed, but it does increase the speed at which the complete page is displayed properly.

When a page is loaded, the text content is loaded first and all of the content that must be retrieved from external files loads next. In this case, the text will load and display, then the images will be loaded. If you do not specify `height` and `width` for your images, the text will be displayed and then will move out of the way of the images once they're loaded. If you provide the dimensions of the image in your code, the browser will know how much space to set aside for the image and place the text around that space.

```
<div>

</div>
```

longdesc

The `longdesc` attribute allows you to specify the URL of a document that provides a narrative description of the image far beyond that which can be displayed within a short bit of alternative text. None of today's browsers that I am aware of actually do anything with this information.

```
<div>

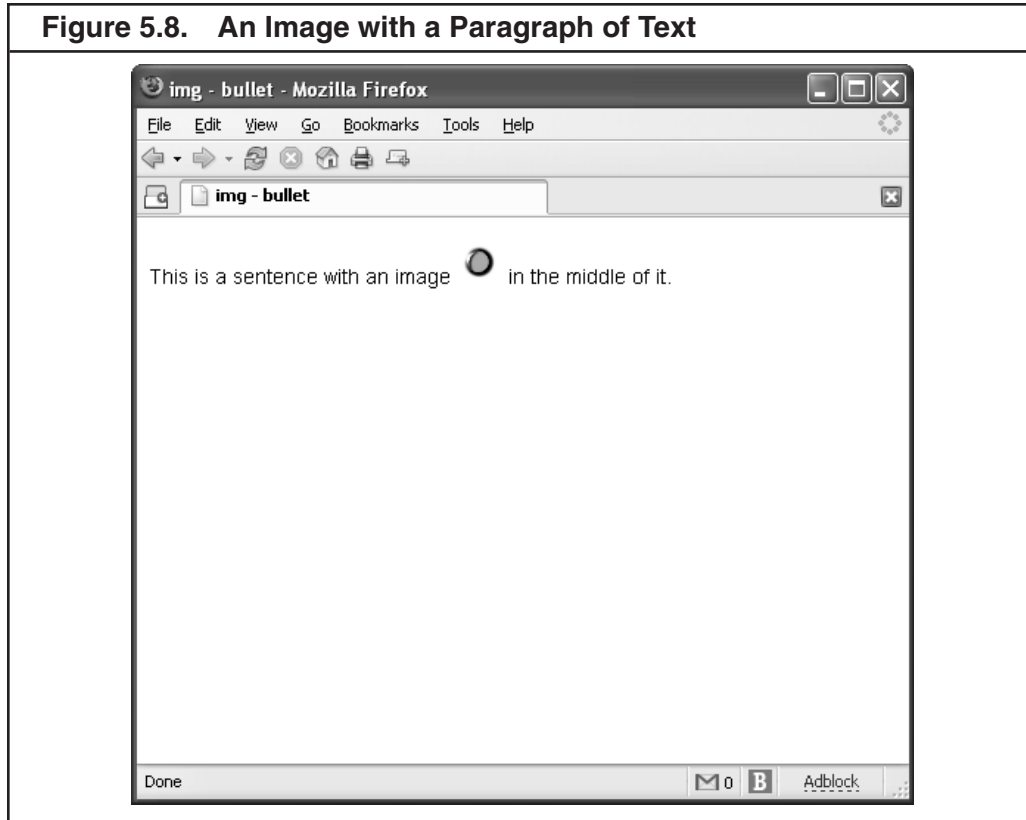
</div>
```

The next four attributes do not appear in the strict DTD. For each of these, use of the CSS alternative is strongly suggested.

alignment

By default, `` is an inline element. As proof, take a look at the following example (Figure 5.8).

```
<p>This is a sentence with an image  in the middle of it.</p>
```



As you can see, the image is placed exactly where specified and the text flows around it as needed. Through the application of the `align` attribute, we have a minimal amount of control over the placement of our image.

The following table covers the values for the `align` attribute followed by example code and a screenshot of each as displayed in today's browsers.

Table 5.1
align Attributable Values

left	Sends the image off to the left of the text, forcing the text to wrap around the right side of the image.
right	Sends the image off to the right of the text, forcing the text to wrap around the left side of the image.
top	Aligns the top edge of the image with the adjacent text.
middle	Vertically centers the image with the adjacent text.
bottom	Aligns the bottom edge of the image with the adjacent text. This is the default value.

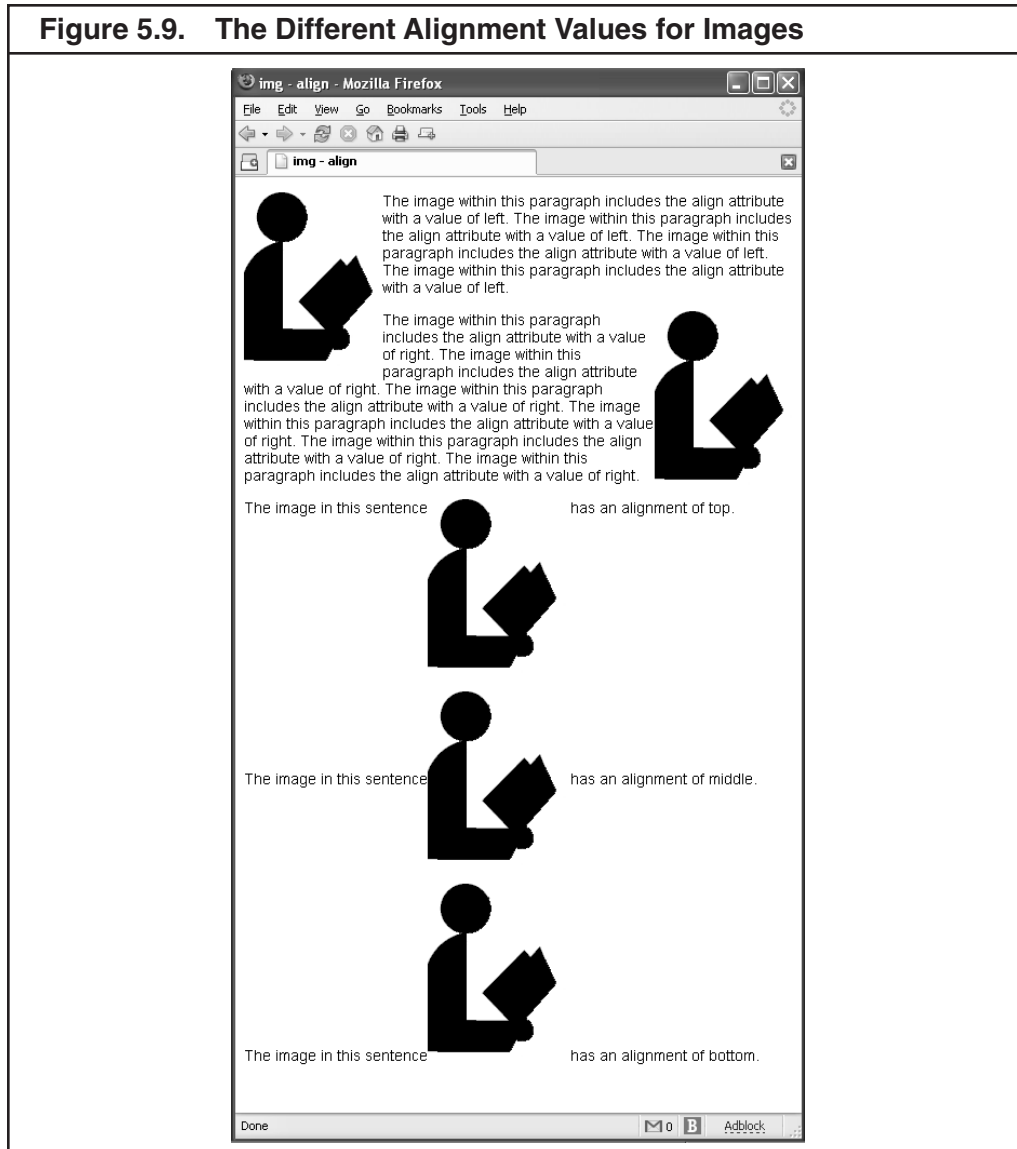
```
<p>The image
within this paragraph includes the align attribute with a value of
left. The image within this paragraph includes the align attribute
with a value of left. The image within this paragraph includes the
align attribute with a value of left. The image within this para-
graph includes the align attribute with a value of left. </p>
```

```

<p>The image
within this paragraph includes the align attribute with a value
of right. The image within this paragraph includes the align
attribute with a value of right. The image within this paragraph
includes the align attribute with a value of right. The image
within this paragraph includes the align attribute with a value
of right. The image within this paragraph includes the align
attribute with a value of right. The image within this paragraph
includes the align attribute with a value of right.</p>
<p>The image in this sentence has an alignment of top.</p>
<p>The image in this sentence has an alignment of middle.</p>
<p>The image in this sentence has an alignment of bottom.</p>

```

Figure 5.9. The Different Alignment Values for Images



BORDERS

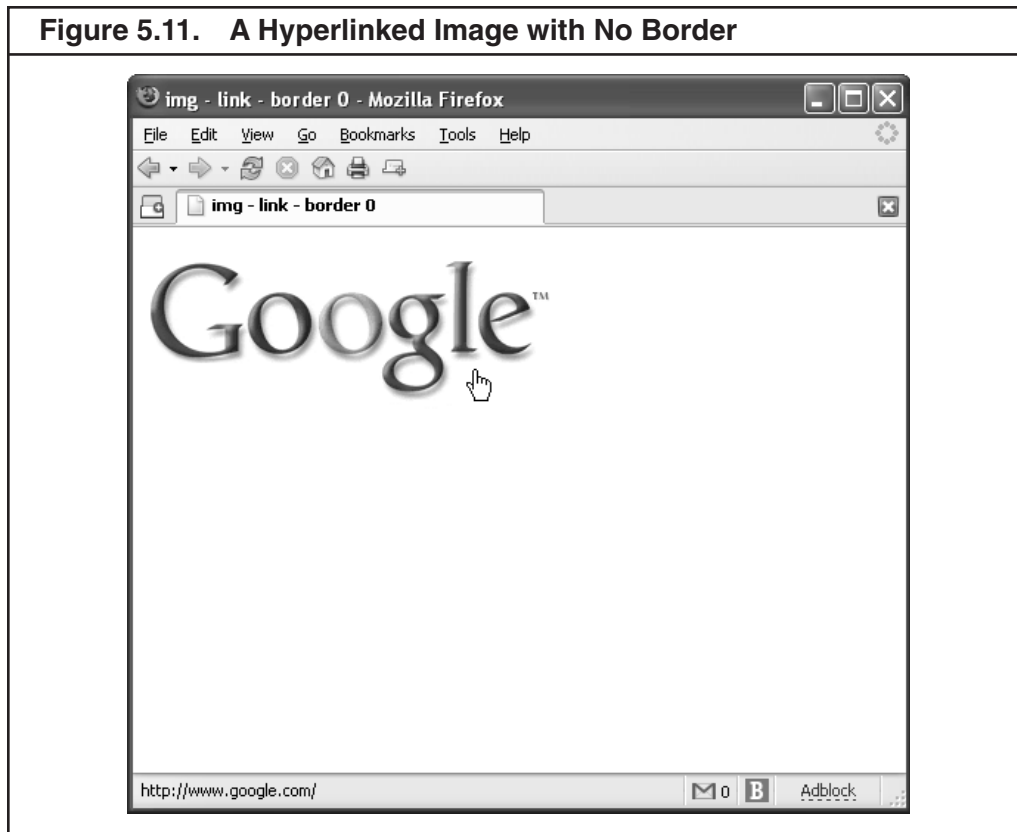
When you add an anchor to text, the text changes color and underlines by default. Let's look at what happens when you add an anchor to an image (Figure 5.10).

```
<p><a href="http://www.google.com"></a></p>
```

As you can see, the image suddenly obtains a colored border. (The color will match whatever colors you've set for your links.) In this example, you can assume that I really don't want that border to appear, especially since this is a transparent image and adding the border ruins that effect.

The odd part about getting rid of this border is that it is done by adding an attribute to the image, despite the fact that the border is caused by the anchor. That attribute is `border="n,"` where `n` is the number of pixels thick you would like the border to be. (The default value is "2.") To make the border disappear, set the value of `n` to zero (Figure 5.11).

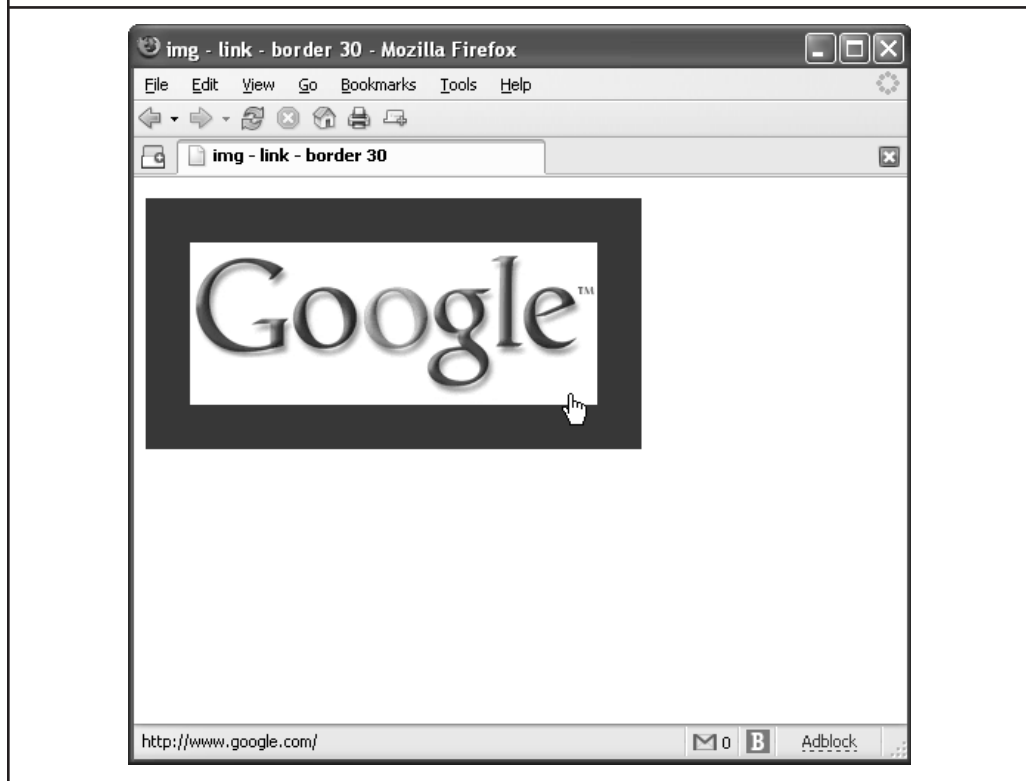
```
<p><a href="http://www.google.com"></a></p>
```

Figure 5.11. A Hyperlinked Image with No Border

Yes, you can set to any other numerical value. The following example illustrates why this is not recommended (Figure 5.12).

```
<p><a href="http://www.google.com"></a></p>
```

Figure 5.12. A Hyperlinked Image With a Border of 30

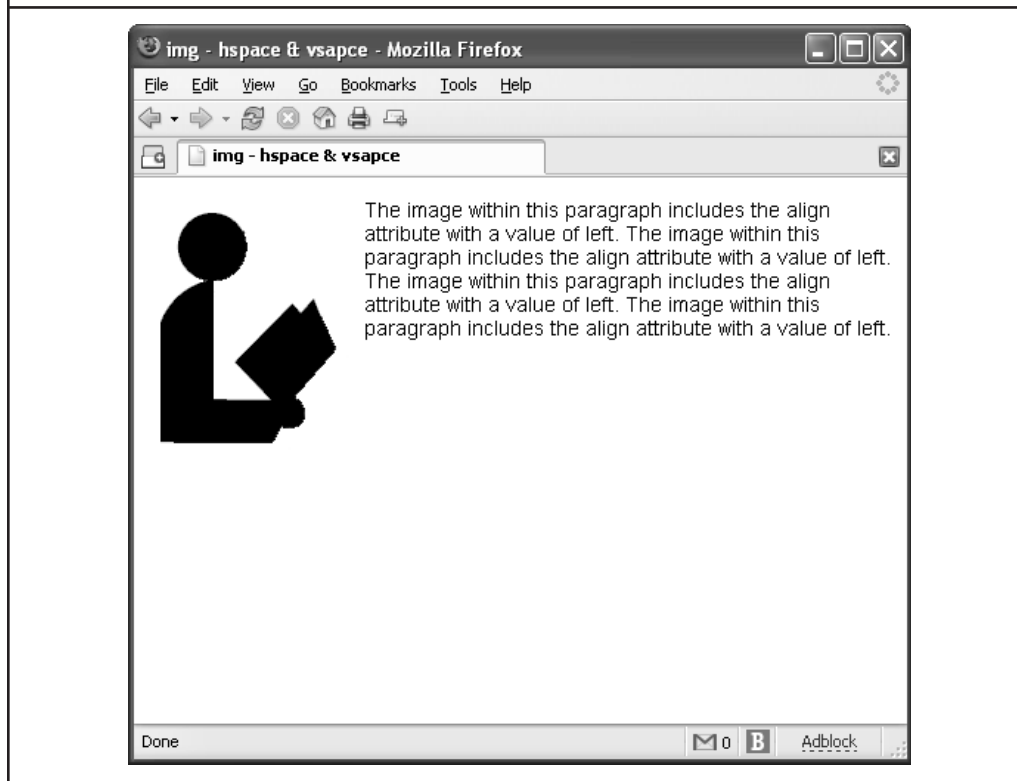


hspace AND vspace

As you may have noticed in previous examples, when an image is immediately next to text, the two practically bump up against each other. When you have a transparent .GIF, this may not be much of a problem since you may have a bit of a buffer zone built into the transparent portion of the image. However, if the image proper comes right up to the edge of the image's box (for example, as with a book cover), having your text and image immediately next to each other may not appear as intended. This is where the `hspace` and `vspace` attributes come into play. The `hspace` attribute allows you to specify a buffer zone on the left and right sides of the image into which no other content may encroach. The value for `hspace` is the number of pixels you wish to set aside. Similarly, the `vspace` attribute does the same thing as `hspace` but works above and below the image.

If I wanted to specify a ten-pixel buffer zone around the entire image, I would add the following code (Figure 5.13).

```
<p>The image within this paragraph includes the align
attribute with a value of left. The image within this paragraph
includes the align attribute with a value of left. The image
within this paragraph includes the align attribute with a value
of left. The image within this paragraph includes the align
attribute with a value of left. </p>
```

Figure 5.13. The hspace and vspace Attributes

As mentioned previously, these attributes have been replaced with CSS, which provides for far greater control than do `hspace` and `vspace`. Because of this, the CSS method for controlling the buffer zone around an image is the preferred method.

IMAGEMAPS

```
<img usemap="" />
<map> . . . </map>
<area />
```

Strict / Transitional / Frameset

As I mentioned at the beginning of this chapter, imagemaps are single images with multiple hotspots (links embedded within the image so that clicking on a different place in the image renders different results).

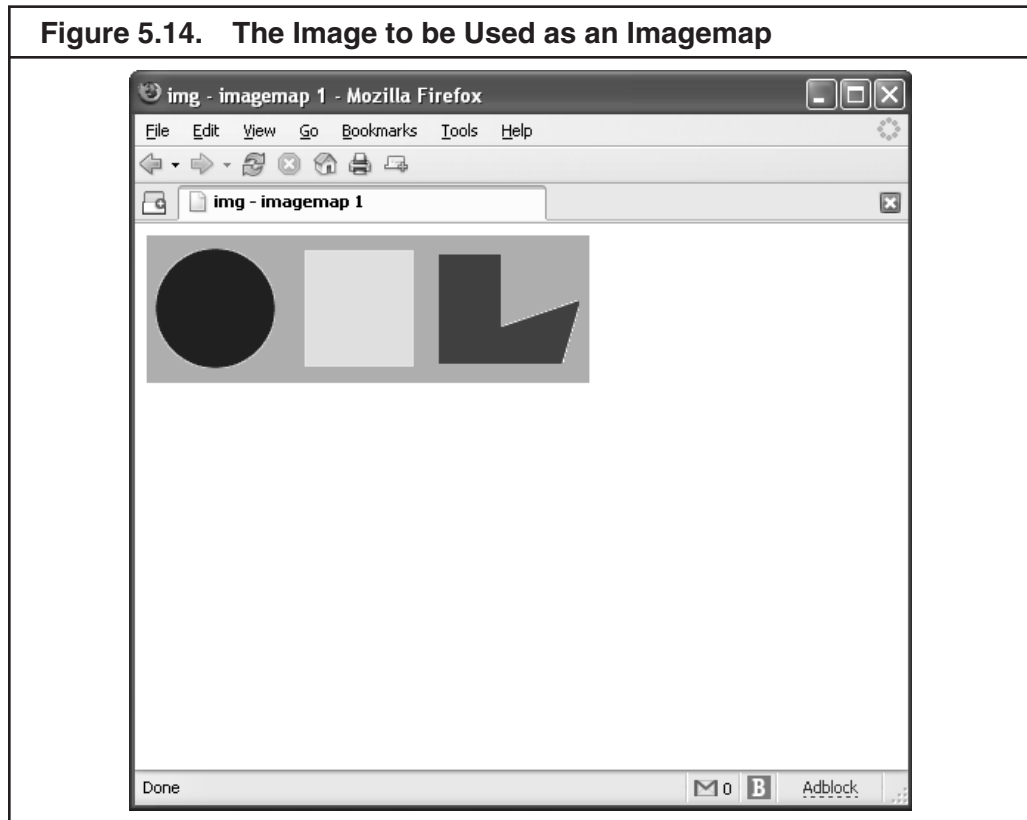
Traditionally, there have been two types of imagemaps an author could create server-side and client-side. Server-side imagemaps require storing the information on how the imagemap worked in a separate file on the server, forcing a connection back to the server whenever the user clicks on the associated image. Client-side imagemaps have nearly replaced server-side, mainly because they are more efficient, placing all of the imagemap code within the document itself. For this reason I will focus on client-side imagemaps. (Information on how server-side imagemaps work can be found at <http://hotwired.lycos.com/webmonkey/96/39/index2a.html>.)

To build an imagemap, you first need to place your image into your document. This is done as with any other image through the `` element. (Figure 5.14).

```
<body>
<div>

<div>
</body>
```

Figure 5.14. The Image to be Used as an Imagemap



I realize this is not exactly an image you would really use, but it does allow me to illustrate the points of imagemaps well.

Next, you need to identify the image as being associated with a particular set of map coordinates. This is done with the `usemap="value"` attribute. In this case we'll call it "shapes," since we're associating it with our `shapes.gif` file.

```
<body>
<div>

<div>
</body>
```

The next step is to place the map code within the XHTML document. Map code can be placed anywhere within the body of a document but is traditionally placed at the top, immediately after `<body>`. What we need to do is to indicate the beginning and end of the map code along with the name of the map.

```

<body>
  <map id="shapes">
  </map>
<div>
  
</div>
</body>

```

Some older browsers will not recognize the `id` attribute on the `map` and therefore will not associate the contents of `<map>` with the image. (This is similar to the name vs. ID issue with internal hyperlinks I discussed in Chapter 4.) To overcome this problem we need to also use the old method of naming the map code.

```

<body>
  <map id="shapes" name="shapes">
  </map>
  <div></div>
</body>

```

What now needs to be placed within `<map>` and `</map>` is one `<area />` element for each hotspot we wish to create. In this case, that will be four, so I'll go ahead and add those elements:

```

<map id="shapes" name="shapes">
  <area />
  <area />
  <area />
  <area />
</map>

```

Each `<area />` element now needs four attributes; `shape`, `coords`, `href`, and `alt`. The `shape` attribute has four possible values.

Table 5.2
area Attributes

circle	Specifies that the hotspot is a circle.
rect	Specifies that the hotspot has four sides.
poly	Specifies that the hotspot has either three sides, or more than four sides, (is a polygon which is not a rectangle).
default	Allows you to specify a hyperlink for any part of the image that is not covered by a hotspot. An area with a shape of default does not require the <code>coords</code> attribute.

Adding the shape attributes to our four areas, we now have:

```

<map id="shapes" name="shapes">
  <area shape="circle" />
  <area shape="rect" />
  <area shape="poly" />

```

```
<area shape="default" />
</map>
```

For each of the three actual shapes (circle, rect, and poly), we need to specify some coordinates. How the set of coordinates are built depends on the type of shape. Here's how they break down:

Table 5.3
shape **Attributes**

circle	Three numbers: the X and Y coordinates of the center of the circle and the value of the radius in pixels.
rect	Four numbers: the X and Y coordinates of the upper-left corner, and the X and Y coordinates of the lower-right corner.
poly	A variable number of pairs of numbers: one X and Y coordinate for each point in the shape. (For example, a triangle would have six numbers.)

Let's take a look at the numbers that would match the image. (The coordinates are based on the upper left corner of the image being 0,0.)

```
<map id="shapes" name="shapes">
<area href="http://www.adobe.com/" shape="circle" coords="47,
48, 41" />
<area href="http://www.google.com/" shape="rect" coords="107,
10, 182, 91" />
<area href="http://www.microsoft.com/" shape="poly" coords="198,
14, 198, 87, 282, 86, 293, 43, 240, 62, 239, 15" />
<area shape="default" />
</map>
```

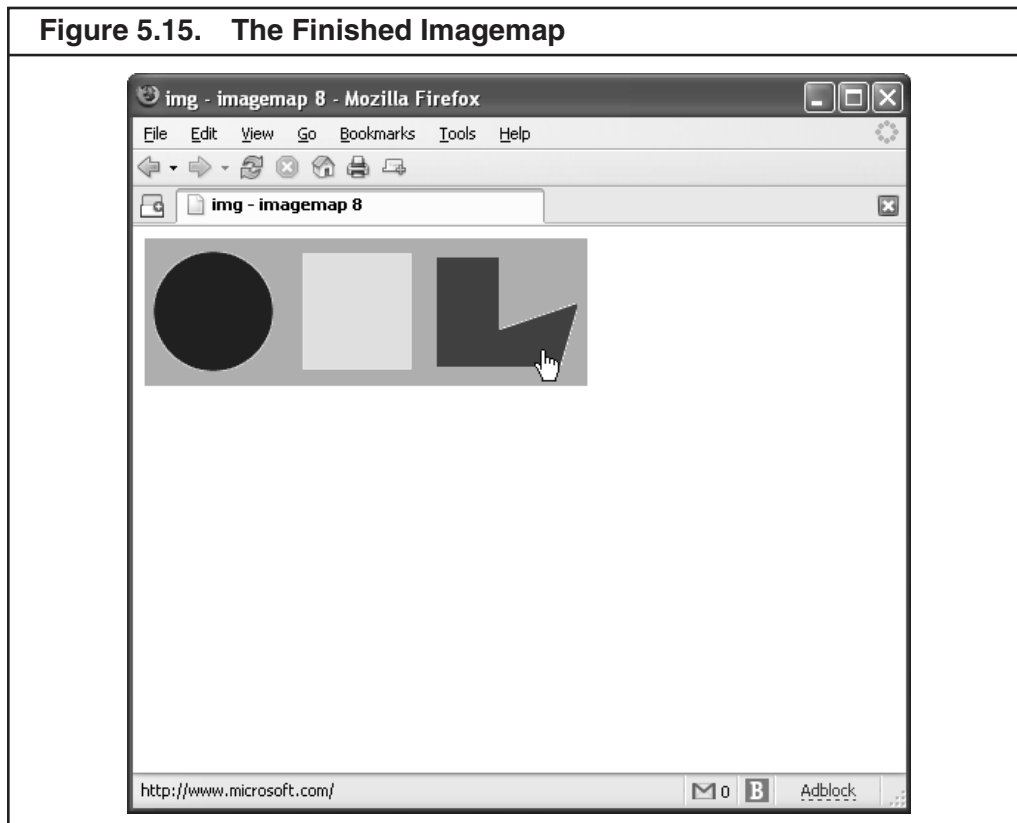
The big question at this point is how to derive the numbers. What you don't do is guess. (I did have a student admit to this at one point. She was very happy when I gave her a better solution.) In this case you really must rely on one of any number of software programs that will allow you to draw the hotspots and produce the numbers for you. Major Web site creation packages such as Microsoft's FrontPage and Macromedia's Dreamweaver have this function built in. If you don't have access to these programs, there are others out there, such as Life Software's Imagemapper, which will cost you about \$15. (<http://www.lifesoftplus.com/>)

Once you have your hotspots created, it's only a matter of adding the URLs you want them to link to, along with optional alt attributes for each hotspot. You'll also want to turn off the image's border. Here's what my final code looks like (Figure 5.15).

```
<body>
<map id="shapes" name="shapes">
<area href="http://www.adobe.com/" shape="circle" coords="47,
48, 41" href="http://www.google.com" alt="Google" />
<area href="http://www.google.com/" shape="rect" coords="107,
10, 182, 91" href="http://www.microsoft.com" alt="Microsoft"/>
<area href="http://www.microsoft.com/" shape="poly" coords="198,
14, 198, 87, 282, 86, 293, 43, 240, 62, 239, 15" href="http://
```

```
www.adobe.com" alt="Adobe"/>
<area shape="default" href="http://www.bcr.org" alt="BCR" />
</map>
<p></p>
</body>
```

Figure 5.15. The Finished Imagemap



NOTE

1. There is some controversy over how GIF should be pronounced. Some argue that it should be pronounced with a hard G as in gift, while others argue that it should be pronounced with a soft G, as a J is typically pronounced. After years of arguing for the side in favor of the hard pronunciation, I've been convinced that GIF should be pronounced with a soft G; like "jiff." For more details on the controversy and why my mind was changed, read "The GIF Pronunciation page" (www.olsenhome.com/gif/).

