CHAPTER
8

Linking Pages

So far in this book, you've worked on individual Web pages. While creating a single page is a crucial first step in building a Web site, sooner or later you'll want to wire several pages together so a Web trekker can easily jump from one to the next. After all, linking is what the Web's all about.

It's astoundingly easy to create links—officially called *hyperlinks*—between pages. In fact, all it takes is a single new element: the *anchor* element. Once you master this bit of XHTML lingo, you're ready to start organizing your pages into separate folders and transforming your humble collection of standalone documents into a full-fledged site.

Understanding the Anchor

In XHTML, you use the anchor element, <a>, to create a link. When a visitor clicks that link, the browser loads another page.

The anchor element is a straightforward container element. It looks like this:

```
<a>...</a>
```

You put the content that a visitor clicks inside the anchor element:

```
<a>Click Me</a>
```

The problem with the above link is that it doesn't *point* anywhere. To turn it into a fully functioning link, you need to supply the URL of the destination page using an

href attribute (which stands for *hypertext reference*). For example, if you want a link to take a reader to a page named *LinkedPage.htm*, you create this link:

```
<a href="LinkedPage.htm">Click Me</a>
```

For this link to work, the *LinkedPage.htm* file has to reside in the same folder as the Web page that contains the link. You'll learn how to better organize your site by sorting pages into different subfolders on page 212.

Tip: To create a link to a page in Expression Web, select the text you want to make clickable, and then hit Ctrl+K. Browse to the correct page, and Expression Web creates the link. To pull off the same trick in Dreamweaver, select the text and press Ctrl+L.

The anchor tag is an inline element (page 46)—it fits inside any other block element. That means that it's completely acceptable to make a link out of just a few words in an otherwise ordinary paragraph, like this:

```
<p>
  When you're alone and life is making you lonely<br />
  You can always go <a href="Downtown.htm">downtown</a>
</p>
```

Figure 8-1 shows this link example in action.

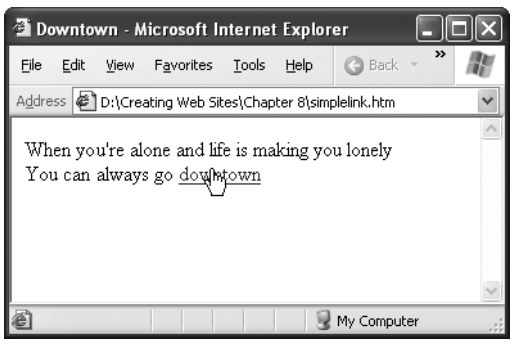

Figure 8-1:
If you don't take any other steps to customize an anchor element, its text appears in a browser with the familiar underline and blue lettering. When you move your mouse over a hyperlink, your mouse pointer turns into a hand. You can't tell by looking at a link whether it works or not–if the link points to a non-existing page, you'll get an error only after you click it.

Internal and External Links

Links can shuffle you from one page to another within the same Web site, or they can transport you to a completely different site on a far-off server. You use a different type of link in each case:

- **Internal links** point to other pages on your Web site. They can also point to other types of resources on your site, as you'll see below.
- **External links** point to pages (or resources) on other Web sites.

For example, say you have two files on your site, a biography page and an address page. If you want visitors to go from your bio page (*MyBio.htm*) to your address

HOW'D THEY DO THAT

Changing Link Colors with Style Sheets

Virtually everyone born since the year 1900 instinctively understands that blue underlined text is there to be clicked. But what if blue links are at odds with the overall look of your site? Thanks to style sheets, you don't need to play by the rules.

Based on what you learned about CSS in Chapter 6, you can quickly build a style sheet rule that changes the text color of all the link-producing anchor tags on your site. Here's an example:

```
a {
  color: fuchsia;
}
```

But watch out: making this change creates two problems. First, custom link colors change the way links work. Ordinarily, when you click a link, it turns purplish red to show that you've visited the page. Custom links, however, never change color—they retain their hue even after you click them. Web visitors who depend on the blue-to-red reminder may not appreciate your artistic flair. Second, if you apply a rule to all anchor tags, that rule will affect any bookmarks in your page. Ordinarily, bookmarks are invisible page markers (see page 221), but if you change the anchor color, your bookmarked text will also change color. This probably isn't the behavior you want.

A better way to create colorful links is to use another style sheet trick: *pseudo-selectors*. Pseudo-selectors are specialized versions of the selectors you learned about earlier. They rely on details that a browser tracks behind the scenes. For example, ordinary selectors apply rules indiscriminately to a given element, like an anchor tag. But pseudo-selectors apply rules to elements that meet certain criteria, in this case to links that are either clicked or unclicked.

Pseudo-selectors are a mid-range CSS feature, which means they don't work on very old browsers like Internet Explorer 4 and Netscape 4.

Four pseudo-selectors help you format links. They are *:link* (for links that point to virgin ground), *:visited* (for links a reader has already visited), *:active* (the color a link turns as a reader clicks it, before releasing the mouse button), and *:hover* (the color a link turns when a reader moves the mouse over it). As you can see, pseudo-selectors always start with a colon (:).

Here's a style rule that uses pseudo-selectors to create a misleading page—one where visited links are blue and unvisited links are red:

```
a:link {
  color: red;
}
a:visited {
  color: blue;
}
```

If you want to apply these rules to some, but not all, your links, you can use a class name with the pseudo-selector rule:

```
a.BackwardLink:link {
  color: red;
}
a.BackwardLink:visited {
  color: blue;
}
```

Now an anchor element needs to specify the class name to display your new style, as shown here:

```
<a class="BackwardLink" href="...">...</a>
```

page (*ContactMe.htm*), you create an internal link. Whether you store both files in the same folder or in different folders, they're part of the same Web site on the same Web server, so an internal link's the way to go.

On the other hand, if you want visitors to go from your Favorite Books page (*FavBooks.htm*) to a page on Amazon.com (*www.amazon.com*), you need an external link. Clicking an external link transports the reader out of your Web site and on to a new site, located elsewhere on the Web.

When you create an internal link, you should always use a *relative URL*, which tells browsers the location of the target page *relative to the current folder*. In other words, it gives your browser instructions on how to find the new folder by telling it to move down into or up from the current folder. (Moving *down into* a folder means moving from the current folder into a subfolder. Moving *up from* a folder is the reverse—you travel from a subfolder up into the parent folder, the one that *contains* the current subfolder.)

All the examples you've seen so far use relative URLs. For example, imagine you go to this page:

```
http://www.GothicGardenCenter.com/Sales/Products.htm
```

Say the text on the *Products.htm* page includes a sentence with this relative link to *Flowers.htm*:

```
Would you like to learn more about our purple
<a href="Flowers.htm">hydrangeas</a>?
```

If you click this link, your browser automatically assumes that you stored *Flowers.htm* in the same location as *Products.htm*, and, behind the scenes, it fills in the rest of the URL. That means the browser actually requests this page:

```
http://www.GothicGardenCenter.com/Sales/Flowers.htm
```

XHTML gives you another linking option, called an *absolute URL*, which defines a URL in its entirety, including the domain name, folder, and page. If you convert the URL above to an absolute URL, it looks like this:

```
Would you like to learn more about our purple <a href=
"http://www.GothicGardenCenter.com/Sales/Flowers.htm">hydrangeas</a>?
```

So which approach should you use? Deciding is easy. There are exactly two rules to keep in mind:

- **If you're creating an external link, you *have to* use an absolute URL.** In this situation, a relative URL just won't work. For example, imagine you want to link to the page *home.html* on Amazon's Web site. If you create a relative link, the browser assumes that *home.html* refers to a file of that name on *your* Web site. Clicking the link won't take your visitors where you want them to go (and may not take them anywhere at all, if you don't have a file named *home.html* on your site).
- **If you're creating an internal link, you really, really should use a relative URL.** Technically, either type of link works for internal pages. But relative URLs have several advantages. First, they're shorter and make your XHTML more readable and easier to maintain. More importantly, relative links are flexible. You can rearrange your Web site, put all your files into a different folder, or even change your site's domain name without breaking relative links.

One of the nicest parts about relative links is that you can test them on your own computer and they'll work the exact same way as they would online. For example, imagine you've developed the site *www.GothicGardenCenter.com* on your PC and

you store it inside the folder *C:\MyWebSite* (that'd be *Macintosh HD/MyWebSite*, in Macintosh-ese). If you click the relative link that leads from the *Products.htm* page to the *Flowers.htm* page, the browser looks for the target page in the *C:\MyWebSite* (*Macintosh HD/MyWebSite*) folder.

Once you polish your work to perfection, you upload the site to your Web server, which has the domain name *www.GothicGardenCenter.com*. Because you used relative links, you don't need to change anything. When you click a link, the browser requests the corresponding page in *www.GothicGardenCenter.com*. If you decide to buy a new, shorter domain name like *www.GGC.com* and move your Web site there, the links still keep on working.

Note: Internet Explorer has a security quirk that appears when you test pages with external links. If you load a page from your hard drive, and then click a link that points somewhere out on the big bad Web, Internet Explorer opens a completely new window to display the target page. That's because the security rules that govern Web pages on your hard drive are looser than those that restrict Web pages on the Internet, so Internet Explorer doesn't dare let them near each other. This quirk disappears once you upload your pages to the Web.

FREQUENTLY ASKED QUESTION

Navigating and Frames

How do I create a link that opens a page in a new browser window?

When visitors click external links, you might not want to let them get away from your site that easily. Web developers use a common trick that opens external pages in separate browser windows (or in a new tab, depending on the browser's settings). This way, your site remains open in the visitor's original window, ensuring the visitor won't forget about you.

To make this work, you need to set another attribute in the anchor element–the *target*. Here's how:

```
<a href="LinkedPage.htm" target="_blank">
Click Me</a>
```

The target attribute names the frame where a browser should display the destination page (you'll learn more about frames in Chapter 10). The value *_blank* indicates that the link should load the page in a new, empty browser window.

Before you start adding the target attribute to all your anchors, it's important to recognize two drawbacks with this technique:

- **It breaks strict validation.** The target attribute isn't allowed in XHTML 1.0 strict. If you want to use it, you should change your doctype to XHTML 1.0 transitional (page 30). Other, clumsier workarounds are possible–for example, Web designers sometimes use JavaScript (see Chapter 14) to open a new window when visitors click a link. The target feature may also reappear in a future version of CSS.
- **It may not always work.** Some vigilant *pop-up blockers* intercept this type of link and prevent the new window from appearing altogether. (Pop-up blockers are standalone programs or browser features designed to prevent annoying pop-up ads from appearing.) Internet Explorer 6 (and later) includes its own pop-up blocker, but its standard settings allow links that use the *target="_blank"* attribute.

Some people love the new-window feature, while others think it's an immensely annoying and disruptive act of Web site intervention. If you use it, apply it sparingly on the occasional link.

Relative Links and Folders

So far, all the relative link examples you've seen have assumed that both the *source page* (the one that contains the link) and the *target page* (the destination you arrive at when you click the link) are in the same folder. There's no reason to be quite this strict. In fact, your Web site will be a whole lot better organized if you store groups of related pages in *separate* folders.

Consider the Web site shown in Figure 8-2.

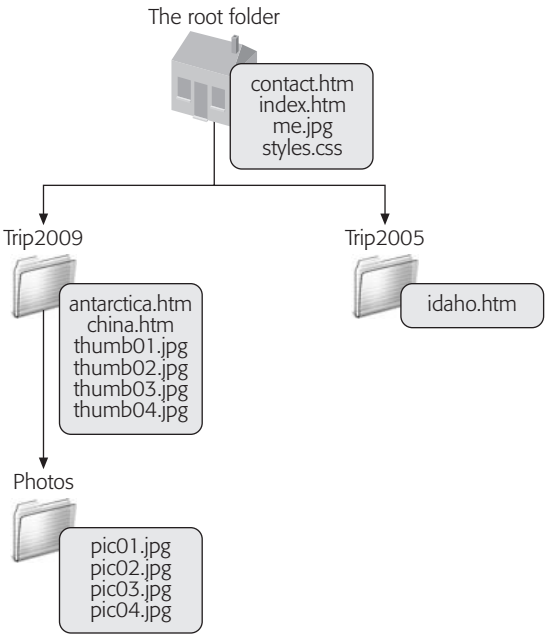

Figure 8-2:
*This diagram maps out the structure of a very small Web site featuring photos taken on a trip. The root folder contains a style sheet used across the entire site (*styles.css*) and two XHTML pages. Two subfolders, Trip2005 and Trip2009, contain additional pages. The Trip2009 folder holds thumbnail images of pictures taken on one of the trips. For each thumbnail, there's a corresponding full-size picture in the Photos subfolder.*

Note: The root folder is the starting point of your Web site—it contains all your other site files and folders. Most sites include a page with the name *index.htm* or *index.html* in the root folder. This is known as the *default page*. If a browser sends a request to your Web site domain without supplying a file name, the Web server sends back the default page. For example, requesting *www.TripToRemember.com* automatically returns the default page *www.TripToRemember.com/index.htm*.

This site uses a variety of relative links. For example, imagine you need to create a link from the *index.htm* page to the *contact.htm* page. Both pages are in the same folder, so all you need is a relative link:

```
<a href="contact.htm">About Me</a>
```

You can also create more interesting links that move from one folder to another, which you'll learn how to do in the following sections.

Tip: If you'd like to try out this sample Web site, you'll find all the site's files on the Missing CD page at *www.missingmanuals.com*. Thanks to the magic of relative links, all the links will work no matter where on your computer (PC or Mac) you save the files, so long as you keep the same subfolders.

Moving down into a subfolder

Say you want to create a relative link that jumps from an *index.htm* page to a page called *antarctica.htm*, which you've put in a folder named Trip2009. When you write the relative link that points to *antarctica.htm*, you need to include the name of the Trip2009 subfolder, like this:

```
See pictures from <a href="Trip2009/antarctica.htm">Antarctica</a>
```

This link gives the browser two instructions—first to go into the subfolder Trip2009, and then to get the page *antarctica.htm*. In the link, you separate the folder name ("Trip2009") and the file name ("antarctica.htm") with a slash character (/). Figure 8-3 shows both sides of this equation.

Interestingly, you can use relative paths in other XHTML elements, too, like the <style> element and the <img> element. For example, imagine you want to display the picture *photo01.jpg* on the page *index.htm*. This picture is two subfolders away, in a folder called Photos, which is inside Trip2009. But that doesn't stop you from pointing to it in your <img> element:

```
<img src="Trip2009/Photos/photo01.jpg" alt="A polar bear" />
```

Using this technique, you can dig even deeper into subfolders of subfolders of subfolders. All you need to do is add the folder name and a slash character for each subfolder, in order.

But remember, relative links are always *relative to the current page*. If you want to display the same picture, *photo01.jpg,* in the *antarctica.htm* page, the <img> element above won't work, because the *antarctica.htm* page is actually in the Trip2009 folder. (Take a look back at Figure 8-2 if you need a visual reminder of the site structure.) From the Trip2009 folder, you only need to go down one level, so you need this link:

```
<img src="Photos/photo01.jpg" alt="A polar bear" />
```

By now, you've probably realized that the important detail lies not in how many folders you have on your site, but in how you organize the subfolders. Remember, a relative link always starts out from the *current* folder, and works it way up or down to the folder holding the target page.

Tip: Once you start using subfolders, you shouldn't change any of their names or move them around. That said, many Web page editors (like Expression Web) are crafty enough to help you out if you do make these changes. When you rearrange pages or rename folders inside these programs, they adjust your relative links. It's yet another reason to think about getting a full-featured Web page editor.

Figure 8-3:
Using a relative link, you can jump from the main index.htm page (top) to a page with picture thumbnails (bottom). Each picture is itself a link–visitors click it to see a larger-sized version of the photo.

Moving up into a parent folder

The next challenge you'll face is going *up* a folder level. To do this, use the character sequence ../ (two periods and a slash). For example, to add a link in the *antarctica.htm* page that brings the reader back to the *index.htm* page, you'd write a link that looks like this:

```
Go <a href="../index.htm">back</a>
```

And as you've probably guessed by now, you can use this command twice in a row to jump up two levels. For example, if you have a page in the Photos folder that leads to the home page, you need this link to get back there:

```
Go <a href="../../index.htm">back</a>
```

For a more interesting feat, you can combine both of these tricks to create a relative link that travels up one or more levels, and then travels down a different path. For example, you need this sort of link to jump from the *antarctica.htm* page in the Trip2009 folder to the *idaho.htm* page in the Trip2005 folder:

```
See what happened in <a href="../Trip2005/idaho.htm">Idaho</a>
```

This link moves up one level to the root folder, and then back down one level to the Trip2005 folder. You follow the same process when you browse folders to find files on your computer.

Moving to the root folder

The only problem with the relative links you've seen so far is that they're difficult to maintain if you ever reorganize your Web site. For example, imagine you have a Web page in the root directory of your site. Say you want to feature an image on that page that's stored in the *images* subfolder. You use this link:

```
<img src="images/flower.gif" alt="A flower" />
```

But then, a little later on, you decide your Web page really belongs in *another* spot—a subfolder named Plant—so you move it there. The problem is that this relative link now points to *plant/images/flower.gif*, which doesn't exist—the Images folder isn't a subfolder in Plants, it's a subfolder in your site's root folder. As a result, your browser displays a broken link icon.

There are a few possible workarounds. In programs like Expression Web, when you drag a file to a new location, the XHTML editor updates all the relative links automatically, saving you the hassle. Another approach is to try to keep related files in the same folder, so you always move them as a unit. However, there's a third approach, called *root-relative* links.

So far, the relative links you've seen have been *document-relative*, because you specify the location of the target page relative to the current document. *Root-relative* links point to a target page *relative to your Web site's root folder*.

Root-relative links always start with the slash (/) character (which indicates the root folder). Here's the <img> element for *flower.gif* with a root-relative link:

```
<img src="/images/flower.gif" alt="A flower" />
```

The remarkable thing about this link is that it works no matter where you put the Web page that contains it. For example, if you copy this page to the Plant subfolder, the link still works, because the first slash tells your browser to start at the root folder.

The only trick to using root-relative folders is that you need to keep the real root of your Web site in mind. When using a root-relative link, the browser follows a simple procedure to figure out where to go. First, it strips all the path and file name information out of the current page address, so that only the domain name is left.

Then it adds the root-relative link to the end of the domain name. So if the link to *flower.gif* appears on this page:

```
http://www.jumboplants.com/horticulture/plants/annuals.htm
```

The browser strips away the */horticulture/plants/annuals.htm* portion, adds the relative link you supplied in the *src* attribute (*/images/flower.gif*), and looks for the picture here:

```
http://www.jumboplants.com/images/flower.gif
```

This makes perfect sense. But consider what happens if you don't have your own domain name. In this case, your pages are probably stuck in a subfolder on another Web server. Here's an example:

```
http://www.superISP.com/~user9212/horticulture/plants/annuals.htm
```

In this case, the domain name part of the URL is *http://www.superISP.com*, but for all practical purposes, the root of your Web site is your personal folder, *~user9212*. That means you need to add this detail into all your root-relative links. So to get the result you want with the *flower.gif* picture, you need to use this messier root-relative link:

```
<img src="/~user9212/images/flower.gif" alt="A flower" />
```

Now the browser keeps just the domain name part of the URL (*http://www.superISP.com*) and adds the relative part of the path, starting with your personal folder (*/~user9212*).

Linking to Other Types of Content

Most of the links you write will point to bona fide XHTML Web pages. But that's not your only option. You can link directly to other types of files as well. The only catch is that it's up to the browser to decide what to do when someone clicks a link that points to a different type of file.

Here are some common examples:

- **You can link to a JPEG, GIF, or PNG image file (page 184).** When visitors click a link like this, the browser displays the image in a browser window without any other content. Web sites often use this approach to let visitors take a close-up look at large graphics. For example, the Trip2009 Web site in the previous section has a page chock full of small image thumbnails. Click one of those, and the full-size image appears.
- **You can link to a specialized type of file, like a PDF file, a Microsoft Office document, or an audio file (like a WAV or an MP3 file).** When you use this technique, you're taking a bit of a risk. These links rely on a browser having a plug-in that recognizes the file type, or on your visitors having a suitable program installed on his PC. If the computer doesn't have the right software, the only thing your visitors will be able to do is download the file (see the next point), where it will sit like an inert binary blob. However, if a browser has the right plug-in, a small miracle happens. The PDF, Office, or audio file opens up right inside the browser window, as though it were a Web page!

UP TO SPEED

The Rules for URLs

The rules for correctly writing a URL in an anchor element are fairly strict, and there are a few common mistakes that creep into even the best Web pages. Here are some pointers to help you avoid these headaches:

- When you create an absolute URL, you have to start it with its protocol (usually http://). You don't need to follow this rule when typing a URL into a browser, however. For example, if you type *www.google.com*, most browsers are intelligent enough to assume the http:// part. However, in an XHTML document, it's mandatory.
- Don't mix up the backslash (\) and the ordinary forward slash (/). Windows uses the backslash in file paths (like *C:\Windows\win.ini*). In the Web world, the forward slash separates subfolders (as in *http://www.ebay.com/Help/index.html*). Once again, many browsers tolerate backslash confusion, but the same mistake in an anchor element breaks your links.
- Don't ever use file paths instead of a URL. It's possible to create a URL that points to a file on your computer using the *file* protocol (as in *file:///C:/Temp/myPage.htm*). However, this link won't work on anyone else's computer, because they won't have the same file on their hard drive. Sometimes, design tools like Expression Web may insert one of these so-called local URLs (for example, if you drag and drop a picture file into your Web page). Be vigilant–check all your links to make sure this doesn't happen.
- Don't use spaces or special characters in your file or folder names, even if these special characters are allowed. For example, it's perfectly acceptable to put a space in a file name (like *My Photos.htm*), but in order to request this page, the browser needs to translate the space into a special character code (*My%20Photos.htm*). To prevent this confusion, steer clear of anything that isn't a number, letter, dash (–), or underscore (_).

- **You can link to a file you want others to download.** If a link points to a file of a specialized type and the browser doesn't have the proper plug-in, visitors get a choice: They can ignore the content altogether, open it using another program on their computer, or save it on their PC. This is a handy way to distribute large files (like a ZIP file featuring your personal philosophy of planetary motion).
- **You can create a link that starts a new email message.** It's easy to build a link that fires up your visitors' favorite email program and helps them send a message to you. Page 337 has all the details.

Image Links and Image Maps

It's worth pointing out that you can also turn images into links. The trick is to put an <img> element inside an anchor element (<a>), like this:

```
<a href="LinkedPage.htm"><img src="MyPic.gif" alt="My Picture" /></a>
```

XHTML adds a thick blue border to pictures to indicate they're clickable. Usually, you want to turn this clunky-looking border off using the style sheet border properties described on page 167. When a visitor hovers her cursor over a linked picture, the cursor changes to a hand.

In some cases, you might want to create distinct clickable regions, called hotspots, *inside* a picture. For example, consider Figure 8-4.

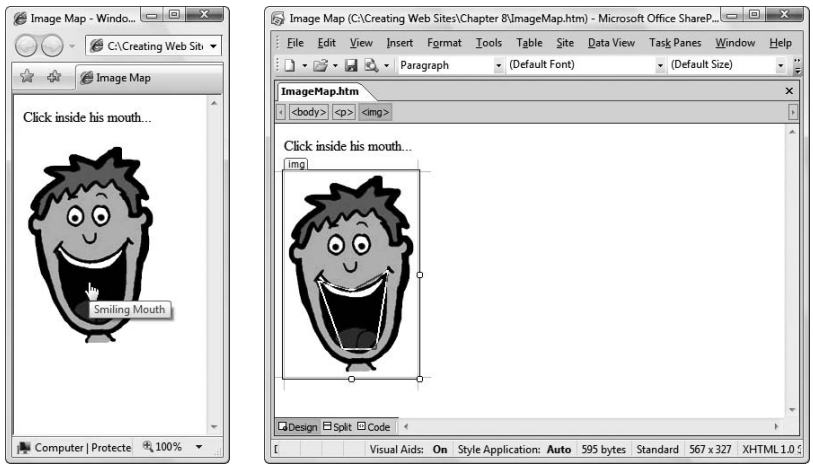

Figure 8-4: *Left: An ordinary picture, courtesy of the <img> element.*

Right: An irregularly shaped region inside the mouth becomes a hotspot—a clickable region that takes visitors to another page. In this example, you can see the hotspot because it's being edited in Expression Web. Ordinarily, you can't see hotspots when you look at a picture in a Web browser.

To add a hotspot to a picture, you start by creating an *image map* using the <map> element. This part's easy—all you do is choose a unique name for your image map so you can refer to it later on:

```
<map id="FaceMap" name="FaceMap">
</map>
```

Note: If you noticed that the <map> element uses two attributes that duplicate the same information (*id* and *name*), you're correct. Although in theory just the id attribute should do the trick, you need to keep the name attribute there to ensure compatibility with a wide range of browsers.

Then you need to define each hotspot, which you do inside the <map> element, between its start and end tags. You can add as many hotspots as you want, although they shouldn't overlap. (If they do, the one that's defined first takes precedence.)

To define each hotspot in an image, you add an <area> element. The area element identifies three important details: the target page a visitor goes to after clicking the hotspot (the *href* attribute), the shape of the hotspot (the *shape* attribute), and the exact dimensions of the shape (the *cords* attribute). Much like an image, the <area> element requires an alt attribute with some alternate text that describes the image map to search engines and ancient text-only browsers.

Here's a sample <area> element:

```
<area href="MyPage.htm" shape="rect" coords="5,5,95,195" alt="A clickable
rectangle" />
```

This hotspot defines a rectangular region. When visitors click it, they go to *MyPage.htm*.

The *shape* attribute supports three types of shape, each of which corresponds to a different value for the attribute. You can use circles (*circle*), rectangles (*rect*), and multi-edged shapes (*poly*). Once you choose your shape, you need to supply the coordinates, which are a bit trickier to interpret. To understand hotspot coordinates, you first need to understand how browsers measure pictures (see Figure 8-5).

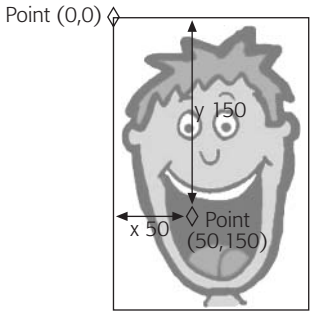

Figure 8-5:
Browsers designate the top-left corner in a picture as point (0, 0). As you move down the picture, the y-coordinate (the second number) gets bigger. For example, the point (0, 100) is at the left edge of the picture, 100 pixels from the top. As you move to the right, the x-coordinate gets bigger. That means the point (100, 0) is at the top of a picture, 100 pixels from the left edge.

You enter image map coordinates as a list of numbers separated by commas. For a circle, list the coordinates in this order: center point (x-coordinate), center point (y-coordinate), radius. For any other shape, supply the corners in order as a series of x-y coordinates, like this: x1, y1, x2, y2, and so on. For a polygon, you supply every point. For a rectangle, you only need two points—the top-left corner, and the bottom-right corner.

You define the rectangle mentioned earlier by these two points: (5, 5) at the top-left and (95, 195) at the bottom right. You define the more complex polygon that represents the mouth region in Figure 8-4 like this:

```
<area href="MyPage.htm" shape="poly" alt="Smiling Mouth"
coords="38, 122, 76, 132, 116, 110, 102, 198, 65, 197" />
```

In other words, your browser creates this shape by drawing lines between these five points: (38, 122), (76, 132), (116, 110), (102, 198), and (65, 197).

Tip: Getting coordinates correct is tricky. Many Web page editors, like Expression Web and Dreamweaver, have built-in hotspot editors that let you create an image map by dragging shapes over your picture, which is a lot easier than trying to guess the correct values. To use this tool in Dreamweaver, select a picture, and then look for the three hotspot icons (circle, square, and polygon) in the Properties panel. In Expression Web, you use similar icons in the Picture toolbar. If the Picture toolbar isn't visible, right-click the picture, and then select Show Pictures Toolbar.

Once you perfect all your hotspots, there's one step left: to apply the hotspots to the image by adding the *usemap* attribute to your <img> element. The *usemap* attribute is the same as the name of the image map, but it starts with the number-sign character (#), which tells browsers that you've defined an image map for the picture on the current page:

```
<img src="face.gif" usemap="#FaceMap" alt="Smiling Face" />
```

Here's the complete XHTML for the mouth hotspot example:

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

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

<head>
  <title>Image Map</title>
  <style type="text/css">
    img {
      border-style: none;
  }
  </style>
</head>

<body>
  <p>Click inside his mouth...</p>
  <p>
    <map id="FaceMap" name="FaceMap">
      <area href="http://edcp.org/factsheets/handfoot.html" shape="poly"
      coords="38, 122, 76, 132, 116, 110, 102, 198, 65, 197" alt="Smiling
Mouth" />
    </map>
    <img src="face.gif" usemap="#FaceMap" alt="Smiling Face" />
  </p>
</body>

</html>
```

The hotspots you create are invisible (unless you draw lines on your picture to indicate where they are). When visitors hover over them, their mouse pointers change to a hand. Clicking a hotspot has the same effect as clicking an ordinary <a> link—visitors get transported to a new page.

Note: It's tempting to use image maps to create links in all kinds of graphics, including buttons you may custom-design in an image editor. Hold off for a bit. Sophisticated Web sites like yours can go many steps further with menus and buttons, but to implement these nifty tricks you need the JavaScript know-how you'll learn in Chapters 14 and 15.

Adding Bookmarks

Most links lead from one page to another. When you make the jump to a new page, the browser plunks you down at the very top of the page. But you can also create links to specific parts of a page. This is particularly useful if you create long, scrolling pages and you want to direct your visitors' attention to a particular passage.

You can create links to another position on the *current* page (see Figure 8-6), or to a specific place in *another* Web page. The place you send your reader is technically called a *fragment*.

Creating a link that points to a fragment is a two-step process. First, you need to identify that fragment. Imagine you want to send a visitor to the third level-3 heading in a Web page named *sales.htm*. To make this work, you need to embed a marker just before that level-3 heading. XHTML calls this marker a *bookmark*.

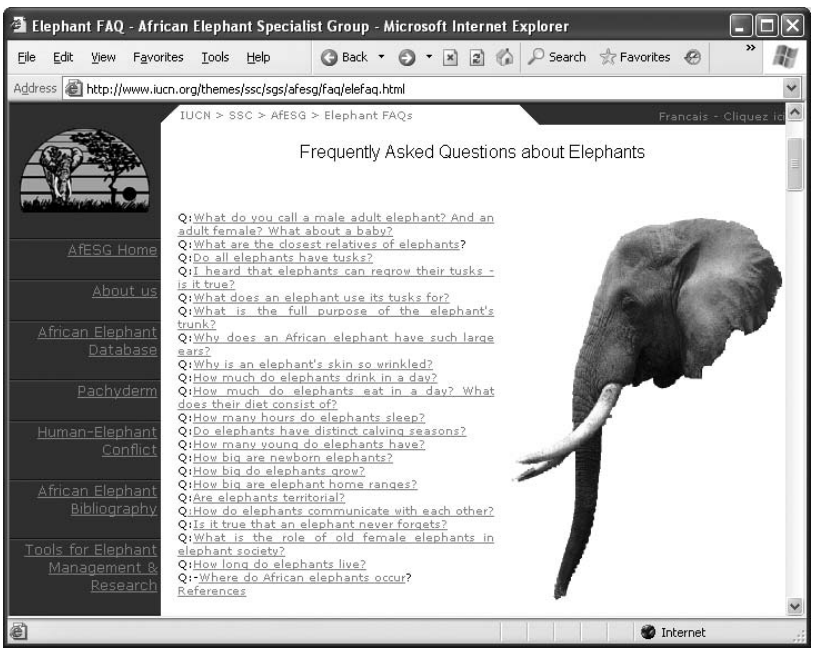

Figure 8-6: *FAQ (frequently asked questions) pages are one of the best examples of bookmarks at work. Often, an entire FAQ is one long page, with a series of bookmark links at the top that let you jump to just the topic you're interested in. You could break a FAQ into separate pages, but readers wouldn't be able to scan through the whole list of questions in order, and they wouldn't have a way to print the entire document at once.*

To create a bookmark, you use the <a> anchor element, but with a twist: You don't supply an *href* attribute, because bookmarks don't actually lead anywhere—they simply identify fragments. What you *do* supply is a *name* attribute, which gives your bookmark a descriptive name. It's up to you whether you put any text inside an anchor—technically you don't need to, but most people find it easier to lock a bookmark to a specific word or title on the target page.

Here's an example:

```
...
<h3><a name="Canaries">Pet Canaries</a></h3>
<p>Pet canary sales have plummeted in the developed world, due in large part
to currency fluctuations and other macroeconomic forces.</p>
...
```

In this example, you create a bookmark named Canaries that drops visitors at the heading "Pet Canaries."

Once you create a bookmark, you can write a URL that points to it. The trick is to add the bookmark information to the end of the URL. To do this, you add the number-sign symbol (#), followed by the bookmark name.

For example, to send a reader to a bookmark named Canaries in the *sales.htm* page, here's the link you use:

```
Learn about recent developments in <a href="sales.htm#Canaries">canary
sales</a>.
```

When you click this link, the browser heads to the *sales.htm* page and scrolls down the page until it encounters the Canaries bookmark, The browser then displays, at the very top of the browser window, the text that starts with the heading "Pet Canaries."

Tip: If your bookmark is near the bottom of a page, a browser might not be able to scroll the bookmark all the way to the top of its window. Instead, the bookmarked section will appear somewhere in the middle of the browser window. This happens because the browser hits the bottom of the page, and can't scroll down any further. If you think there's some potential for confusion (perhaps because you have several bookmarked sections close to each other at the bottom of a page), you can add a few
 elements at the end of your document, which lets the browser scroll down farther.

Sometimes you want to create a link that points to a bookmark in your *current* page. In this case, you don't need to specify a page name at all. Just start with the number sign, followed by the bookmark name:

```
Jump to the <a href="#Canaries">canary</a> section.
```

Using bookmarks effectively is an art. Resist the urge to overcrowd your pages with links that direct readers to relatively small sections of content. Only use bookmarks to tame large pages that take several screenfuls of scrolling.

When Good Links Go Bad

Now that you've learned all the ways to build links, it's a good time to consider what can go wrong. Links that go to pages on the same site can break when you rename or move files or folders. Links to other Web sites are particularly fragile—they can break at any time, without warning. You won't know that anything's gone wrong until you click the link and get a "Page Not Found" error message.