APPENDIX **A**

HTML BASICS

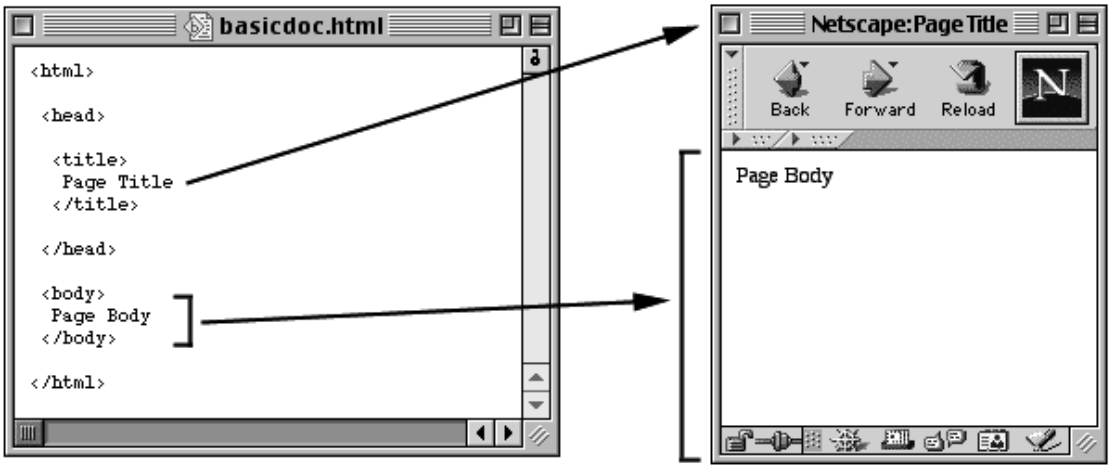


FIGURE A.1 The structure of an HTML document.

A.1 HTML LANGUAGE AND DOCUMENT STRUCTURE

Figure A.1 shows a minimal HTML document and its rendition in a browser.

A markup instruction, called a **tag**, is identified with angle brackets <>. A tag with no / (forward slash) is called a **start tag**, and a tag with a / after the first angle bracket is called an **end tag**. A matching pair, start tag and end tag, is called an **element**. Everything between the start and end tag of an element is called the **content** of the element. The `<html> ... </html>` element can contain any other HTML element, subject to some constraints. The `<head> ... </head>` element's contents define information about the Web page the document is to create, and the `<body> ... </body>` element's contents define the actual stuff that goes into the Web page. This is shown in Figure A.1. The content of the `<title> ... </title>` element is displayed at the top of the browser window, whereas the content of the `<body> ... </body>` element is displayed as the Web page. For simplicity, we will not continue to refer to elements using full HTML syntax. For example, we will just say "the `title` element."

Below are some key HTML syntax rules :

- All HTML elements should be **nested**. For example, the following elements are *not* nested:

```
<title><head>…</title></head>
```

A browser builds the parse tree for the document based upon element containment. If elements are not nested, the containment hierarchy is not well defined. If you go back and examine the parse trees shown in Section 2.1, you will see that they reflect the containment relationships of the elements used there.

- Browsers are particular about extra spaces in tags. For example,

```
< body> , < /body>, </ body>, and </body >
```

violate HTML syntax rules. However,

```
<body >
```

is permissible. When we explore HTML attributes, you will see why spaces at the end of a start tag are allowed.

- Outside of HTML tags, browsers recognize only single spaces in an HTML document. They have to observe single spaces in text so that words don't run together. The following document structure is equivalent to the one shown in Figure A.1 as far as a browser is concerned:

```
<html><head><title>Page Title</title></head><body>Page Body</body></html>
```

Just as when you write a computer program, extra spaces, tabs, and blank lines are to make it organized and more readable for humans. Based upon containment (nesting), this HTML code generates the same parse tree as the document of Figure A.1.

The following is not really an HTML syntax rule but an issue of good practice: All HTML file names should be appended with one of the extensions `.html` or `.htm`. You can see the HTML file's name in Figure A.1 at the top of the window in which it is displayed (not in the browser window).

NOTE

Like other programming languages, HTML provides for **comments**, which the HTML processor will completely ignore. Comments are only for human benefit, allowing notes to be placed in the document. In HTML, comments can be

```
<!-- on a single line -->
<!-- or on
multiple lines -->
```

A.2 CREATING AND VIEWING A PAGE

Because every HTML document you will create while reading this book has the skeletal form shown in Figure A.1, you should create one using your favorite plain text editor. Name it something like `skeleton.html`. Each time you create a new page, simply make a

copy of your skeleton file. That way you won't have to retype all the basic tags every time you make a new page. We recommend *Notepad* if you are using a PC and *SimpleText* if you are using a Mac. These editors come with the respective operating systems, and they create plain ASCII files. We discourage use of HTML WYSIWYG editors for your purposes in this book because they often generate sloppy HTML. Moreover, you will need to understand and be able to produce HTML code by hand as you write CGI programs that generate HTML as output. There are some plain text editors listed on the Web site that are geared more toward formatting programming code. For example, they help keep track of your levels indentation. Check those out if you wish.

A common way to view a Web page is drag the icon for the HTML file onto the icon of your favorite browser. Keep a shortcut icon for the browser handy (such as on the desktop) to load your pages quickly. Depending on your operating system, you may figure out a method you like better. For example, Microsoft's Windows typically gives HTML files an Internet Explorer icon, and double-clicking the file renders the page in IE rather than opening it in Notepad. In that case, you may wish to keep a Notepad shortcut handy on the desktop to drag a file onto for editing, or a Netscape shortcut if you like that browser better or simply wish to compare your Web pages in both browsers. Whatever the case, you should quickly be able to adapt to your environment.

When a browser displays one of your *local* HTML files, you will see a `file` **URL**, rather than an `http` URL, in the address field of the browser. By "local" we mean that the file is sitting on your hard drive rather than on a Web server somewhere. Figure A.2 shows a typical `file` URL for each of Windows and Mac.

While HTTP transactions over the Internet require an HTTP URL (`http:// . . .`), local viewing uses the **`file` protocol**. In principle, a `file` transaction works the same way, but the browser just grabs any files necessary for the Web page off of the local hard drive. The Mac file URL in Figure A.2 shows the complete file URL structure, whereas Windows typically suppresses mention of the `file` protocol and begins the URL with the drive letter. Either way, the address field gives the absolute path to the HTML document from the local hard drive. In practice, you won't have to deal directly with `file` URLs (typing one in for example), but you should be cognizant of the fact that no Internet connection is required to view Web pages locally, and of what to expect in such cases.

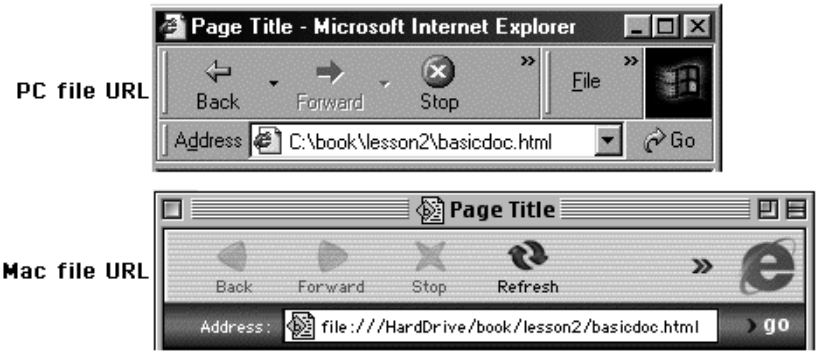


FIGURE A.2 File URLs resulting from local browser renditions of `basicdoc.html`.

Understanding local `file` viewing is important because of the general strategy for developing and maintaining a Web site. Here are some tips.

- Make a folder with the same name as the public directory in your Web server account. This folder is a local version of your site's root directory and will contain a local copy of your Web site.
- Create and edit your files locally. Once you are satisfied with them, copy them onto the Web server with an FTP client.
- If you wish to update some pages in your Web site, first make the changes to the local copies. When you have tested the changes locally and are satisfied, ftp[1] them to the Web server. This way you maintain two copies of your Web site.
- It is imperative that you ftp your HTML files to the Web server as plain text, rather than some other format, such as binary.

A.3 INLINE AND BLOCK ELEMENTS

Now, let's get back to HTML. The `body` element contains all other markup elements and text that make up the Web page. In fact, you will not see any other elements that are allowed in the `head` section until Section A.7. The first category of elements we use in the body is **inline elements.** These elements (with the exception of `br`) do not cause a line break in the flow of the Web page. They are typically used "inline" within sentences or paragraphs. Table A.1 lists the inline elements we feature here, and Figure A.3 demonstrates their use.

Here are two important things to note:

- Browsers automatically wrap text around according to the width of the browser window. (You can see that by pulling up the source file for Figure A.3 on the Web site and resizing the window.) If you want a line break at a particular spot, you have to force it with a `<br />` element.
- `<br />` is an **empty element**, meaning that it has no content. All empty elements should have a space followed by a / before the closing bracket. This notation tells the browser that it need not wait for an end tag, which helps the browser to construct the

TABLE A.1 Inline Elements in HTML

ELEMENT	DESCRIPTION
`<b>...</b>`	Boldface text
`<big>...</big>`	Bigger text
`<i>...</i>`	Italic text
`<small>...</small>`	Smaller text
`<tt>...</tt>`	Text in a monospace (fixed-width "typewriter" or "teletypewriter") font
` `	Line break (subsequent text begins on a new line)

[1]For convenience, we will use lowercase "ftp" as a verb. Here, to "ftp a file to the server" means to "use an FTP client to copy the file onto the server." It's not every day you get to make up a new verb!

inline.html

```
<html>
  <head>
    <title>Inline Elements</title>
  </head>
  <body>
    <b>Boldface</b> and <i>Italic</i> are standard markup
    instructions. They can be nested to produce text that is both
    <b><i>bold and italic</i></b>. The <i><b>nesting order</b></i>
    is unimportant so long as the inline elements are properly
    nested. Inline elements are not just for whole words or
    phrases, but can be applied inside <b>wo</b>r<i>ds</i>. The
    HTtML code may look funny in that case, but the tags do not
    appear in the web page. Since browsers ignore spaces, this

    will not cause a line break just as extra         spaces between
    words will not have any effect. We will have to force a line break
    here.<br /> <big>Big text</big> or <small>small text</small>,
    how do you like your text? We like monospace text for programming code
    <br />
    <tt>for(x=1;x<=10;x++)</tt>
    <br />
    so it stands out from the rest of the text. <br /><br />
    It's all just text. Text.Text.Text.
  </body>
</html>
```

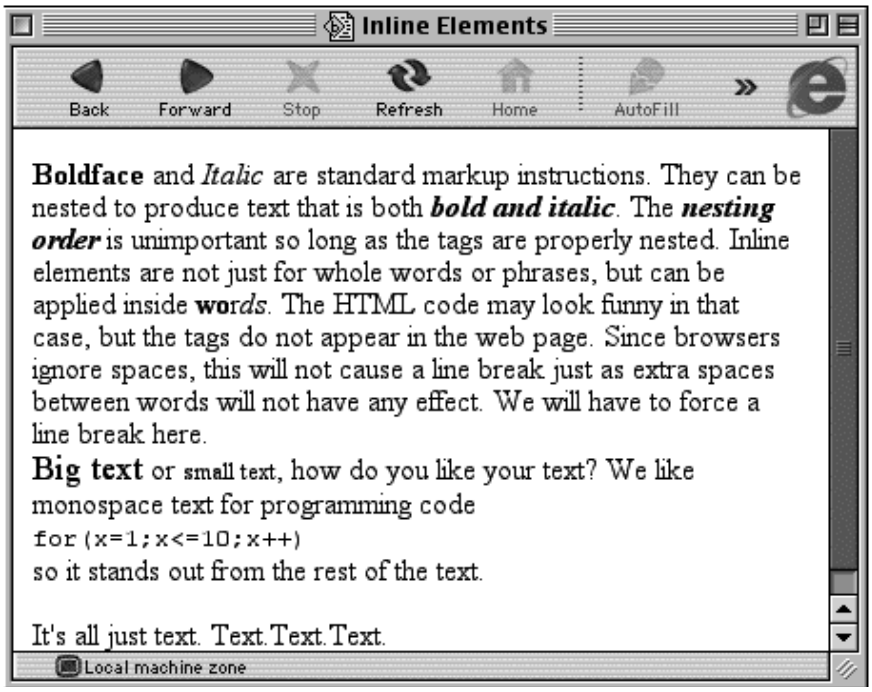


FIGURE A.3 A simple Web page using inline elements.

TABLE A.2 Block Elements

ELEMENT	DESCRIPTION
<h*x*>...</h*x*>	Headings, six levels (*x* = 1, 2, 3, 4, 5, 6), creates header blocks from largest (*x* = 1) to smallest (*x* = 6)
<hr />	Horizontal rule (solid horizontal division across page)
...	Paragraph
<pre>...</pre>	Preformatted text, typically rendered in a monospace font

parse tree efficiently. (That is, if there is no content, then terminate the branch.) Conceptually, empty markup elements are like sentences with intransitive verbs: "Cause a line break." There is no content (direct object) for the instruction to act upon. In contrast, nonempty elements are like sentences with transitive verbs: "Apply boldface to the enclosed text." Here the instruction must act upon a specific quantity.

The second category of elements we use in the `body` section are **block elements**. These elements create "block" structures in the flow of the Web page. Each block is automatically preceded and followed by a line break. Table A.2 lists the block elements we consider here, and Figure A.4 demonstrates their use.

Here are some things to note:

- Since block elements are automatically preceded and followed by line breaks, they occupy an entire "block" in the page, from the left to the right margin.
- Inline elements are used inside block elements. We used two inline elements inside the second paragraph. Block elements provide "block scope," and inline elements fine-tune text within that scope. Using block elements inside of inline elements does not make logical sense.
- The `pre` element preserves spaces. (I know, we already said that.) But the point here is to type several consecutive spaces if you need them, rather than using the Tab key. Spaces are absolute quantities, but tabs will be interpreted differently by different browsers. Also, blank lines are preserved, so don't use the `p` or `br` elements inside preformatted text. However, if you wish, you can use `b` or `i` to force fancier formatting of the monospace font.

NOTE

The best advice for the beginning Web page author (besides practicing a lot) is to forget about perfection. Pages rarely look the same in two different browsers. For an easily perceived difference, Netscape's default font size is nearly always smaller than that of Internet Explorer. You can easily see this by contrasting Figures A.3 and A.4. The former features Explorer, whereas the latter features Netscape. In practice this is no big deal. Just get your page looking pretty good, and don't worry about it.

A.4 HTML ATTRIBUTES

Elements tell the browser what to do in a minimal sense. The `hr` element makes a fine example. It just tells the browser to make a horizontal rule, but does not say how wide or

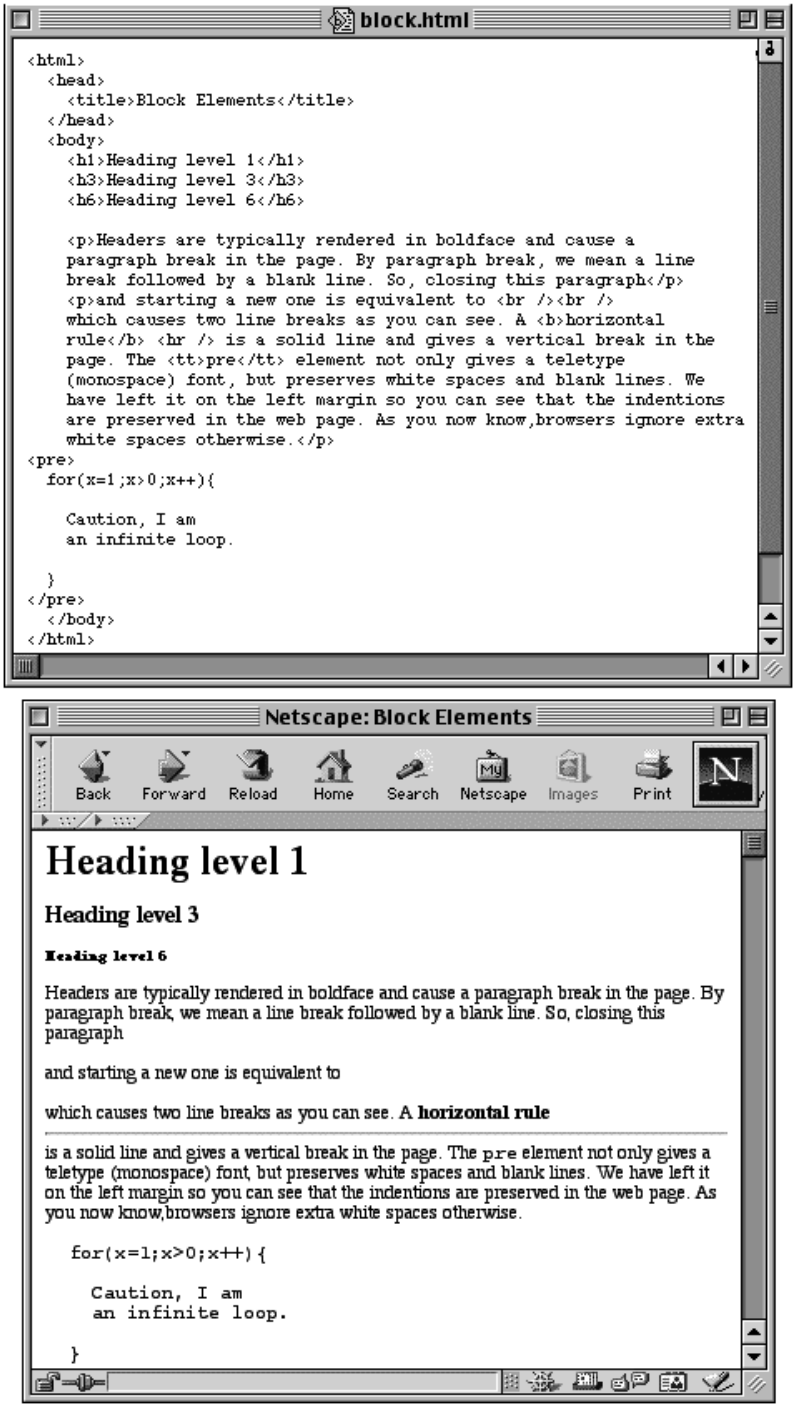


FIGURE A.4 A simple Web page using block elements.

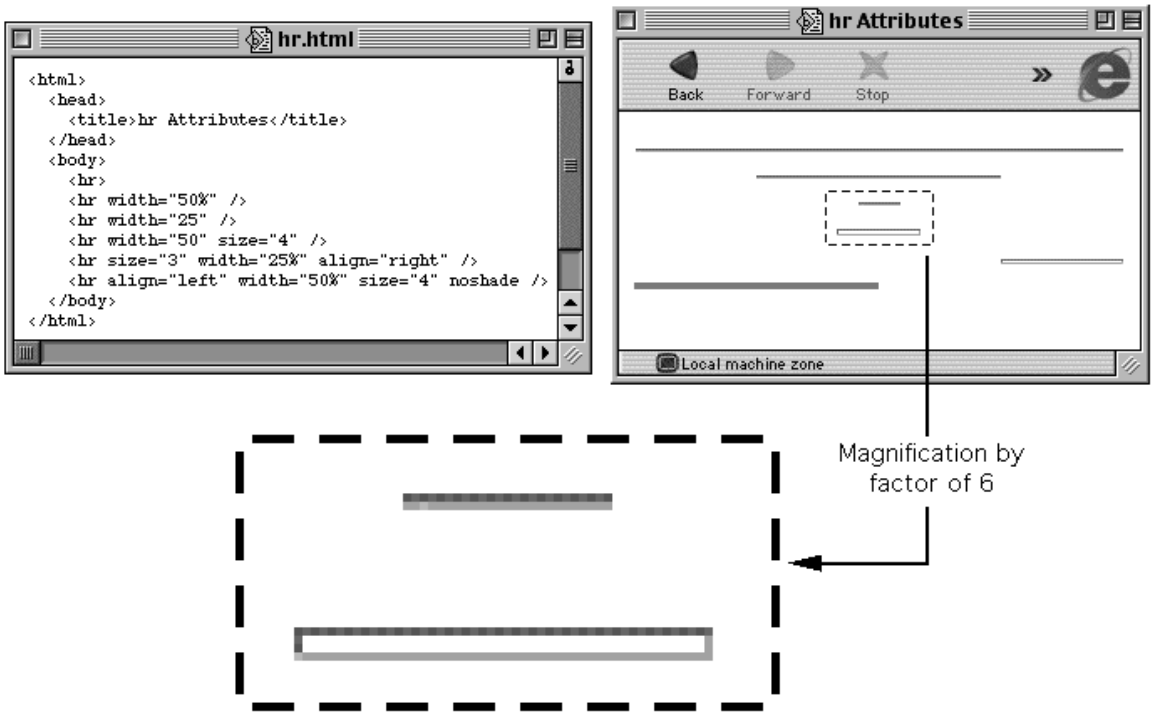


FIGURE A.5 Using attributes in the `hr` element.

how high. In this case, a browser simply uses its *default settings*. Typically, browsers default to drawing a horizontal rule as a block 100% of the width of the browser window and two *pixels*[2] high. Figure A.5 gives an illustration for the discussion to follow.

The first `hr` uses the browser's default settings. Each of the other `hr` instances use attributes to override the default behavior. An HTML **attribute** is a *name*`="`*value*`"` pair that provides an extra instruction to tell the browser how to mark up the element. The second `hr` uses the `width` attribute to set the horizontal rule to cover 50% of the width of the browser window. The third sets it at an absolute 25-pixel width. Resizing the browser window changes the percentage setting proportionally, whereas the pixel setting remains constant. (You can see this by pulling up the source code from the Web site with your browser.) The fourth uses two attributes. Here `size` specifies the height of the `hr` to be 4 pixels. You can see that the first three horizontal rules specify no size, so the default of 2 pixels is used. The magnification of part of the diagram attests to this, and also shows how the shading effects are achieved using pixels. The last two instances use the `align` attribute to override the default of centering the `hr` in the browser window. Finally, the last instance uses the `noshade` attribute to override the shading effect. Contrast the `noshade` rule with the other rule that is also of size 4.

[2]Pixel stands for picture element. (We are guessing it just sounded cooler with an *x* to whoever made up the abbreviation.) These are the little colored dots that collectively make up what you see on the computer screen.

Here are some important notes regarding HTML attributes. All but the second point are evident in Figure A.5.

- Attributes must be placed in a tag, following a space, after the tag name.
- All attributes must be placed in the start tag.
- Do not put spaces on either side of the = sign.
- When multiple attributes are used in an element, the order in which they appear does not matter. Moreover, multiple attributes need only be separated from each other with single spaces.
- The quotes around the attribute value make the value well defined as a string.
- A given attribute may accept more than one value type (percentages or pixels, for example).
- Some attributes don't take values (`noshade`, for example). Such attributes function like true or false (on or off) Boolean quantities. Note that to be valid in XHTML syntax, the attribute would be written `noshade="noshade"`.
- If an attribute is not specified, the browser reverts to a default setting. In many cases the default is dependent on the browser type or version.

Our choice to use `hr` to demonstrate attributes is based upon the fact that it uses multiple attributes that accept many of the different value types you will need to understand. Just in terms of the usefulness of `hr`, the foregoing discussion would be overkill. (Man, I love that `hr`!) With that in mind, we summarize its attributes in Table A.3 in much the same way we summarize attributes for other elements we will encounter. Here you begin to understand what a full summary for the capabilities of an HTML element entails.

We conclude this section by summarizing the two `block` elements we have introduced that have standard attributes (Table A.4) and by giving an example in Figure A.6 that puts some of them to good (that's a judgment call) use. What about the inline elements and other block elements? Well, they just don't have standard attributes. There is really no "fine tuning" necessary in those cases. CSS offers a better means to control font properties more explicitly.

TABLE A.3 The `hr` and Its Attributes

`<hr />` Horizontal rule (block element)

ATTRIBUTE	POSSIBLE VALUES	DEFAULT
`width`	%, Pixels	`100%`
`size` (height of `hr`)	Pixels	`2`
`align`	`left, center, right`	`center`
`noshade`		Shaded markup

TABLE A.4 Block Elements with Attributes

`<hx >...</hx>` Headings: $x = 1, 2, 3, 4, 5, 6$ (block element)

ATTRIBUTE	POSSIBLE VALUES	DEFAULT
`align`	`left, center, right`	`left`

`<p >...</p>` Paragraph block element

ATTRIBUTE	POSSIBLE VALUES	DEFAULT
`align`	`left, center, right`	`left`

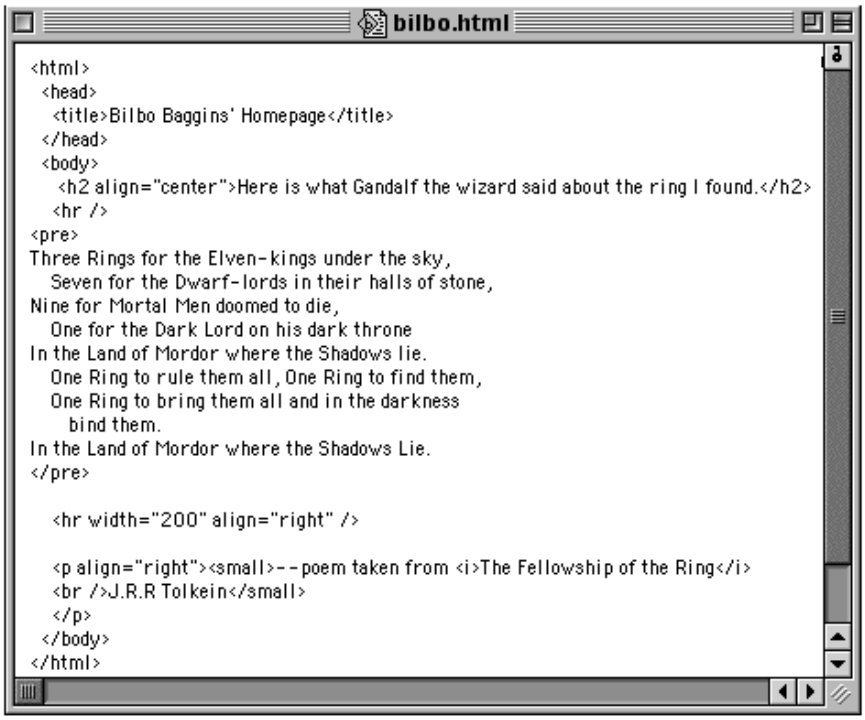


FIGURE A.6 Using attributes in various elements.

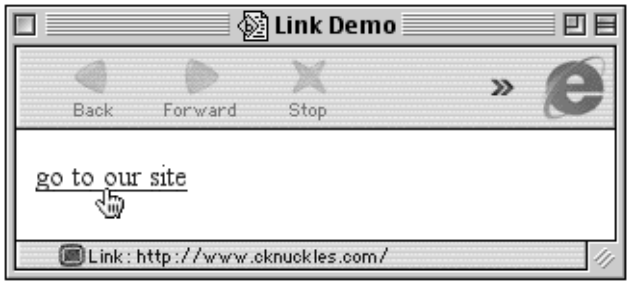
A.5 HYPERLINKS

Hyperlinks, or simply *links*, are created with the `a` (**anchor**) element, which is an inline element. Table A.5 below summarizes the attributes of the anchor element that we feature in this section.

The **hypertext reference (`href`)** attribute provides reference to another hypertext document. For an example, we provide the HTML code that creates a link to our Web site.

```
<a href="http://www.cknuckles.com/">go to our site</a>
```

When this element is rendered by a browser, you see only the content of the anchor element, which the browser underlines to signify that the text represents a link. When you pass the mouse over the link (but don't click it), the URL to which it points is displayed in the *status bar* at the bottom of the browser window, as shown:



The value of the `href` attribute used in the example we have just given is called an **absolute URL**. This type of URL specifies absolutely the location of a resource, in this case the default file of our Web site. Of course, an absolute URL can contain more directory path information, as in

```
<a href="http://www.cknuckles.com/somepath/somefile.html">go to some
file deep in our Web site</a>
```

Recall Figure 1.7, which featured absolute URLs, although we did not call them that there.

When you create a link to another document in your own Web site, using a **relative URL** is the way to go. A relative URL specifies the file path to a document relative to the document in which the link appears. To illustrate this, we return to the example of

TABLE A.5 The Hyperlink and Its Attributes

`<a >...</a>` Anchor (inline element)

ATTRIBUTE	POSSIBLE VALUES	DEFAULT
`href`	URL (absolute or relative), URL fragment	
`name`	Fragment identifier	

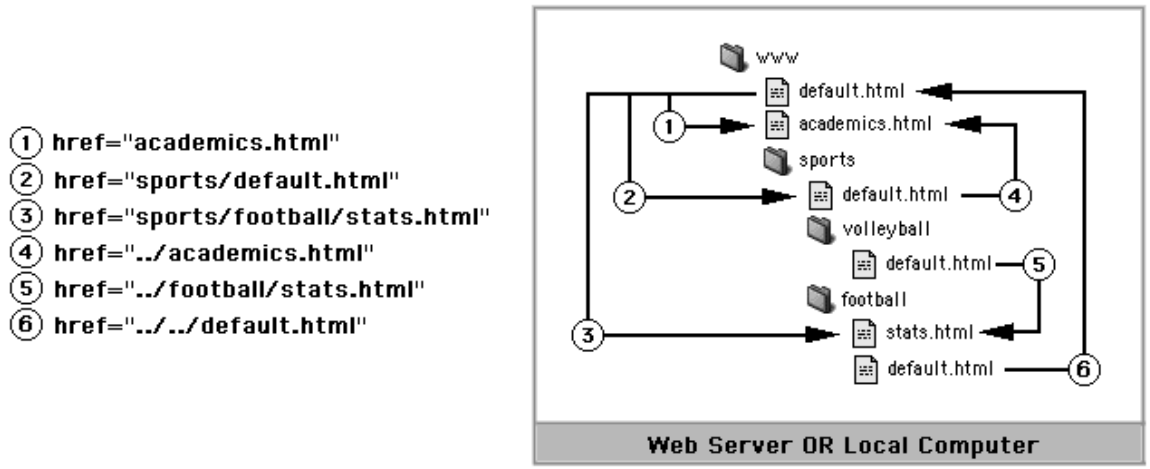


FIGURE A.7 Internal links using relative URLs.

UWEB's Web site used in Section 1.5, but this time, we create some internal links for the site using relative URLs rather than pointing to the files using absolute URLs.

Figure A.7 provides the illustration. The first three URLs in the list are for links in UWEB's home page, `default.html`, which is in the university's root directory. The first URL demonstrates that only the name of the file is necessary if the link points to a file in the same directory. Here the actual link would be produced by putting

```
<a href="academics.html">go to academics page</a>
```

in UWEB's home page. Similarly, the rest of the `href` attributes shown in Figure A.7 would be contained in similar anchor elements, but we won't write the rest of them out in full form.

As the second URL shows, targeting a file deeper in the directory structure requires a relative path. From the frame of reference of UWEB's home page where the link appears, that URL says "look inside the `sports` directory and grab the default file." Incidentally, the second URL is equivalent to `href="sports/"` because the default file is targeted. The third URL targets the `stats` page, two directories below UWEB's home page.

The fourth URL appears in a link in the `sports` home page. The construct `../` says to back up one directory. In words, the fourth URL says, "Back up outside of the current directory and find the `academics` page." The fifth URL is a little more involved, because the link in the `volleyball` home page targets a file in a folder that is "parallel" in the directory structure to the volleyball folder. Here the URL says, "Back up one directory (into the `sports` folder), then go into the `football` folder and find the `stats` page." Finally, the sixth URL says to back up one directory (into the `sports` folder), then back up one more (into the root folder) and find the default page. Note that `href="../../"` is equivalent.

Here are some important notes for links:

- There should be ample description in a Web page as to where a link points. This can be accomplished through the anchor content that appears as the underlined link in the page or by the text surrounding the link. It is poor practice to leave a link's target ambiguous to the surfer.

- As your Web site gets larger, you will want to organize it with some directory structure, much as UWEB has done. It is *important* that the directory structure you create as you develop the site locally be preserved when you transfer it to the Web server. Otherwise, your links will fail. You can see that in Figure A.7. If UWEB's webmaster updates the local copy of the `sports` page but accidentally transfers it with his FTP client into the wrong folder on the Web server, the relative URLs will no longer be accurate, and the links will fail. Worse yet, if it is accidentally transferred into the root directory for example, it will overwrite UWEB's home page because both are named `default.html`. No biggie. The local copy of the Web site is still intact, and the webmaster can ftp the files over into the correct directories once the mistake has been discovered (hopefully quickly).
- Use lowercase file and directory names with no spaces between letters. File and directory names on most Web servers are *case sensitive*. That means that if the server looks for `somepage.html`, it will not find it if it's named `Somepage.html`. Even if the links with mismatched cases work in your local copy of the site, they will likely fail when the pages are HTTP requested from the Web server. When working locally, your browser and the `file` protocol are laid back about case, but HTTP Web servers are generally uptight. If you stick to lowercase names as a general rule, you will have fewer broken links in the long run.
- With the preceding point in mind, directory paths and file names of relative URLs must match letter for letter and case for case with the paths and file names within the Web site.

These bulleted notes emphasize the importance of testing your pages locally to get them right and then testing them again once you ftp them to the server. Although conceptually simple, links can be a bit tricky. Even the experienced Web programmer occasionally makes an FTP mistake or mistypes a file name. We do! But we usually catch the mistake when we test the site. How will you fare?

NOTE

You could make all of the links depicted in Figure A.7 using the absolute URLs as shown in Figure 1.7 of Section 1.5. Besides more typing, there is a major drawback to that approach. Remember, UWEB's webmaster designed the site on his personal office computer. Links with

```
href="http://www.uweb.edu/etc."
```

will obviously not work for local files. The site would not be portable. Interestingly, you can use absolute URLs with the `file` protocol. For example,

```
href="C:\www\sports\default.html"
```

would adequately link to the local copy of the sports home page, but it would be useless for HTTP requests once put on the server. Again, if internal links are not relative, the site is not portable.

A last comment on `file` URLs is important. The \ (backslash) is a Microsoft Windows directory path convention. *If you use a Windows machine, do not use \ when you write URLs*. PC browsers and servers will deal with / in a Web context. No other platform in the world (that we know of) will deal with \ in a directory path.

One of the two attributes, `href` or `name`, is mandatory in an anchor element. If you have ever clicked a link that took you to a different location in the same Web page rather than loading a different page, you have seen a *named anchor* in action. A typical situation that calls for named anchors is when a Web page is fairly long, say with several sections (such as if we put this appendix in a Web page). You would then put a list of links at the top, one pointing to each section, and a link at each section pointing back to the top. Before reading on, you should go to the Web site and play with the named-anchor demo. While the following discussion contains enough detail by itself, it would be helpful first to see in action the page upon which the example is based.

In order to set up the list of links to sections, you mark each section with a named anchor like the following:

```
<a name="section2"></a>
```

which obviously would be for Section 2. The value that the `name` attribute specifies is called a **fragment identifier**. To target the named anchor, you put the following link (`href` anchor) at the top of the page:

```
<a href="#section2">Go to Section 2</a>
```

When the "Go to Section 2" link is clicked, the named anchor is instantaneously brought to the top of the browser window. Here the URL **fragment**, `#section2`, causes the link to target the named anchor identified as `section2`. Note that the `#` sign is mandatory to signify that the URL is merely a fragment.

Similarly, each of the other sections is marked by a named anchor with an appropriately chosen identifier, and links at the top of the page point to them. However, for the links that point back to the top of the document, one named anchor, placed at the beginning of the `body` section,

```
<a name="top"></a>
```

suffices. The "Back to top" links then all target the one named anchor.

You can also target a named anchor with a link in a different Web page. For example, suppose you have an external table of contents in a separate Web page. If all the named anchors we just discussed are in a file named `lesson2.html`, you could then target Section 5 from the external table of contents,

```
<a href="lesson2.html#section5">Go to Section 5</a>
```

by appending the fragment onto the relative URL pointing to Lesson 2. Clicking the link causes `lesson2.html` to load into the browser, but with Section 5 at the top of the window. In this example, `lesson2.html` is in the same directory as the file with the table of contents, so the file name suffices. In general, you can append a fragment onto a relative URL with a longer directory path or an absolute URL. Note that with no content given, a named anchor is invisible in the rendered Web page. There are a couple of subtleties involved, and the demo on the Web site addresses them.

A.6 GRAPHICS

Browsers uniformly support two types of images that can be embedded in Web pages. The majority of graphics on the Web are of type **GIF (Graphics Interchange Format)**. GIF file names are appended with the `.gif` extension and the format has the following features:

- GIF supports a *color palette* of a maximum of 256 colors, but the palette can be reduced to only 2 (such as black and white), 4, 8, 16, 32, 64, or 128 colors.
- GIF is flexible. Images can have transparent backgrounds or be animated. Examples are provided on the Web site.

The other graphic type in wide use is **JPEG (Joint Photographic Experts Group).** JPEG file names are appended with the `.jpeg` or `.jpg` extension and the format has the following features:

- JPEG uses the full RGB spectrum, supporting over 16 million colors.
- JPEG uses a superior (to that of GIF) compression algorithm to create smaller file sizes.

When small graphics that only require a few colors suffice, GIF is usually used. With small color palettes, they are easier to edit. Moreover, their flexibility is ideal for creating fancy Web pages. If you see something moving in a Web page, it's likely an animated GIF. Even though GIF's compression algorithm is less sophisticated than that of JPEG, it's small color palettes tend to make small graphic files when the graphics are kept simple. Those files are typically 2K to 4K in size.

When one needs a digital likeness of a real-life photograph, JPEG is the way to go. The 16 million colors create a much sharper image, while the superior compression algorithm does a decent job of keeping the file size relatively small. For an example, pull up `www.cknuckles.com/beest.jpg` and `www.cknuckles.com/beest.gif` in separate browser windows and compare them. You should be able to see that the GIF image gets somewhat "splotchy" in places where many of the colors were approximated with the 256-color palette. That file takes up 68K of computer memory. In contrast, the JPEG version is somewhat sharper given all the colors it uses, and JPEG compression has limited the file to 20K of computer memory.

So how do you get images to put in a Web page? If you want to put a picture of your dog (or wildebeest) on your homepage, you can upload the file to your computer from a digital camera or video recorder. Or you can use a scanner to create a digital likeness of a conventional photograph. If you want some GIFs to add some spice to your Web page, you can simply grab them from other Web pages. If you find one to your liking, you can easily acquire it:

- In Windows, right-click on the image and choose to Save Image As... from the pop-up menu.
- Using a Mac, you can simply drag the image from off the Web page onto the desktop or into a folder. Also, doing a control-click on the image provides a pop-up menu similar to that on Windows.

Of course, one should avoid acquiring copyrighted images in this manner. There are numerous free image libraries on the Web that contain virtually any GIF you might want. Links to some of these are provided on the Web site.

TABLE A.6 Image Element

`<img />` Image (inline element)

ATTRIBUTE	POSSIBLE VALUES	DEFAULT
`align`	`top`, `middle`, `bottom`, `left`, `right`	`bottom`
`alt` (alternate description)	Any text string	
`border`	Pixels	
`height`	Pixels, percent	Actual height of image
`hspace` (horizontal space)	Pixels	Browser-dependent
`src` (source; required attribute)	URL (relative or absolute)	
`vspace` (vertical space)	Pixels	Browser-dependent
`width`	Pixels, percent	Actual width of image

To mark up a graphic in a Web page, you use the `img` (image) element. It is an inline element, and it is summarized in Table A.6. In practice, marking up images is straight forward. Figure A.8 shows the source code for an HTML file that explores the `img` element, and Figure A.9 shows its rendition.

You can add functionality to an image in a Web page by making it active as a link. You simply put the `img` element as content of an `href` anchor.

```
<a href="dude/links/to/file.html"><img src="dude.gif" border="0" /></a>
```

Here are some things to note about including images in a Web page:

- The URL value of `src` gives the location of the source file for the image. Typically, the source file is sitting somewhere in your Web site and you use a relative URL. The same rules apply that we outlined in Section A.5 for relative URLs as values of `href`.
- When you make an image active as a link, the `href` anchor may give it a colored border, much as it underlines textual links. This is unsightly. You likely will want to suppress the border by using the `border` attribute in the image element, as we did in the image link example just above.
- Even if you don't intend to resize an image, you should include the `height` and `width` attributes using the image's actual dimensions. That way, when the browser reads the HTML file, it knows how much space to allocate for the images. It can draw empty boxes for the images and start rendering the page before all of the images arrive. As the images arrive, it slaps them into the preallocated boxes. Otherwise, the browser would have to wait for all of the images before starting to render the page. Always using `height` and `width`, even if you don't resize an image, increases the efficiency of your pages in general. An easy way to determine the dimensions of an image is to load it into a browser by itself (not embedded in a Web page). Its height and width the should be displayed at the top of the browser window. In Windows you can right-click on the image when it is displayed by itself in Internet Explorer, and select **Properties...** from the pop-up menu.

imagedemo.html

```
<html>
  <head>
    <title>Image Attributes</title>
  </head>
  <body>

    The top, middle, and bottom values align images inline with text. This one
    <img src="dude.gif" width="32" height="32" align="top" /> is top-aligned and this one
    <img src="dude.gif" width="32" height="32" align="middle" /> middle-aligned. Since we
     included no <tt>align</tt> attribute in this one
    <img src="dude.gif" width="32" height="32"> you can see that bottom is the default alignment.<br />
    <img src="dude.gif" width="30" height="30" align="left" /> It is often more useful to have text
    flow around an image, rather than putting the image inline with text.
    <img src="dude.gif" width="32" height="32" align="right" hspace="10" vspace="5" />
    After all, that's how newspapers  do it! That's precisely the effect caused by left and right
    alignment. It certainly does create a better effect. But, notice how the text is right up against
    the left aligned image. If you don't think it's cool to have text right up against the image,
    some <tt>hspace</tt> and <tt>vspace</tt> around the image is in order. These attributes simply
    add horizontal and vertical padding around the image. Please note that we're starting to have
    to include a substantial amount of babble here to get the text to flow all the way around the
    image <br>  The border attribute can be used to give an image a thin
    <img src="dude.gif" width="32" height="32" border="2" align="middle" /> border or a thick
    <img src="dude.gif" width="32" height="32" border="10" align="middle" /> border.<br />
    <img src="Dude.gif" width="100" height="40" align="left" alt="picture of a dude" /> The
    <tt>alt</tt> attribute gives a textual description of the image. This was important in the
    early days when computers and internet connections were sloooow, and some people would set
    their browsers to "text only" so they could surf faster. Now, with cable and DSL connections,
    pages full of images are no problem. Even with 56K phone modems, most people tolerate the
    extra overhead incurred by graphics. However, the importance of <tt>alt</tt> is emerging again
    since images are a drag on handheld internet devices. Note that we did not disable images on our
    browser to show the alternate description. We simply supplied a bad URL so that the browser
    couldn't find the image file. (When the browser asked for the image, the server software sent back
    a packet indicating it could not find the image file. The browser resorted to the alternative.) <br />
    Finally, you can resize an image
    <img src="dude.gif" width="16" height="16" /> proportionally or
    <img src="dude.gif" width="32" height="16" /> disproportionately.

  </body>
</html>
```

FIGURE A.8 Using attributes of the image element.

- Recall that, whereas HTML files need to be transferred to a Web server as plain text, *it is imperative that you ftp image files to the Web server as binary data*. If you transfer them as text or a proprietary format, they probably won't show up in the Web page.

A.7 THE body AND head ELEMENTS

In this section we focus primarily on the `body` element, and include a note at the end about the `head` element. The `body` element has several attributes that set properties for the whole body of the Web page, summarized in Table A.7. Keep in mind that you put these attributes in the start tag of the `body` element that is part of the skeleton document. You don't add a new tag to the document to use these.

All but one of these attributes take values that specify colors. You can use *named colors* such as `red`, `green`, and `blue`. There are two problems with using named colors, however. First, many named colors have weird names ("light goldenrod yellow" or "papaya whip", for example), and browsers support somewhat different sets of named colors. Second, there are only a relatively few colors with names.

TABLE A.7 The `body` Element

`<body>...</body>` Body of Web page (block element)

ATTRIBUTE	POSSIBLE VALUES	DEFAULT
`alink` (active link)	`#`*hexcolor*, named color	Browser-dependent (often red)
`background` (background image)	URL (relative or absolute)	
`bgcolor` (background color)	`#`*hexcolor*, named color	Browser-dependent (often white or light gray)
`link` (unvisited link)	`#`*hexcolor*, named color	Browser-dependent (often blue)
`text`	`#`*hexcolor*, named color	`black` (`#000000`)
`vlink` (visited link)	`#`*hexcolor*, named color	Browser-dependent (often light purple)

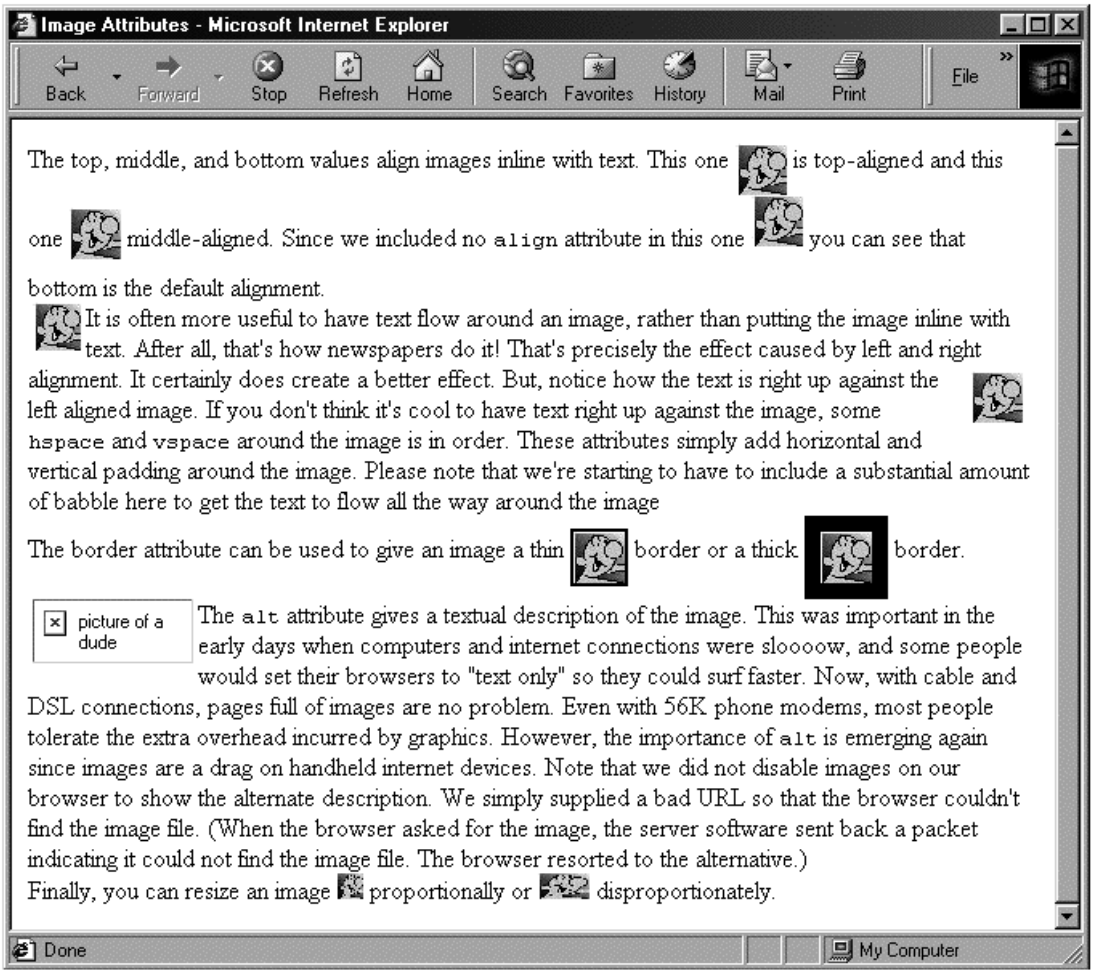


FIGURE A.9 The document of Figure A.8 rendered in a browser.

In contrast, **hexadecimal colors** (hexcolors) give you over 16 million choices (the same color spectrum used by JPEG). Hex colors have three components: one each for red, green, and blue. Thus they are sometimes called **RGB colors**. Each hex color is of the form **#***RGB* where each of the three color components is a two-digit hexadecimal number. For example, `#000000` is black, `#FFFFFF` is white, `#FF0000` is red, `#00FF00` is green, and `#0000FF` is blue. Hex colors must be preceded by a `#` sign. Moreover, links to some color-picking utilities are given on the Web site.

To illustrate the attributes of the `body` element, we stick to some simple named colors, and the few hex colors just listed.

```
<body bgcolor="red" text="white">
```

and

```
<body bgcolor="#FF0000" text="#FFFFFF">
```

are equivalent, and both produce a Web page where the entire body is colored red and the default text color is white. That's really all there is to it.

The following ensures that the Web page has a white background, while setting colors for the lines drawn under `href` links in the page. Text will remain the default color of black.

```
<body bgcolor="#FFFFFF" link="red" vlink="green">
```

Here, links that have yet to be visited will be underlined in red, and visited links will be underlined in green. (Browsers maintain a recent history list to keep track of such things.) Controlling link colors might seem a bit picky, but in pages with carefully chosen color schemes the blue and purple defaults for link colors can look gnarly. Setting the `alink` attribute is less important, because you see the link activated only the moment you click it. (If you right-click a link to get the pop-up menu, it should stay activated while the menu is open.) You can experiment with `alink` for yourself.

The value of the `background` attribute is a URL that points to an image file to be used as the background of a Web page. An image used for a background is often a GIF that provides a subtle texture for the Web page.

Such an image is *tiled* to create a solid background in much the same way as tiles are laid for a kitchen floor. If the image is cleverly made, the tiling will appear seamless, just as a kitchen floor appears to have a unified pattern even though it is a conglomerate of small tiles. Tiling is advantageous because only a small image file need be transferred to the client with the Web page. A graphic whose dimensions would actually fill up the background of the whole page would generally have much too large of a file.

If you specify both a `bgcolor` and a `background` image in the same page, the image will obscure the `bgcolor`. However, if the image has any transparent regions, the `bgcolor` will show through. You can find tons of images suitable for backgrounds in the graphic libraries we have referenced on the Web site. Some are nice and subtle, while others will make you cringe.

NOTE

Only seven elements are allowed to be placed as content of the `head` element: `base`, `link`, `meta`, `object`, `script`, `style`, and `title`. We have used the `title` element, will use the `link` and `style` elements in the section on CSS, and will use the `script` element when we utilize JavaScript. That leaves `base` and `object`, which are rarely used, and finally `meta`. The `meta` element is for information about the document: meta-information. The most common use of `meta` is for the benefit of the search engines that crawl the Web indexing pages. Some of them look for a comma-delimited list of keywords:

```
<meta name="keywords" content="key,words,that,characterize,page,..." />
```

Others look for a short description:

```
<meta name="description" content="page about wildebeest and..." />
```

Others simply read the contents of your page looking for multiple occurrences of words.

A.8 FRAMES

HTML **frames** provide a way to show more than one Web page simultaneously in the same browser window. It is best to proceed by example. Figure A.10 shows two distinct Web pages displayed in a browser window. On the right are the source files for two different Web pages. You can see that one has been stuffed into the top frame of the browser window and the other into the bottom. The source file for the page that creates the actual *frameset* is shown below the browser window. You can tell from the page title at the top of the browser that it is the page whose URL the browser called.

That document has no `body` section. Rather, the `frameset` element has taken its place. The `frameset` element has called for a frameset with two rows, the first being 75 pixels tall and the second filling up the rest of the browser window. The first `frame` element has specified that the `src` (source) for the first frame should be the file `navpage.html`, and the second `frame` element has specified `placeholder.html`. Those two pages are rendered in their respective frames in the browser window.

If you click a link in the top "navigation frame," the Web page to which that link points is loaded into the bottom "display frame", replacing the initial page contained there. (You can pull up the source for Figure A.10 on the Web site to get a feel for how it works.) You can see in the frameset document that the bottom frame was assigned a name using the `name` attribute. Each link in the top frame simply specifies `target="displayframe"` as the destination for the link. That causes the documents to be loaded into `displayframe`

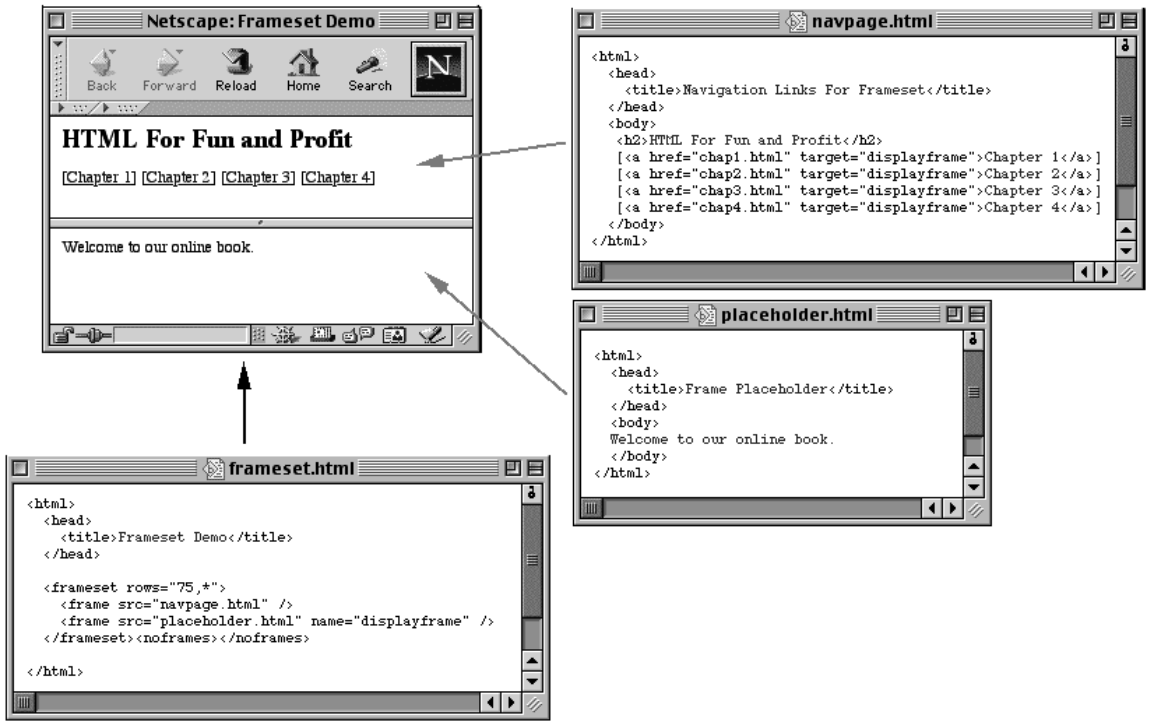


FIGURE A.10 A simple frames page.

rather than the entire browser window. Now that you have a feel for how framed pages work, we summarize the `frameset` and `frame` elements in Table A.8.

The margin, border, resizing, and scrolling attributes are self-explanatory. You would be better off experimenting with them than reading anything further we might say about them. However, the `rows` and `cols` attributes require some elaboration. We will use `cols` for explanation, because we used rows in Figure A.10. However, the same applies to both. It is clear that

```
<frameset cols="30%,20%,50%">
```

creates three columns of various widths relative to the size of the browser window. For fixed widths,

```
<frameset cols="200,300,*">
```

creates two columns of widths 200 and 300 pixels, respectively, while the third column would cover the rest of the browser window. The setting

```
<frameset cols="30%,*,*">
```

TABLE A.8 Elements Used in Frame Pages

`<frameset >...</frameset>` Replaces the `body` element (block element)

ATTRIBUTE	POSSIBLE VALUES	DEFAULT
`cols` (width of columns of frameset)	Pixels, percent, `*`	
`rows` (heights of rows of frameset)	Pixels, percent, `*`	
`border`	Pixels	Browser-dependent (about 5)
`bordercolor`[a]	`#`*hexcolor*, named color	Browser-dependent (usually some shade of gray)
`frameborder`[a]	`yes` (3-dimensional borders), `no` (plain "flat" borders)	`yes`

`<frame />` Used only inside `frameset` element

ATTRIBUTE	POSSIBLE VALUES	DEFAULT
`marginheight` (vertical padding between frame border and frame content)	Pixels	Browser-dependent (about 10)
`marginwidth` (horizontal padding between frame border and frame content)	Pixels	Browser-dependent (about 10)
`name` (background image)	Text string	
`noresize`	`noresize="noresize"`	User can resize frames in browser window by dragging borders
`scrolling` (does a frame receive its own scrollbars?)	`yes` (always, even if not needed), `no` (never, even if needed), `auto` (if needed)	`auto`
`src` (source)	URL (relative or absolute)	

[a] You can use `bordercolor` and `frameborder` in a particular `frame` element as well. In that case the border instructions from `frameset` are overridden. Browsers deal with that unpredictably, however.

would set the first column at 30 percent and divide the last two columns evenly in the remaining space. The setting

```
<frameset cols="100,*,2*">
```

would set the first column at 100 pixels, the second at one-third of the remaining space, and the third at two-thirds of the remaining space. (`*` says "one portion of the remaining space," and `2*` says "two portions".) Finally,

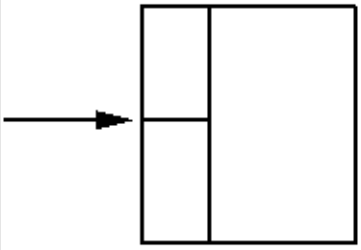
```
<frameset cols="15%,2*,3*,100">
```

```
<frameset cols="30%,*">

  <frameset rows="*,*">
    <frame src="somepage.html" />
    <frame src="somepage.html" />
  </frameset>

  <frame src="somepage.html" />

</frameset>
```



```
<frameset rows="75%,*">

  <frame src="somepage.html" />

  <frameset cols="25%,*">
    <frame src="somepage.html" />
    <frame src="somepage.html" />
  </frameset>

</frameset>
```

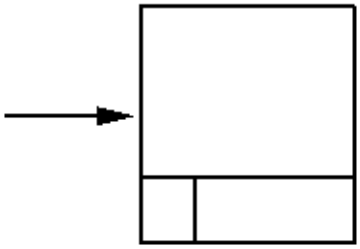


FIGURE A.11 Nesting frames.

sets the first column at 15 percent of the browser window, the last column gets the last 100 pixels on the right side, the second gets two-fifths of what's left over in the middle, and the third gets three-fifths, although we can't imagine why anyone would have a practical reason to do such a thing.

To nest frames, simply use a new `frameset` where you would normally put a `frame`. The examples in Figure A.11 require only a brief explanation. The first one sets up two columns. Then the first column gets another `frameset` element rather than a `frame`. That's the two-row frameset you see in the first column. The second column gets a `frame` as usual.

The second example sets up a two-row `frameset` in which the first row gets a `frame`. What would be the second `frame` row gets a two-column `frameset`.

We conclude this section by further discussing the use of links in framed documents and then noting a few things about frames in general. We saw in Figure A.10 that by naming a frame, you can enable links to `target` that frame. If you specify no `target` in a link, the default is to load the Web page to which the link points into the same frame as the page containing the link. For example, in Figure A.10, if we had forgotten to specify a `target` in the link for Chapter 2, the Web page for Chapter 2 would get loaded into the top frame, replacing the navigation page.

With that in mind, we summarize targeting options in Table A.9. We won't enter into further discussion about targeting, because the table is self-explanatory. However, do keep in mind that the underscore is necessary for the `_self`, `_parent`, and `_top` special `target` values. You can get a hands-on feel for how targeting works by visiting the targeting demo on the Web site.

Here are some notes regarding frames.

- Frames are ideal when a lot of information needs to be swapped in and out quickly, as in a slide show-style presentation, for example. In that case you have a "fixed"

TABLE A.9 Options for Targeting Links in Frames Pages

OPTION	RESULT
No `target` attribute supplied	Same as `"self"`
`target="`*framename*`"`	The page is loaded into the specified frame.
`target="_self"`	This is the default. The page is loaded into the frame in which the link was clicked.
`target="_parent"`	The page is loaded into the window or frame containing the parent `frameset` page.[a]
`target="_top"`	The page is loaded into the main browser window.[a]
`target="`*non-existent-framename* `"`	The page is loaded into a new window that pops up. (Misspelling a frame name usually makes for an unwanted pop-up window.)

[a] For almost any case, `_parent` and `_top` accomplish the same thing. Their difference is subtle. The target demo on the Web site explores this subtlety.

navigation bar that stays put in one frame, which swaps pages (or images) in and out of other frames. A fine example putting frames to highly functional use by can be found at

```
http://www.bnrmetal.com/
```

This site uses frames to organize a lot of information to great functionality.

- Tables provide for much more control over page layout. Indeed, nearly every major commercial site features a tabular layout. These sites usually feature a "pseudo-fixed" navigational bar. Even though the nav bar appears in the same place on several pages, it's simply part of a page template on which all the pages are based, and it gets reloaded every time.
- Besides layout deficiencies, frames have the following two disadvantages. You can't bookmark pages deep within a frames site. (Go to a band page at the site mentioned in the first item on this list, and try to bookmark it.) Also, many browsers have difficulty sending content to a printer from pages rendered within frames. That is a deterrent for many commercial sites as well.
- It is kind of cool to get a new window using the nonexistent-frame trick. However, there is a much better way to open new windows using JavaScript.
- Use your frames to load your own pages. It is generally uncool to load other people's pages into your frames. In other words, if you link to someone's page, their domain name should show in the address window. For example, you could make a page with only one frame whose source is Amazon.com. Then your domain name could effectively be for their site. (There is a way to protect pages against that.)
- There is a `noframes` element that can be used to provide alternative content for browsers or devices that can't handle frames. Just put `<noframes> <body> alternate content </body> </noframes>` right after the first `frameset` tag in your top frameset. The `body` of `noframes` provides an alternative body for the page if frames can't be used.

A.9 DEPRECATED ELEMENTS AND ATTRIBUTES

To **deprecate** means to downgrade or devalue. In the context of the W3C HTML and XHTML specifications, a deprecated element or attribute is one that has been designated potentially to be dropped from the specifications at some point in the future, although at what point in the future is not clear. The HTML elements and attributes listed in this section have been on deprecated status since the 1997 issue of the HTML 4.0 standard.

Deprecated Elements

We now summarize the most notorious deprecated elements in Tables A.10 and A.11. In particular, if you look at much HTML code currently on the Web, you will see `font` and `center` used quite a lot. We do not use these elements in this text (other than `font` in Section 2.1), but it pays to know which elements are deprecated and why.

With the exception of `applet`, all of these elements can easily be replaced using CSS. Indeed, that is why they are on deprecated status. The goal is to "keep HTML small," leaving text formatting issues for CSS. It is also worthwhile to note that, while they are not officially on deprecated status, the elements associated with HTML frames will likely be targeted for deprecation in the future. However, as noted in Section 2.5, deprecated elements, and especially frames, will not disappear from the Web for many years.

TABLE A.10 The Deprecated `font` Element

`<font >...</font>` Inline element

ATTRIBUTE	POSSIBLE VALUES	DEFAULT
`color`	`#`*hexcolor*, named color	`#000000` (black)
`face`	Text string (font name)	Browser default font (often Times New Roman)
`size`	Absolute size (1, 2, 3, 4, 5, 6, 7) relative size (±1, 2, 3, …)	Browser default (typically 3)

TABLE A.11 Other Noteworthy Deprecated Elements

`<basefont />`	Takes the same attributes as the `font` element. It is usually placed at the top of the `body` section and sets font properties for the whole page. The font element overrides the default font properties set by `basefont`.
`<center>...</center>`	Takes no attributes and centers its contents, whether text or a block, on the page.
`<applet>...</applet>`	Used to embed Java applets in Web pages. Deprecated in favor of the `object` element, which can embed several types of multimedia objects in a page.
`<u>...</u>`	Causes its contents to be rendered as underlined text.

Deprecated Attributes

Stylistic properties of most HTML elements can be completely controlled using CSS. Thus, a surprising number of very common attributes are also on deprecated status. The following list gives a reasonably comprehensive overview of the deprecated HTML attributes:

- All atributes of deprecated elements such as `font` and `applet` are themselves deprecated.
- The `align` attribute, which is used in numerous elements such as `p`, `hr`, `img`, `table`, `td`, is deprecated.
- Virtually all attributes of the `body` element (such as `background`, `bgcolor`, `link`, and `vlink`) are deprecated.
- Most formatting and positioning attributes of the `img` element (`border`, `hspace`, `vspace`, and, of course, `align`) are deprecated. The `src`, `height`, and `width` attributes are *not* deprecated. Of course, `src` is necessary to specify the source file for an image. Presumably, the `height` and `width` attributes are not deprecated in favor of CSS because Web pages can be rendered more quickly when image dimensions are specified (even if the image is not to be resized) in the HTML code. See the second to last bullet note at the end of Section A.6.
- The formatting attributes of HTML lists (`start`, `type`, `compact`) are deprecated.
- Most table-formatting attributes (`bgcolor`, `background`, `valign`, `align`, `height`, `width`) are deprecated. However, essential table-structuring attributes such as `colspan`, `rowspan`, `cellpadding`, and `cellspacing` are *not* deprecated.

This is an alarming amount of deprecation of attributes to which we have grown accustomed as being an essential part of HTML. However, limiting the amount of *attribute-based formatting* is another part of the objective to "keep HTML small." Source-providing atributes (such as `src` and `href`) and structure-providing attributes (such as `colspan` and `rowspan`) can't easily be eliminated, but the color-providing and alignment-type attributes can be replaced by setting corresponding CSS properties.

Of course, keeping the language small by eliminating HTML attributes (and elements) means that the missing complexity is absorbed by CSS. That does serve to meet the W3C objective of keeping style formatting separate from content and structure formatting. This development is also good for the Web because it tends to result in HTML files that take up less memory. For example, if 10 table rows each have the same formatting properties, it is a waste to include 10 identical sets of attributes in each of the `tr` elements. Rather, one style class can be created and applied to each table row. But, on the negative side, use of CSS increases the knowledge base required of Web programmers and designers.

NOTE

We have included many deprecated attributes in this book, although we have chosen not to use the deprecated elements. The `font` element, for example, is quite inefficient and is becoming less frequently seen in newly created pages. Currently, use of `font` is considered bad stylistically from a design standpoint.

However, use of the deprecated attributes is not seeing appreciable decline, especially in images, lists, and tables. Indeed, not only do WYSIWYG HTML editors use the deprecated attributes liberally, but Web page authors are reluctant to learn CSS in great detail because they can do most of the same things with pure HTML.

A.10 LOGICAL ELEMENTS

This section introduces HTML's **logical elements**. They are used infrequently, and we include them for the sake of completeness. The most common ones are listed in Table A.12.

These are called *logical elements* because they logically describe their content to some extent. For example, `<p>` some text`</p>` makes no attempt to define the text, whereas `<address>` some text `</address>` gives you a better indication as to what the text represents. However, from a standard markup standpoint, most of them are useless. For example, these elements provide four different ways to make text italic.

That's clearly not their only purpose. They in some sense can provide some meaning to the document. For example, a particular search engine or search utility could easily pull out all of the citations in a group of Web pages to build a bibliography. Considering that Berners-Lee created HTML to mark up abstracts of physics research papers, the existence of these logical elements makes more sense. We will not use these elements in this book, and there is little point in allocating any further cerebral cortex to consideration of them unless some need arises in the future.

TABLE A.12 Logical Elements[a]

ELEMENT	DESCRIPTION	RECOMMENDED MARKUP
`address`	Any address	Italic
`blockquote`	Block of quoted material	Both margins indented
`cite`	Citation	Italic
`dfn`	Definition	Italic
`em`	Emphasized text	Italic
`q`	Inline quotation	Surround with quotation marks
`strong`	Stronger emphasis	Boldface
`sub`	Subscript	Subscript to preceding text
`sup`	Superscript	Superscript to preceding text

[a] These are all container elements.