What is multimedia? The technical definition, for people who wear pocket protectors, goes something like this: A computer-based method of presenting information by using more than one medium of communication, such as text or graphics and sound, and emphasizing interactivity.

That's a mouthful, but what does it all mean? It's the difference between looking up Zanzibar in your 1973 edition of the World Book Encyclopedia and dropping a CD-ROM in your computer and watching a clip of Zanzibar farmers harvest and package cloves for export. If you want to see what cloves are for, you'd need to pick up volume 3 of the encyclopedia or click a hyperlink on the computer to see and hear another clip of Julia Child talking about the wonders of cloves for desserts and pork.

Multimedia reaches its fullest potential when it's implemented in a standalone form—all of the graphics, sound, and video are contained on a CD-ROM that is connected directly to the computer. Web-based multimedia tends to pale in comparison, due to the inherent limitations of network connections and modem speeds. Many developers are working hard to bring Web-based multimedia up to the same level as a CD-ROM, however.

At the time when HTML 3.2 was released, multimedia was just beginning to get a serious hold on the World Wide Web. Standards are still not entirely in place to support this specialized content across all browsers and platforms, but as usual, browser vendors are working hard to make sure you have access to the latest and greatest in multimedia and interactive content.

Initially, Web multimedia was limited to simple animations and links to sound files. Recent advances have expanded this to full-motion video and sound, virtual reality, and interactive games. This chapter takes a look at some of the ways this content is implemented through HTML.

External and Internal Content

Multimedia on the Web is divided into two basic categories—things that can be displayed inline or as an integrated part of the browser (internal), or content that is handled outside the browser (external).

Implementing both plug-in content and external applications requires the user to have the necessary software already installed on his or her system. This can lead to some problems, because not all content is available for all platforms and browsers. Using specialty items such as multimedia means that you have to leave behind some people who won't have the chance to appreciate all of your site's features.

Implementing Internal Content

With the advancement of browsers and supporting applications, the list of internal content is growing week by week. This content includes items directly supported by a browser, such as animated GIFs, Java applets, and virtual reality. It also includes items supported through helper

applications and plug-ins, which are special applications that the browser allows to run in the browser window. Both Netscape plug-ins and Microsoft ActiveX components are examples of this kind of content. Sometimes, internal content consumes the entire browser, such as display for an Adobe Acrobat document. Other times, it only takes up as much space as an icon, such as for specialized video.

With HTML 3.2, two primary tags are used for both items. First is the `<APPLET>` tag for Java applets. The other is `<IMG>` for inserting graphic images. A couple of other tags are used by other browsers, including `<EMBED>` for Netscape plug-ins and `<OBJECT>` for ActiveX controls.

Depending on the browser's capabilities, each of these items may in turn be handled internally by the browser, or a helper application may be invoked. In both cases, the content remains as part of the rest of the Web page.

HOW DOES A BROWSER KNOW WHAT TO DO?

When your browser loads any particular file—whether it is an HTML file, image, or multimedia—how does it know what to do with it? The browser's reaction is determined by two things: the filename extension and the file's content type.

You'll see a lot of file extensions tossed around in examples throughout this book, including HTML (Web pages), GIF (images), DCR (Shockwave), and WRL (virtual reality). The browser uses the file extension to determine what to do when it retrieves a file from your local disk.

The content type is used when the browser receives a file from a Web server, because the Web server doesn't always send a filename. In some cases, the server sends only data with no file information. Instead of a filename and extension, it sends a special code called the content type, which identifies the information it's sending. Content types include `text/html`, `image/gif`, `application/mpe`, `application/msword`, and so on.

Both the browser and server contain lists of file extensions and content types. The server uses the list to determine which content type to send with a given file. The browser's list includes an additional entry for helper applications that corresponds to content types.

One of the proposals for HTML 4.0 is an expansion of the `<OBJECT>` tag to encompass all internal content. The new version of `<OBJECT>` could apply to applets, plug-ins, ActiveX, images, and any future type of embedded content.

The basic proposed syntax is

```
<OBJECT data=filename type=MIMEtype height=pixels width=pixels>
<PARAMETER name=paramName value=paramValue>
</OBJECT>
```

where `data` is the name of the file, such as `image.gif`, and `type` is the Internet content type, such as `image/gif`, followed by the height and width of the object's area in pixels. This is followed by any number of `<PARAMETER>` tags, for additional control of the content, and a closing

</OBJECT>. You can insert alternate content for incompatible browsers immediately before the closing object tag.

How does <OBJECT> work with applets, plug-ins, and ActiveX controls? Let's take a look, beginning with a Java applet. Syntax using the <OBJECT> tag is slightly different from the <APPLET> tag, shown here side by side:

```
<APPLET CLASS="driverOperator.class" CODEBASE="http://host/somepath/"
        HEIGHT=100 WIDTH=100>
   <PARAM NAME="start" VALUE="1">
   This browser doesn't drink Java.
</APPLET>

<OBJECT CLASSID="java:driverOperator.init" CODETYPE="application/java-vm"
        CODEBASE="http://host/somepath/" HEIGHT=100 WIDTH=100>
   <PARAM NAME="start" VALUE="1">
   This browser doesn't drink Java.
</OBJECT>
```

Here's what is happening attribute by attribute. The first attribute is the CLASSID. This identifies the program type (java), followed by the applet name (driverOperator) and the first method to execute (init). Note that it doesn't name the .CLASS file explicitly, as the <APPLET> tag does.

Next comes the CODETYPE. This is similar to the TYPE attribute for other Internet content, but it applies only to files containing code that must be interpreted on the client's computer. The CODEBASE is the URL where the class file is located. The last information is the height and width of the applet on the page.

After the opening <OBJECT>, you see any parameters in the same basic format as all HTML parameters and then any text content to use if any of the other tags are not recognized by the browser. This ensures compatible content for everyone, regardless of their browser or platform choice. Last is a closing </OBJECT>.

For another example, let's look at a multimedia plug-in for a Shockwave file:

```
<EMBED src="bounce_logo.dcr" alt="Hockey Pucks" width=400 height=150>
   <img src=shocknew.gif alt="Best viewed with Shockwave">
</EMBED>

<OBJECT data=bounce_logo.dcr type="application/director"
        standby="Hockey Pucks" width=400 height=150>
   <img src=shocknew.gif alt="Best viewed with Shockwave">
</OBJECT>
```

This is similar to the applet, with a couple of notable exceptions. The first exception is the data tag, which identifies the plug-in content file. It uses the type to tell the browser which helper application needs to be run. The <EMBED> tag uses alt to display a message while the content is loading, whereas <OBJECT> uses standby, although alt should still be included for nongraphic browsers.

The <EMBED> tag is followed by the old <IMG> tag, which provides alternate content for browsers that don't support objects or Shockwave. You see that even the image tag includes alternate content for nongraphical browsers. This set of tags provides content that degrades in a predictable manner, depending on the capabilities of the user's browser.

Finally, the following example shows how <OBJECT> works with an ActiveX control:

```
<OBJECT id="oompa_1" classid="clsid:663C8FEF-1EF9-11CF-A3DB-080036F12502"
   DATA="http://www.loompa.com/ole/oompa.stm">
</OBJECT>

<OBJECT id="oompa_1" classid="clsid:663C8FEF-1EF9-11CF-A3DB-080036F12502"
   data="http://www.loompa.com/ole/oompa.stm">
</OBJECT>
```

As you can see, ActiveX is the only content that isn't affected by the change to the new tag.

Now that you understand the internal content, let's move outside of the browser's sandbox.

Implementing External Multimedia Content

The browser starts external applications (also known as *helper applications* or *helper apps*), but the browser doesn't give the applications space within the browser window to do their stuff. The applications start a separate program that then loads and displays the content for the user. One popular application that fits this category is RealAudio. RealAudio is a way to deliver streaming audio (prerecorded or live) to the user across the Internet. When a user clicks a RealAudio link, the RealAudio Player launches and takes over receiving and playing the audio stream (as shown in Figure 13.1).

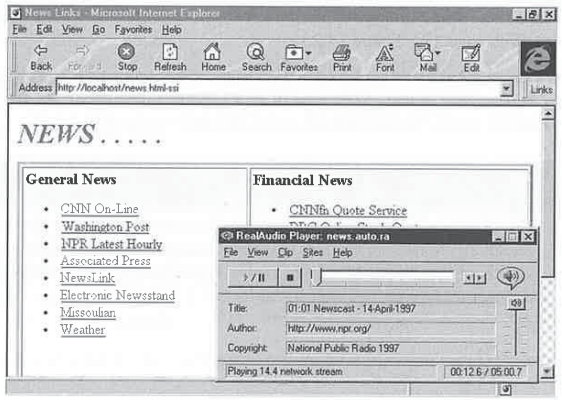


FIGURE 13.1.
RealAudio is an example of multimedia content that is accessed via a link on a Web page but delivered outside the browser via a separate application.

To connect to external media from a Web page, use the <A REF="*URL*"> tag. The path to the external file is a URL similar to an HTML page, except you're pointing to something that doesn't end in HTML. Here is an example:

```
<A HREF="volcano.mpeg">Mount St. Helens (3.6 MB)</A>
```

When the user clicks the preceding link to a video file, one of two things happens. If the content is something the browser can understand, such as some types of images or text files, the browser loads the file into its current window. If the browser can't handle it directly, it downloads the file and passes it off to the application on the user's system that is designed to read and handle that file.

A convention is used in the hyperlink, especially for large multimedia files—a notation of the file size. For users still relying on slower modem connections, this notation is a polite way of letting them know they'll be sitting and waiting a while for the content to load.

HTML 3.2 Options for Multimedia

HTML standards directly support only two technologies that are capable of delivering some form of multimedia content. The first is Java. Java is a programming language that provides methods for integrating graphics and sound into synchronized displays, complete with the capability for user interaction.

Including Animation and Sound with the Java Animator

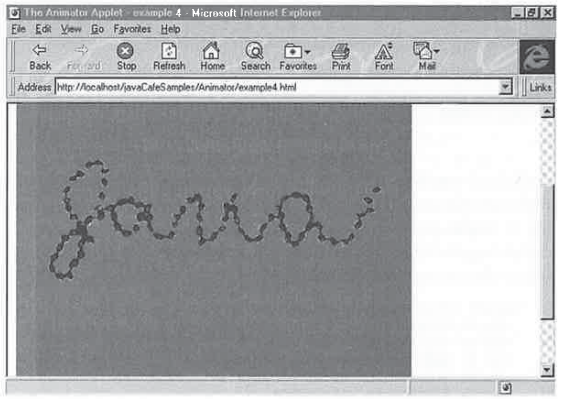
As part of its release for Java, Sun included a variety of applets to demonstrate Java's capability. Some of these applets are strictly demonstrations that really don't serve any useful purpose. Then there's the Animator applet, which anyone can use to add animation and sound to his or her Web page. (See Figure 13.2.)

To use the Animator applet on a Web page, make a copy of it and all its support files to the same directory as your Web page or in a subdirectory such as `/Animator`. The files include `Animator.class`, `ParseException.class`, and `DescriptionFrame.class`.

TIP

The location of the Animator applet depends on your system and where the Java Development Kit (JDK) files are located. Look for a `/samples/Animator` path on your computer or search for the actual `Animator.class` file. If you don't have the most recent JDK, you can download it for free from `www.javasoft.com`.

FIGURE 13.2.
The Animator applet enables page developers to assemble images and sound into a synchronized animation sequence for their Web pages.



For the easiest operation, just copy the CLASS files to the same directory as your Web page, and then create two subdirectories: `/images`, to hold the animation graphic files, and `/audio`, to hold any sound files. When you're finished, the directory structure should look something like this:

```
/MyWebPages
  MyAppletPage.html
  Animator.class
  DescriptionFrame.class
  ParseException.class
/MyWebPages/images
/MyWebPages/audio
```

Next, copy the graphic files to the `/images` directory. Animator supports both GIF and JPEG file formats. The names of the files don't matter, but a couple of conventions are built into Animator to make your life easier.

The default name that the applet looks for is `T#.gif`, in which `#` is a series of numbers beginning with 1 for the first image and continuing in order to the last image. I'll use this default for the examples.

You can also use your own name with the same naming convention, such as `dance0.gif`, `dance1.gif`, and so on. The last option is to give each animation a unique name, such as `first.gif`, `second.gif`, and so on, although I don't recommend this because it requires a lot more typing for a couple of the parameters.

Repeat the process for any sound files. The naming conventions for images don't apply to sound files.

TIP

Currently, Java supports only one sound format: 8-bit, 8,000 Hz, one-channel, AU files. You can create these on a Sun workstation using the AudioTool application or convert files from other formats using an audio format conversion utility.

With all of your files in the proper place, it's time to get to work building the HTML code to implement the animation. Using the information from the example so far, you can build the tags to implement the Animator. This example assumes eight images with eight sound files that correspond to each image. The line numbers are not part of the HTML; I added them to help in the subsequent explanation.

```
1: <applet code="Animator.class" width=460 height=160>
2:   <param name=imagesource value="images">
3:   <param name=endimage value=8>
4:   <param name=soundsource value="audio">
5:   <param name=sounds value="1.au¦2.au¦3.au¦4.au¦5.au¦6.au¦7.au¦8.au">
6:   <param name=pause value=200>
7:   <param name=repeat value="true">
8: </applet>
```

Here's what this section of HTML does, line by line:

1. The applet filename (`Animator.class`) is identified and the size is defined. Note that the other two CLASS files are not listed. They are referenced directly by Animator and don't need their own `<APPLET>` tag.
2. The directory of the images is specified with the `imagesource` parameter.
3. The last file is identified by its number, `10`. Because you're using the default naming convention, Animator will load 10 files beginning with `T1.gif` and ending with `T10.gif`.
4. The directory for the audio files is specified with the `soundsource` parameter.
5. Each of the audio files is matched with an image in the animation using the `sounds` attribute. The files are matched in order and separated with a vertical bar. If you didn't desire a sound for one of the images, you would leave its spot blank, such as `1.au¦¦3.au`. Because the sounds must be listed by filename, it's a good idea to keep the names as short as possible.
6. The `pause` parameter indicates how long, in milliseconds, the applet should wait before displaying the next image. This example uses a delay of 200 milliseconds (0.2 seconds).
7. The last parameter is `repeat`, which tells the applet whether this is a continuous loop animation or a one-time shot. Most Web animations are loops, so the default is `true`. However, I encourage you to use this parameter to avoid ambiguity when reviewing the code later.

8. As the last step, the `<APPLET>` tag is closed. If you wanted to include any alternate content, you would place it immediately before the closing tag.

HTML APPLET SYNTAX

The World Wide Web Consortium now officially supports Java with their HTML 3.2 standard, which includes the specifications for including Java applets using the `<APPLET>` tag. A complete overview of this tag is provided in Chapter 29, "Integrating Java Applets and HTML," but I've provided a quick overview here to help you understand what's happening with the Animator applet.

The basic syntax for a Java applet is

```
<APPLET code="appletName.CLASS" width="wPixels" height="hPixels">
</APPLET>
```

in which `code` is the name of the Java CLASS file and `width` and `height` specify the space in pixels that the applet should occupy on the Web page. The applet file must be of the type `CLASS` or it will not work.

The browser makes an important assumption when you use this simple version of the `<APPLET>` tag: The Java CLASS file *must* be located in the same directory as the Web page. If it is located somewhere else, such as in a special directory on the server that is dedicated to holding Java applets, you'll need to use the `codebase` attribute.

The `codebase` attribute identifies the path to the CLASS files. It is added to the `<APPLET>` tag as a relative path (`/java_applets/class_files/`) or a complete URL (`http://www.grace.net/java/class/animator/`). It's important to note that `codebase` is used only to specify a path to the applet; it doesn't identify the applet file. The CLASS file must still be identified using the `code` attribute.

The `<APPLET>` tag also includes several other attributes that are identical to their counterparts in the `<IMG>` tag: `alt`, `align`, `hspace`, and `vspace`. For anchor or script references, you can also specify a `name`.

To pass parameters to the applet, use the `<PARAM>` tag. The syntax is

```
<PARAM name="paramName" value="paramValue">
```

in which the `name` is one of the applet's parameters and `value` is what is being submitted. If the applet does not recognize the parameter `name`, the tag is ignored. Remember that each applet is a little application unto itself and can have as many parameters as it needs or have none at all.

After any `<PARAM>` tags, you can also include alternate text or images to display. Any valid HTML text formatting is allowed, including block quotes, images, and hyperlinks.

Anything other than the `<PARAM>` tags within the `<APPLET>` tags is ignored by a Java-compatible browser. Because an incompatible browser ignores the `<APPLET>` and `<PARAM>`

continues

continued

tags, you can use this feature to display other HTML content in lieu of the applet. This is a good place to provide a snapshot of what the applet looks like (with <IMG>), admonishing the user to get a Java-compatible browser (with HTML text such as <BLOCKQUOTE> or hyperlinks to a source for a Java browser [with <A href>]).

Although the basic syntax serves most purposes, Animator includes a host of parameters to fine-tune its operation. These parameters are listed in Table 13.1, along with the parameters used in our basic example.

Table 13.1. Java Animator parameters and their use.

Parameter	*Value*
`imagesource`	Directory that contains the animation frames.
`namepattern`	Template for creating names of image files to load, in the form *`prefix`*`%N.`*`suffix`*, in which `%N` is the number of the file.
`startup`	URL of the image to display at load time.
`background`	Background image for frames.
`backgroundcolor`	Background color for frames.
`startimage`	Number of the starting frame.
`endimage`	Number of the ending frame.
`images`	Explicit order for animation frames by their number, such as `1¦2¦3¦2¦1`.
`positions`	The x and y positions for each frame in the form *`x1@y1¦x2@y2¦xN@yN`*. A blank value uses the previous value.
`pause`	Number of milliseconds to pause between all images, unless otherwise specified by `pauses`.
`pauses`	Number of milliseconds to pause between frames in the form *`pause1¦pause2¦pauseN`*. Omitting a value defaults to the `pause` value.
`soundsource`	Directory containing audio files.
`sounds`	List of audio files to synchronize to individual frames in the form *`sound1`*`.au¦`*`sound2`*`.au¦`*`soundN`*`.au`.
`soundtrack`	Audio file to play in a continuous loop throughout the animation.

Parameter	*Value*
`repeat`	Query whether to repeat the animation sequence answered with `true` or `false`.
`href`	URL of a page to visit when the user clicks the animation. If not set, a click toggles pausing of the animation.

If you use the default naming convention, `endimage` is the only attribute that Animator requires.

TIP

To view the Java Animator on your browser, you'll need a recent version of Netscape Navigator (2.0 or later) or Microsoft Internet Explorer (3.0 or later). Or, you could use Sun HotJava, the first browser to support Java, which also happens to be written in Java.

Microsoft Internet Explorer is available through `www.microsoft.com/`. Netscape Navigator is located at `home.netscape.com/`. Sun HotJava is located at the JavaSoft site at `www.javasoft.com/`.

For the work involved, Animator is really a simple way to offer multimedia content to your Java-enabled users. It especially becomes handy when you throw the synchronized sound component into the bag. It might not be the equivalent of CNN film footage, but it's an ideal tool to put together an animation to make a point or illustrate a process, or even animate a hyperlink button.

Including Animated GIF Files

The other technology HTML indirectly supports through its <IMG> tag is animated GIF files. An animated GIF file contains several images combined into one package that are then displayed in turn to create a simple animation.

There are some good and bad points to this technology. The good news is that it is even simpler to implement than the Java Animator, although you lose the sound component. The bad news is that it creates very large GIF files, which can take a long time to download. In a GIF animation, all of the images it includes are saved within the same file, along with the instructions about how many times to display it.

By displaying only the first image in the file, an animated GIF will also work with browsers that don't support the animation portion. Of course, the user still must wait for the other images that are included in the animation package.

To create an animated GIF, you need the set of image files and a utility to assemble them together. You have the following options, depending on your platform:

- Windows: The most popular tool is Alchemy Mindworks GIF Construction Set, which also supports creating transparent and interlaced images. It's available from `www.mindworkshop.com/alchemy/`.
- Macintosh: GIFBuilder is a freeware utility with which you can assemble a set of GIF, PICT, or TIFF files into an animated GIF file. It's available from `www.mid.net/INFO-MAC/`.
- UNIX: With a command-line utility called whirlGIF, you can assemble a set of GIF files into an animated GIF. It includes a variety of options for the animation, which are explained at `www.msg.net/utility/whirlgif/`.

At this point, you might be wondering why you can't just use your old reliable image editor. Animation has been a capability of GIF files for quite a while, but it hasn't been supported in very many places. Its newfound popularity on the Web has caught graphics programs unprepared.

After you assemble the animation, attempt to load it with your browser. If nothing happens, you need to make sure you're using the latest version of Navigator, Internet Explorer, or Mosaic.

Non-HTML 3.2 Options for Multimedia

Other multimedia options are also covered in this chapter, including RealAudio, QuickTime movies, and virtual reality, although they are not currently part of the HTML 3.2 specification. I'll also spend a little time with browser-specific attributes to existing HTML tags.

Adding Video with `dynsrc`: Internet Explorer

Although Internet Explorer is not widely supported by other browsers, Microsoft has equipped it to handle its native AVI video files. Microsoft accomplished this by adding the `dynsrc` attribute to the `<IMG>` tag. The syntax for an image then becomes

```
<IMG src="imageURL" dynsrc="video.avi">
```

If the browser doesn't support `dynsrc`, it ignores `dynsrc` and uses the image identified in `src`.

In addition, Microsoft added several other attributes to go along with `dynsrc` to help control its display, including the following:

- `controls`: This set of controls added to the bottom of the video frame direct fast-forward, rewind, stop, and play.

- `loop`: This value determines how many times the video should replay. For example, a value of 3 causes the video to repeat three times and then stop. A value of `-1` or `INFINITE` causes it to play in an endless loop.
- `start`: The video can be started in two ways. The first is `FILEOPEN`, which causes the video to begin as soon as the entire HTML page and AVI file are loaded. The other option is `MOUSEOVER`, which prevents the video from starting until the user passes the mouse over the top of the frame.

Adding Background Sounds: Internet Explorer

Another option Microsoft added for its browser is the capability to play a sound file as a soundtrack for a page. This is accomplished with the `<BGSOUND>` tag, which has the following syntax:

```
<BGSOUND src="soundURL" loop=##¦INFINITE>
```

In this syntax, the `src` is a WAV, MIDI, or AU sound file, and `loop` specifies how many times the sound should play. If `loop` is omitted, the default is one time. Otherwise, select `-1` or `INFINITE` to play the sound in an endless loop.

Scrolling Marquees: Internet Explorer

One other Microsoft innovation that has yet to catch on with other browsers is a tag to create a scrolling display. The basic syntax is

```
<MARQUEE>Text</MARQUEE>
```

Any text between the two tags appears on its own line, scrolling from right to left at a pace slow enough to read. When the message completely disappears on the left, it starts over from the right again, and so on, ad infinitum. Incompatible browsers simply show the text within the tags inline with surrounding text.

The formatting of the surrounding text determines the appearance of the marquee text. If the marquee happens to be in the middle of body text, it appears in the same size and face as the body text. If the marquee is enclosed in a heading element, it appears in the heading typeface.

A couple of restrictions are made on this formatting. You can't change the font color or make formatting changes within the `marquee` tag. All HTML tags within `<MARQUEE>` are ignored.

A Microsoft marquee includes the following optional attributes to fine-tune its behavior, depending on your intended effect:

- `behavior`: This attribute determines how the message moves, either `SCROLL`, `SLIDE`, or `ALTERNATE`. The default is `SCROLL`. A `SLIDE` behavior is when the message comes in from the right and stops when the first letter touches the left margin. An `ALTERNATE` behavior is when the message comes in from the right, touches the left margin, and then moves back to the right margin. It then continues to bounce back and forth between the two margins.

- `direction`: This attribute, used only with `behavior=SCROLL`, determines the direction in which the marquee text moves. The default is `RIGHT`, which specifies text moving from the right side to the left. The other possible value is `LEFT`.
- `loop`: Like the attribute of the same name for AVI files and background sounds, this attribute determines how many times the marquee will scroll. The default is `INFINITE`, which also can be represented by `-1`. After a marquee stops, the message remains on the screen.
- `scrollamount`: This attribute determines the number of pixels by which to increment the text at each step in the scrolling process. Higher numbers mean faster scrolling, although it might not appear to move as smoothly.
- `scrolldelay`: This attribute determines the number of milliseconds to delay between each step in the scrolling animation. Larger numbers result in slower and less smooth scrolling.
- `bgcolor`: Normally, the marquee is transparent to the page behind it. The `bgcolor` attribute specifies a color for the space behind the marquee and is represented using a color name or an RGB hexadecimal number.
- `height, width`: These two attributes set the size occupied by the marquee. Either value can be set in pixels or as a percentage of the screen size. The default size is 100 percent of screen width and one text line of height.
- `hspace, vspace`: These attributes determine the amount of space (in pixels) between the edges of the marquee's space and other elements on the page. `hspace` stands for horizontal space (left and right), and `vspace` stands for vertical space (above and below).
- `align`: This attribute sets how text and other material adjacent to the marquee will align with the marquee's area. The values are `TOP`, `MIDDLE`, and `BOTTOM`. It does not affect text within the marquee, which is always aligned to the top of the available space.

NOTE

You also can include a marquee using one of the many Java applets designed for the purpose. The applet method guarantees compatibility with at least three browsers, while the `<MARQUEE>` tag limits your users to Internet Explorer.

It's unknown whether the `<MARQUEE>` tag will gain widespread acceptance. Sites that are designed specifically for Internet Explorer use `<MARQUEE>` widely, but outside of those situations, its use is rare.

Plug-ins: Navigator and Internet Explorer

In the early days of Mosaic and other early Web browsers, for new additions to Web technology, you had to wait for a new version of the browser. If a new type of content was introduced, such as video or virtual reality, you had to wait until a new version of the browser that supported it was introduced.

Then, Netscape developed the plug-in concept with the 2.0 version of Navigator. Plug-ins add capability to a browser without changing the browser itself by directing the browser to use helper applications. This allows the browser to display a variety of content inline with other standard Web content. Plug-ins are available for displaying and editing spreadsheets, viewing any kind of graphics file imaginable, listening to sounds in varying formats, and displaying multimedia presentations.

TIP

Netscape includes several of the more popular plug-ins, including plug-ins for virtual reality and Web chat, with its standard software release.

NOTE

The HotJava browser is designed for Java developers and programmers for customization and expansion. By adding new classes to the base browser, developers can extend HotJava without completely rewriting the code.

This is similar to the plug-in concept, but it's actually the browser that is changing; the add-on becomes part of the browser rather than a helper application.

Sun doesn't expect HotJava to become a general-use browser like Navigator or Internet Explorer. Instead, it's hoping it will become the base for corporate intranet applications, because it is easily customizable to specific situations and circumstances.

One key problem with plug-ins is that they require your users to have a browser that supports plug-ins and have the proper plug-in installed. Many plug-ins are available for only one or two platforms, leaving another chunk of users out in the cold. And, you must coordinate with your system administrator to make sure the Web server knows how to deliver the goods.

If all of these items are in place, the user is set to receive the advanced content you want to deliver. One of the most popular multimedia plug-ins available is Shockwave from Macromedia, which enables page authors to display multimedia content developed using Shockwave Director. This content can take the form of animations, electronic books, games, or other items requiring synchronized sound and video with interactivity.

NOTE

Macromedia Director is a popular tool among multimedia developers for creating presentations, including many of the multimedia CD-ROMs available for purchase. If you use Director, Shockwave provides a convenient way to put Director presentations on the Web. If you're looking to do serious multimedia work on the Web or anywhere else, Director is definitely a tool to check out.

To place a plug-in on a page, use the `<EMBED>` tag. Three attributes are required, with the syntax

```
<EMBED src="URL" width=wPixels height=hPixels>
```

in which `src` is the location of the plug-in file and `width` and `height` define how much space the plug-in will occupy on the Web page. For plug-ins that require additional parameters, add additional attributes to the `<EMBED>` tag. Check the documentation for the specific plug-in you're using to see what it requires.

To support users who aren't using plug-in–capable browsers, use the `<NOEMBED>` tag immediately following the `<EMBED>` tag. This creates a place to provide a snapshot of the plug-in in action, alternate text, or a hyperlink to the application home page. The syntax is

```
<EMBED src="URL" width=wPixels height=hPixels>
<NOEMBED>
...substitute content...
</NOEMBED>
```

Plug-in–compatible browsers ignore everything between the `<NOEMBED>` tags. If a browser doesn't recognize the `<EMBED>` tag, it also won't recognize the `<NOEMBED>` tag or display anything within its borders.

ActiveX Controls: Internet Explorer and Netscape

ActiveX is yet another option by which Web page designers can deliver specialized content to the user. This standard, developed by Microsoft, is a bit of a cross between Java and plug-ins. Like Java, it is loaded automatically when it encounters the browser. Like a plug-in, after it's loaded on the user's system, it stays there for the future and doesn't have to be reloaded.

TIP

Only Internet Explorer and Netscape Navigator support ActiveX. Other browsers might follow their lead, but there's no indication of that happening any time soon.

ActiveX controls also offer a variety of content options, including Shockwave, spreadsheets, and Windows-type controls. To include an ActiveX control, use the `<OBJECT>` tag with the syntax

```
<OBJECT classid="classID" data="contentFile" height="114" width="101">
</OBJECT>
```

in which `classid` is a 36-character string that identifies the control and `data` identifies the file that contains the content, such as a DCR file for a Shockwave Director movie. The `height` and `width` features set the space the object will occupy.

TIP

Class IDs are available from the software vendor who developed the ActiveX control.

NOTE

The World Wide Web Consortium is considering using the `<OBJECT>` tag for all specialized content, including Java applets, plug-ins, ActiveX controls, and other specialty items that have yet to be created. This would make life easier for page designers because several tags would be combined under one umbrella.

You also can use the following attributes with an ActiveX control:

- `codebase`: This attribute is the URL for the source of the control. Every Windows machine has a registry that includes virtually every bit of information about hardware and software for that particular machine. When a browser loads the class ID, it looks in the registry to see whether it's already loaded on the machine. If it is, it uses the local version. If it is not, the browser uses `codebase` to load it from the network.

  Because `codebases` can change with new versions of software, be sure to check with the software vendor to make sure you're using the right version.
- `codetype`: This attribute is an alternative method for the browser to determine whether it can handle the ActiveX object. The content type for ActiveX is `application/x-oleobject`.
- `align`: As with other content based on the `<IMG>` tag, you can use the `align` attribute to control the alignment of the ActiveX space in relation to surrounding text. The choices are `LEFT`, `RIGHT`, `TOP`, `MIDDLE`, and `BOTTOM` (default).

You should consider two final issues with ActiveX, the first of which is passing parameters to the control. To do this, use the <PARAM> tag between the <OBJECT> tags. The syntax is

```
<PARAM name="parameter" VALUE="content">
```

in which name is the parameter identifier and value is its setting. Parameter names and values will be unique for each control, so you need to consult any documentation for your specific circumstance.

Next is the issue of alternative content. As in a Java applet or Netscape plug-in, the <OBJECT> tag also supports content for incompatible browsers by inserting the appropriate HTML text, anchors, or image tags immediately before the closing </OBJECT>.

Summary

In this chapter, you learned about several options for multimedia, including external and internal media, HTML-supported options, and browser-specific extensions.

Your Web browser cannot read external media files directly. Instead, your browser links to an external file to start up a helper application to view or play those files. The link is created using the existing hyperlink anchor tag—<A href="*URL*">.

Inline multimedia is directly supported by HTML 3.2 in only two ways. The first is through Java applets, which use the <APPLET> tag to insert small Java programs that provide a variety of capabilities. One example provided with the Java Development Kit is Animator, which is a utility to provide synchronized sound and animation on a Web page.

The other multimedia method that HTML 3.2 directly supports is GIF animation. Animated GIFs are created with a separate utility that packages several image files into one and that is then interpreted and played back by the browser.

In HTML 4.0, you can expect the <OBJECT> tag to replace all of the other current tags used to insert inline content, including images, sounds, Java applets, plug-ins, and ActiveX controls. The tag, as proposed by W3C, includes all of the attributes needed by all of these items so that you'll retain the same control over their appearance and behavior.

Netscape and Internet Explorer support a variety of new tags and capabilities of those browsers, including tags for scrolling marquees, inline video, ActiveX controls, and plug-ins.

CHAPTER 14

Creating Image Maps

by Rick Darnell

IN THIS CHAPTER

Image maps are one of the most useful navigation tools developed in recent years. The first step toward this feature was simple. It was a row of separate images, each with its own hyperlink anchor. It looked something like this:

```
<A HREF="home.htm"><IMG SRC="home.gif"></A>
<A HREF="search.htm"><IMG SRC="search.gif"></A>
<A HREF="next.htm"><IMG SRC="next.gif"></A>
```

This worked well for navigation bars, but it still was a little unwieldy with the number of tags, and it slowed down the browser with three additional images to load. It also didn't allow images to be embedded inside each other.

Enter the image map concept. An image map is a graphics image (GIF, JPEG, or other) that is subdivided into regions called *hotspots*. Each of the hotspots corresponds to a different URL, enabling one graphics file to lead to many destinations. For example, you can use a geographic map of the Rocky Mountain region to create an image map that offers site-specific weather forecasts, allowing users to click on the area for the information they need. Or, you can create an image map from a picture of several people, so that a click on a person's image brings up information about that person.

Image maps are implemented in two ways. The first way is through a *server-side* map. While the image is loaded on the user's browser, the hotspot coordinates are located on the Web server. When the user clicks on an area, the coordinates are passed to the server, where they are interpreted and the corresponding URL is sent back to the URL. The browser then requests the document at the URL from the server.

The other method is through a *client-side* map. A client-side map has all of the hotspot information included as part of an HTML page. Usually, this is on the same page as the image, but it can reside elsewhere. When the user clicks on the image map, the browser does all the work to determine the URL, which is then passed to the server.

Client-side image maps are the preferred option because they are quicker and require less work from the server. Most browsers support client-side maps, although there are still many in use that support only server-side maps.

TIP

Whichever method of image mapping you use, it's a good practice to include text hyperlink versions of the links contained in the image map. If a browser doesn't support images or images are turned off, the browser still has access to the links.

Server-Side Image Maps

As mentioned in the introduction, server-side image maps are the way that this technology was introduced to the Web. The user clicks on an image, and the `x,y` coordinates are sent to a special program on the server. Other than actually creating the image map information, it is a simple matter to include on the Web page. The syntax is

```
<A href="/scriptURL/imageMapURL">
<IMG src="imageSource" ismap></A>
```

where *`scriptURL`* is the path to the script that interprets image maps, and *`imageMapURL`* is the path to the MAP file containing the hotspot coordinates. The construction of the MAP file is covered in the next two sections under each server type.

The path to the script file varies depending on the type of Web server in use. On an NCSA server, the path is probably something similar to `/cgi-bin/imagemap`, while a CERN typically uses `/htbin/htimage`. In both cases, the script file is immediately followed by the path to the image map information.

TIP

Your map file cannot reside in the top document directory of your server because there is no way to specify this directory in the concatenated format. The map coordinates *must* be in a subdirectory. Many authors include the MAP file in the same directory as the image for easy identification.

So far, the first three steps of creating a server-side image map have been covered in this chapter:

1. Select a graphics file to use for the image map. This can be any browser-compatible image file, but GIFs and JPEGs are the most common and widely supported.
2. Insert the image file using the HTML `<IMG>` tag with the `ismap` attribute. For more information on using this tag, see Chapter 12, "Adding Images to Your Web Page."
3. Enclose the image tag with a hyperlink anchor using the server image map script URL combined with the path to the image MAP file for the `href` attribute.

Now comes the last step—actually creating the MAP file. The method differs slightly depending on the type of server you're using. The two predominant options are CERN and NCSA.

NOTE

CERN stands for *Conseil European pour la Recherche Nucleaire*, or in English, *European Laboratory for Particle Physics*. It is based in Geneva, Switzerland, and is the birthplace of the World Wide Web. Researchers at CERN developed the Web in 1989 as a collaborative network.

continues

continued

NCSA stands for *National Center for Supercomputing Applications*. It is a research center affiliated with the University of Illinois at Urbana-Champaign and specializes in scientific visualization. NCSA labored in relative obscurity until it developed NCSA Mosaic, the first graphical browser for the Web.

Knowing what these two organizations are won't make you any richer, but it might win you a dozen donuts on a radio quiz show sometime.

Most servers in the United States use the NCSA HTTP standard, although CERN servers are still widespread. You'll need to check with your Internet service provider or system administrator to see which format you'll need to use.

Before you begin creating your image map, you'll need a copy of the image you're using and a way to identify pixel coordinates within it. Most popular graphics programs support this capability, including PaintShop Pro and xRes.

Creating a MAP File for CERN

There are many utilities you can use to create a MAP file, and three are covered at the end of this chapter. For now, let's leave utilities behind and work with the MAP at its lowest level. If you're using a mapping utility, these details are typically handled by the program.

At its basic level, a MAP file is simply a text file with a list of shapes, coordinates, and URLs. Four keywords are used with a CERN MAP:

- `rectangle`: Indicates a rectangular area by defining two opposite corners (usually the left top and bottom right). It can be abbreviated as `rect`:

```
rectangle (25,50) (35,70) http://www.wossomatta.edu/
rect (25,50) (35,70) http://www.wossomatta.edu/
```

- `circle`: Indicates a circle by defining the center and a radius. This is for circles only; image maps don't support ovals. It can be abbreviated as `circ`.

```
circle (25,50) 30 http://www.wossomatta.edu/
circ (25,50) 30 http://www.wossomatta.edu/
```

- `polygon`: Defining a polygon is a lot like creating a dot-to-dot puzzle. It includes a list of coordinates that are connected in turn until the last. If the figure isn't closed by a set of coordinates identical to the first, the script automatically connects the first and last set of coordinates, as in this example:

```
polygon (25,50) (35,60) (25,70) (15,60) http://www.wossomatta.edu/
poly (25,50) (35,60) (25,70) (15,60) http://www.wossomatta.edu/
```

- `default`: This specifies a URL to use if a mouse click doesn't land within any of the other shapes. It is always a good idea to add this item, even if the URL just points back to the current page:

```
default http://www.wossomatta.edu/
def http://www.wossomatta.edu/
```


The shapes are checked individually in the order in which they appear in the MAP file. Searching stops at the first match, and the corresponding URL is returned to the browser. This process has an interesting effect when used to define overlapping areas. The following snippet defines two areas—a red circle and a blue rectangle covering the circle's lower-right quarter:

```
rect (94,95) (290,213) /blue_rect.html
circ (103,93) 77 /red_circ.html
```

The image is represented in Figure 14.1. When the mouse is clicked on the overlapping area, the mapping information is evaluated in the order in which it appears in the file. Because the rectangle is listed first, its URL is returned after the click, and the information for the circle is never reached.

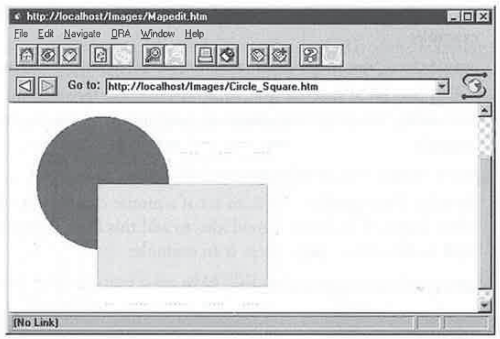


FIGURE 14.1.
Overlapping hotspots are handled by placing the topmost element in the MAP file first and following with each lower element.

The last step is to locate the `imagemap.conf` system file on the Web server. This file includes entries for every image map located on the server. You'll need to add a line with the name and location of the MAP file. Using the preceding circle and rectangle example, the entry would look like this:

```
Circle_Square : /images/Circle_Square.MAP
```

The downside to all of this work is that it can be a problem getting access to the `imagemap.conf` file if your pages are on someone else's server. Chances are you won't have the rights to modify system files. And, if your pages move from a CERN server to an NCSA server, you'll need to change the syntax of your MAP file.

Creating a MAP File for NCSA

The syntax for creating a MAP file for an NCSA server is similar to the syntax for a CERN server. However, a couple of key differences prevent a MAP file created for one type of server from working on the other.

The format of the file is the same—a text file with a list of shapes and coordinates. The following five types of values are allowed in the file:

- `rect`: Indicates a rectangular area by defining the top left and bottom right corners. Here is an example:

```
rect http://www.wossomatta.edu/ 25,50 35,70
```

- `circle`: Indicates a circle by defining the center and an outside point on the circle. This is for circles only; image maps don't support ovals. Here is an example:

```
circ http://www.wossomatta.edu/ 25,50 55,30
```

- `poly`: A polygon includes a list of up to 100 pairs of coordinates that are connected in turn until the last. If the figure isn't closed by a set of coordinates identical to the first, the script automatically connects the first and last set of coordinates. Here is an example:

```
poly http://www.wossomatta.edu/ 25,50 35,60 25,70 15,60
```

- `point`: Instead of specifying an area, this method specifies one point. The URL is resolved by returning the address of the point closest to the mouse click. Here is an example:

```
point http://www.wossomatta.edu/ 25,50
```

- `default`: This specifies a URL to use if a mouse click doesn't land within any of the other shapes. It is always a good idea to add this item, even if the URL just points back to the current page. Here is an example:

```
default http://www.wossomatta.edu/
```

TIP

It doesn't make a lot of sense to include only one point with a default line in an image map. Anywhere the user clicks will be closer to the point, and the default URL will never be returned.

Like the CERN MAP file, each set of coordinates is evaluated in the order in which it was placed in the file. This means that overlapping images also are handled in the same manner.

Client-Side Image Maps

Most major browsers now support client-side image maps. This is a method by which the browser handles all the work of associating URLs with image hotspots. This is a preferable option because it doesn't place an extra load on the server and it is faster.

An additional benefit of client-side maps appears in the status bar of the user's browser. In a server-side map, the only URL that appears when the mouse is passed over the image is the path to the CGI script and MAP file. In a client-side implementation, the URL represented by the hotspot is displayed in the status bar.

TIP

Don't forget that the URLs in an image map can point to any Web resource, including sound files, video files, and FTP servers.

A client-side image map is enabled by adding the `usemap` attribute to the `<IMG>` tag. The syntax is

```
<IMG src="imageURL" usemap="mapName">
```

where *`imageURL`* is the location of the image file, and *`mapName`* is the URL of the mapping tags. As you'll see in the next section, every set of client-side mapping information is given a unique name. For the `<IMG>` tag, this name is referenced the same as an anchor. If the map information is located in the same file as the image tag, the format is `usemap="#mapName"`. If the information is located in a different file, the format is `usemap="URL#mapName"`.

With the `<IMG>` tag in place, it's time to actually assemble the shapes and coordinates that define the hotspots in the image.

Defining a <MAP>

Creating a client-side image map requires a set of HTML tags that define the hotspots and their corresponding URLs. This utilizes the `<MAP>` and `<AREA>` tags. Their syntax is

```
<MAP name=mapName>
<AREA shape="rect|circle|poly" coords="pixelCoordinates" HREF="URL">
...
</MAP>
```

where *`mapName`* is a required attribute with a unique name for the map. This name must not conflict with other named elements on the page, such as anchors or applets; otherwise, unpredictable behavior could result.

The `<AREA>` tag requires three attributes. The `shape` attribute determines the basic shape of the hotspot and is one of four values: `rect`, `circle`, `poly`, or `default`. The `size` attribute also determines the formatting of *`pixelCoordinates`* in the `coords` attribute. The format for each of the values is slightly different, as you see in the following list:

- `rect`: A rectangular area defined by the area extending from the top left to the bottom right. Here is an example:

```
<AREA shape="rect" coords="25,50 , 55,80" href="http://www.wossomatta.edu/">
```

- `circle`: A circle (no ovals) defined by the center of the circle and the radius in pixels. Here is an example:

```
<AREA shape="circle" coords="25,50 , 30" href="http://www.wossomatta.edu/">
```

- `poly`: An irregular shape defined by a series of coordinate pairs, resulting in a dot-to-dot area. The polygon is closed by connecting the last set of coordinates to the first. Here is an example:

```
<AREA shape="poly" coords="25,50 , 35,60 , 25,70 , 15,60"
      href="http://www.wossomatta.edu/">
```

- `default`: The default URL for a click that doesn't land in any of the defined hotspots. An alternate attribute for `href` is `nohref`, which indicates that nothing should happen:

```
<AREA shape="default" href="http://www.wossomatta.edu/">
<AREA shape="default" nohref>
```

A PROPOSED METHOD FOR IMAGE MAPS

One of the new tags under consideration for subsequent versions of HTML is the `<FIG>` element. It supports client-side image maps in a way that is slightly different from the `<IMG>` and `<MAP>` combination.

The basic format for inserting an image with `<FIG>` is

```
<FIG src="Image.gif" width=150 height=50>
</FIG>
```

Note the closing tag. This is where the functionality for image maps comes into play:

```
<FIG src="Image.gif" width=150 height=50>
<H3>Destinations</H3>
<UL>
<LI><A href="http://www.wossomatta.edu/football"
       shape="rect 25,50 , 55,80" >Football</A>
<LI><A href="http://www.wossomatta.edu/fight_song"
       shape="rect 55,50 , 85,80" >Fight Song</A>
<LI><A href="http://www.wossomatta.edu/coach_canute"
       shape="rect 85,50 , 115,80" >Our Leader</A>
</UL>
</FIG>
```

There are several things to pay attention to with the new proposal. First, you can embed any HTML within the `<FIG>` elements. If the browser doesn't support images, the text is displayed instead.

Second, an additional attribute is added to the anchor tag to convert it into an image map parameter. If the image is displayed, the information in the shape attribute is used to define the hotspots. If the text is used, the shape information is ignored, and the anchor is treated as any other hyperlink.

These two features of `<FIG>` are especially useful for speech-based browsers for the sight impaired and for people still using text-only browsers who would normally miss the links offered in the image.

After you assemble a client-side map, the next question is how do you know if the user's browser will know what to do with it? Basically, you don't. Too many browsers are surfing the Web to guarantee compatibility with a client-side solution. Therefore, to ensure that everyone can use your image map, use a client-side and server-side map.

This doesn't require two images for each one. You can combine all the tags and necessary attributes within one image:

```
<A href="/cgi-bin/images/Circle_Square.MAP">
<IMG src="/images/Circle_Square.gif" usemap="#Circle_Square" ismap>
</A>
```

When interpreted by a browser that doesn't support client-side maps, the `usemap` attribute is ignored and the server-side map is used.

Another option for supporting browsers that are incompatible with client-side maps is to provide a hyperlink to the choices that would have been available through the map. The preferred location for these choices is in the same file, although they could be in a separate file. Here is an example:

```
<A href="#imageMapText">
<IMG src="/images/Circle_Square.gif" usemap="#Circle_Square">
</A>
```

In this case, if the browser doesn't support `usemap`, clicking on the image will lead the user to the location with the choices.

Image Map Editing Tools

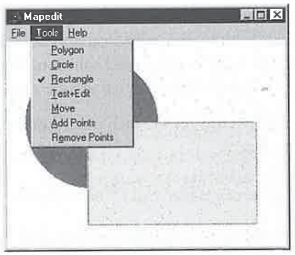
Many applications are available for a multitude of platforms that make the process of generating image maps much easier than working with a text editor and an image editor to determine coordinates and syntax. The choices are by no means limited to the applications I've reviewed here, nor are these necessarily the best utilities. This should just give you an idea of what is available for use.

Two of the easier-to-use applications are discussed in the following sections. The last includes more advanced features but isn't as easy or intuitive to use.

MapEdit

MapEdit is one of the more popular applications for making image maps. It provides a WYSIWYG (what you see is what you get) interface and comes in versions for both UNIX and Windows. (See Figure 14.2.)

FIGURE 14.2.
MapEdit is a simple visual editor for creating image maps, but it is more than adequate for most users.



MapEdit loads the image into a scrollable and resizable window, where you can draw polygons, circles, and rectangles; a URL is specified for each. It also allows you to go back, delete each hotspot, and set a default URL for clicks outside of any of the hotspots.

You can also associate comments of arbitrary length with each object if you are creating an NCSA-format map. If you are creating a client-side image map, you can provide alternate text (using the `alt` attribute) to be shown to users of nongraphical browsers.

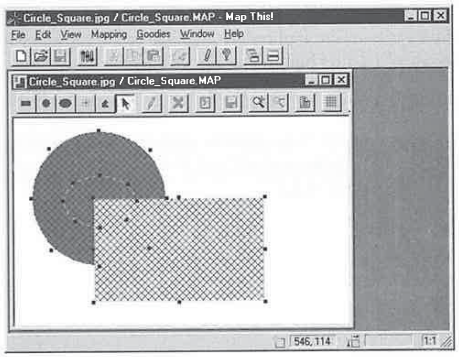
Although you can save any of your maps as client-side HTML files, MapEdit allows you to load only maps formatted for server-side mapping.

MapEdit is available on the Web at `http://www.boutell.com/mapedit/`.

Map This!

Map This! is a complete mapping utility that supports all three map formats—CERN, NCSA, and client-side—in a graphics format. (See Figure 14.3.) After reading any of the three formats, Map This! can convert it to either of the remaining two.

FIGURE 14.3.
Map This! is a freeware program that includes support for all three image map formats.



Like other image map programs, Map This! doesn't verify URLs. It is up to you to verify that you're providing valid values. Map This! also doesn't check values for your hotspots. Because of a glitch in its editing interface, it will allow you to identify hotspots in "offscreen" areas that will never be hit.

After the initial hotspots are defined, you can resize, reshape, move, or delete them. In addition to specifying the URL, Map This! also supports an additional line of explanation where you can enter a brief note. This additional information is embedded with special codes so that it won't interfere with the operation of your MAP file.

The program supports up to 1,024 areas per map. If you try to load more, it generates an error. This shouldn't be a limitation for the majority of these situations because most servers will choke before processing that many URLs.

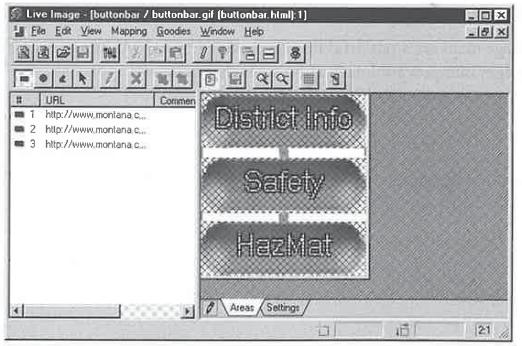
Map This! is available on the Web at

`http://galadriel.ecaetc.ohio-state.edu/tc/mt/`

Live Image

Live Image is a similar product to Map This!, except it's shareware instead of freeware and offers a few more features. (See Figure 14.4.) Live Image includes an Image Map wizard for easier creation of any type of image map. You can create your first image map in just a few minutes by answering a few questions. For general editing, Live Image also supports dragging and dropping of URL links from Netscape and Internet Explorer.

FIGURE 14.4.
LiveImage includes all of the basic features of an image map editor, plus wizards to help with common tasks and easy creation of button bars.



The program supports three file formats—GIF, JPEG, and PNG. Other file formats are available via application plug-ins. Images can also be converted between formats.

Another very useful feature is a wizard for creating button bars. LiveImage includes several predefined button images that can be put in a row, stacked, or placed in columns to create a custom button bar. The images are joined together, along with labels and links you provide, into one image file along with a companion map file. The map file can be in HTML, NCSA, or CERN format.

LiveImage is available on the Web at `http://www.mediatec.com/`.

Summary

Image maps enable you to turn a GIF or JPEG into a clickable map by designating polygons, circles, and rectangles within the image and specifying a URL for each spot. The two different kinds are server-side and client-side. A server-side map requires a special script on the Web server to interpret the coordinates from the user's mouse click, and it is slower due to the extra time needed for client/server communication. Client-side maps do the same thing server-side maps do, without the help of a special program on the server.

If you're using a server-side map, you'll need to create a MAP file that contains all of the image coordinates and URLs. The syntax for relaying the shape, coordinate, and URL information varies depending on whether you're using a CERN or NCSA-based server.

A client-side image map includes all of the mapping information embedded in the `<MAP>` and `<AREA>` tags. This enables the browser to handle all of the processing for the user's clicks without extra communication with the server. To be safe, you can combine server-side and client-side mapping information for the same image. This ensures compatibility, no matter what browser is in use.

Image maps are a powerful feature provided by HTML for your graphics. They can turn your images into powerful navigation tools that users will appreciate.

CHAPTER 15

Building and Using HTML Forms

by Rick Darnell

IN THIS CHAPTER

So far, the HTML elements covered in this book have been static creatures. You put them on a page and the user can look at them, but not much else. That all changes in this chapter.

Forms are the most important interactive feature HTML has to offer. With a form, users can give feedback and comments, submit orders and queries, or provide registration information. In short, forms open a two-way dialog between you and your users. You ask the questions by designing a form, and the user answers by filling out the form.

HTML form elements were added with HTML 2.0. All the basic features were added that computer users are already familiar with, including lists, buttons, checkboxes, and text fields. The contents are sent to a CGI script for processing or e-mailed directly to one or more recipients.

Creating a form is a two-part process. First, there's the form itself, which is a set of tags and attributes contained within a pair of `<FORM>` elements. That part is covered in this chapter. The next step is putting together a CGI script on the server to handle the form's contents. More information about processing HTML forms with databases is covered in Chapter 27, "CGI Programming," and Chapter 33, "HTML and Databases."

Beginning with Basics: <FORM>

All forms have one thing in common: They include an opening and closing `<FORM>` tag. What happens between the tags is as varied as any set of HTML. Any other text or form element is allowed within a form, including paragraphs, tables, images, and multimedia. The only thing that is *not* allowed is another form.

The usual syntax for the `<FORM>` tag is

```
<FORM method="get¦post" action="scriptURL">
```

where `method` determines how the information is sent to the script, and `action` sets the path to the CGI script that will process the contents. The URL can be a relative path for a script located on the same server as the form, or a complete URL including protocol and server name.

There are two choices for `method`—`GET` or `POST`. If the `method` is not specified, it defaults to `GET`. Data sent using the `GET` method is appended to the script URL. When it arrives at the server, it is separated and assigned to two variables—the script URL (`SCRIPT_NAME`) and the string of data (`QUERY_STRING`). It is limited by constraints on the length of the URL that can be handled by the server, which is usually 255 characters.

The other method, `POST`, is becoming the preferred method for sending information. The data is sent as a separate stream of information to the script. This enables the script to directly receive information without passing through a decoding process in the server, so the script can read an unlimited amount of information.

The script will dictate the `method` used, but here's a simple rule of thumb if you're offered a choice. Use `GET` only when the length of data input is small.

NOTE

You can use the `POST` method to test your forms, depending on the type and brand of Web server you're using. Most NCSA-type servers include a script called `post-query` in their `cgi-bin` directory. WebSite has a program script called `cgitest.exe` or `cgitest32.exe` in the `cgi-win` directory, depending on the platform and version.

To use these testing scripts, make sure your opening form element looks similar to this:

```
<FORM method="POST" action="/cgi-bin/post-query">
```

The NCSA version is usually accessed by using `/cgi-bin/post-query`, while the WebSite version uses `/cgi-win/cgitest32.exe/Form`. Check your server documentation or ask your system administrator for the path and name of the script for your site.

When a generic test script is used, its output usually resembles what you see in Figure 15.1. It consists of a list of each of the form fields and the corresponding value that was sent with it.

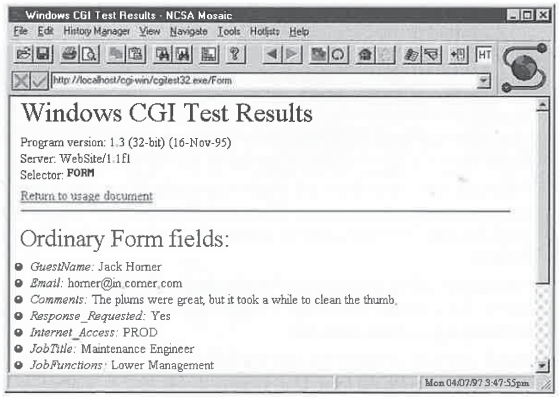


FIGURE 15.1.
A form-testing script on the WebSite server returns each of the form element names and values it received to make sure the form is working properly.

The last attribute to use with `<FORM>` is `enctype`. This attribute specifies the MIME content type for encoding the form's data. In short, it tells the browser how to format the information before passing it to the server. The default value is `x-www-form-encoded`. This is also referred to as URL encoding, and it is discussed in greater detail in Chapter 27.

Theoretically, you could use any valid MIME type, such as `text/plain`. However, most forms use the default encoding to avoid problems in manipulating data that is not encoded in some form. Form data is not encoded to keep it secret; it is encoded to ensure that input fields are easily matched to their values.

Giving a Place to <INPUT> Content

The elements for use within a form are simple but include enough variety to fit most any situation that will arise during form design. The base tag for defining each element is the <INPUT> tag, which is used to add buttons, images, checkboxes, radio buttons, passwords and text fields. The base syntax is

```
<INPUT type="elementType" name="fieldName">
```

where *elementType* defines how the input field looks on the screen, and *fieldName* assigns a keyword to the field that is used to reference its value within the CGI script. The <INPUT> tag also can have other attributes, depending on the value of the type attribute. The type-specific attributes are covered for each item.

Although the appearance of each type of input item is different to the user, they are all submitted to the CGI script as pairs of names and values.

Two other types of form elements are not implemented through <INPUT>—text boxes and lists. These are covered in their own respective sections later in this chapter.

FORM ETIQUETTE

You should follow a few simple rules with any form you put on your Web page. These aren't hard and fast rules; rather, they are just some suggestions to make your life easier—kind of like knowing which fork to use first at a fancy restaurant.

First, make the form as short and simple as possible. A form that stretches for more than a couple of pages or screens is awfully cumbersome and wears down users before they reach the end. The more questions you ask, the more users won't hang around to complete the form.

Second, plan, plan, plan. Know what information you want from the user and why you want it. If your reason for asking for a particular item, like age, is "just because," then it's probably a good idea not to ask.

Sketch your form on paper the way it should appear on the user's screen. There should be some kind of flow to the form. One element should lead to another, so that the user doesn't skip anything. If users have to guess where to go next, chances are higher that they won't get to the end of the form to press the Submit button.

It's always a good idea to tell the user why you're asking for the information. Is this to register software or to be on a mailing list? Is it to sign a guest book or provide an opinion? There are a lot of reasons for people not to trust what they see on their browser. Give them a reason to trust your request.

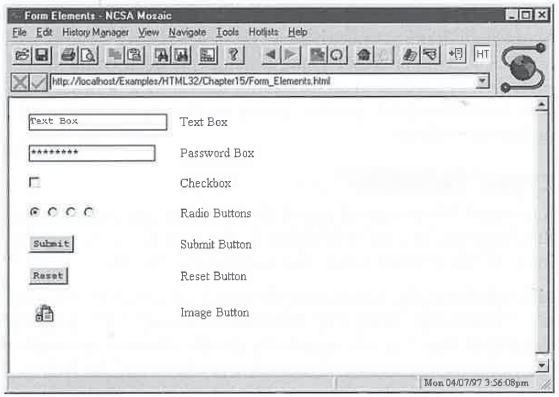
By the same token, don't lock them into submitting your form. Always give users a Reset button to clear the information and a link to more information about the form.

The eight values for type are covered in the following sections. Table 15.1 presents a rundown of each; Figure 15.2 shows their basic appearances.

Table 15.1. The eight values for input type.

Type	*Description*
TEXT	One-line text field
PASSWORD	One-line text field where the input is masked with asterisks
HIDDEN	One-line text field that is not displayed but sent to the script
CHECKBOX	Yes/no or on/off choice
RADIO	One or more checkboxes grouped together
SUBMIT	Button to submit form contents
RESET	Button to clear the form contents to default values
IMAGE	Image to click instead of a Submit button

FIGURE 15.2.
Seven of the eight input types are shown in this figure. Because it's not displayed by the browser, HIDDEN *is not shown.*



type="TEXT"

Text fields are used for any information that doesn't fit within the confines of the other types. This includes information such as names, addresses, job titles, telephone numbers, and so on. Because a text field accepts virtually any type of input, it is one of the most versatile elements.

Here are the three additional attributes that can be used with a text field:

`maxlength`	This attribute sets the maximum allowable length of the field, specified in characters. If this value is not specified, there is no limit.
`size`	This controls how wide, in characters, the box appears on the Web page. How many characters are actually displayed at one time varies because many browsers use proportional instead of fixed-width typefaces.
`value`	This is used to initialize the fields with a default string of text. The user can add, change, or delete the information if needed. If a Reset button is used, the contents of `value` are reloaded back into the field.

The following example shows a text field that allows 35 characters but displays only 20 at a time. It doesn't have a default value:

```
<INPUT type="TEXT" name="NameFirst" maxlength="35" size="20">
```

The next example is a text field that includes a default value for a state name:

```
<INPUT type="TEXT" name="State" maxlength="2" size="4" value="WY">
```

Notice the difference in value between `maxlength` and `size`. The `size` attribute is larger, allowing for a little more display room, although the input is still limited to two characters by `maxlength`. This is done to compensate for the way some browsers handle the text. If two characters were allowed for `size` also, certain two-letter pairs wouldn't fit in the box for display, such as `"WY"` and `"MA"`.

type="PASSWORD"

A password field is a special type of single-line text box that uses asterisks to mask the input. Anything typed in a password field is hidden from view to keep prying eyes away from passwords, account numbers, and other guarded information.

Although the display is masked on the screen, the actual transmission of password field contents is not encoded in any way. This provides security against people who look over the user's shoulder. It doesn't provide security for actually transmitting the information. You'll need to rely on the security features of your server to take care of that task.

In its other behavior, a password field includes the same attributes as a text field: `maxlength`, `size`, and `value`.

TIP

A default value for a password might seem a little strange, but it is actually pretty useful for things such as visitor passwords.

type="HIDDEN"

This is another type of text field, but nothing is shown on the screen. It provides a way to send additional information to the CGI script that cannot be changed by the user. For example, to send the name of an HTML file, you could use a line such as

```
<INPUT type="HIDDEN" name="File" value="survey.html">
```

Some CGI programs use hidden fields to pass information from one page to another. If a user types a name or account number on one form, the CGI script processes it and includes it on a second follow-up form that retains the information in a hidden field. This makes it easier for the users because they don't have to supply the information again.

TIP

Even though the field is hidden from view on the browser, it can still be examined by the user through looking at the HTML source code. The hidden input field is not a good place to mumble nasty things under your breath.

type="CHECKBOX"

Checkboxes are used to record yes/no or true/false information such as the following:

- Do you want to receive more information about our products?
- Are you age 18 or older?
- Do you own a car?

The checkbox is typically represented as a small box that the user clicks to place or remove an *x* or checkmark. This type of input element returns a value to the CGI script only if it's selected. If the box isn't checked, no value is returned to the script.

There are two attributes to consider with a checkbox. The first is `value`. This is the value sent to the CGI script if the checkbox is selected. If omitted, the default value is `"on"`.

The other attribute is `checked`. This controls the initial status of the checkbox. The following example is for a field that a user selects to be placed on a company mailing list. The element defaults to `"yes"`:

```
<INPUT type="checkbox" NAME="send_mail" VALUE="yes" checked>
```

TIP

Some older browsers do not support the checkbox. If an input type is not supported, it defaults to a text box representation.

type="RADIO"

Radio buttons are similar to checkboxes, but they present a range of choices. Only one radio button in a group is selected at a time. Selecting another button in the same group deselects the current choice.

The first attribute to use with radio buttons is `name`. This is used to identify each of the buttons in the group. One `<INPUT>` tag is required for each button, with each button having the same `name`. Then, each button is assigned a `value` to return if it is selected, as shown in the following:

```
Cash <INPUT type="radio" name="payment_type" value="Cash">
COD <INPUT type="radio" name="payment_type" value="COD">
Visa <INPUT type="radio" name="payment_type" value="Visa">
Mastercard <INPUT type="radio" name="payment_type" value="Mastercard">
American Express <INPUT type="radio" name="payment_type" value="Amex">
```

The preceding set of radio buttons presents a list of payment options. However, if the user doesn't select one of the payment types, nothing is returned to the CGI script. You can specify a default value by adding the `checked` attribute to one of the buttons.

Only one button in a group can be initially selected, although it's possible to include the `checked` attribute with more than one button. This situation is usually resolved by the browser by selecting the last button with `checked`. However, this is not consistent across all browsers and versions. Usually, the last item is marked for selection, but Netscape and Microsoft occasionally use the first, depending on the version. For consistent results, make sure only one button is marked as default.

type="SUBMIT"

Although it is included with input types, the `SUBMIT` type results in a button that, when pressed, sends the contents of the form to the CGI script specified in the `action` attribute of the `<FORM>` tag.

NOTE

Scripting languages such as JavaScript and Visual Basic Script can add additional functionality to buttons and forms beyond just submitting the contents to a URL. You can calculate totals, load pages, create frames, and perform other activities besides the basic submit and reset functions.

See Chapter 28, "Integrating JavaScript and HTML," and Chapter 30, "Integrating ActiveX and VBScript," for more information.

The default label on a Submit button is Submit. You can use the `value` attribute with a Submit button to change the default to something else, as shown in the following

```
<INPUT type="submit" value="Place your order">
```

The button will grow in width to accommodate any text you add. If the available width is constrained by another element, such as a table, the button's height will also adjust until all text fits.

type="RESET"

Clicking a Reset button restores any default values specified in individual form elements. If a default value is not defined, any content within the element is erased.

Like the Submit button, you can also change the name of the Reset button using the `value` attribute.

type="IMAGE"

The image type is similar to the Submit button, but it uses a graphic instead of a button. One attribute is required and two attributes are optional. The base syntax for the image type is

```
<INPUT type="IMAGE" src="ImageURL">
```

where *`ImageURL`* is the location and name of the image file. In addition, you also can control the alignment of the image in relation to other items around it by using the `align` attribute. The legal values are `TOP`, `MIDDLE`, and `BOTTOM`, and they behave the same as the corresponding values for the `<IMG>` tag, described in Chapter 12, "Adding Images to Your Web Page."

One of the features of using an image is that the coordinates of the pointer's x and y position on the image are sent to the browser, allowing you to use it as an image map for different processing options depending on where the user clicked.

The last attribute is `name`. The name assigned to the image is also sent to the CGI script in the form

```
Name.x=xPosition
Name.y=yPosition
```

where *`Name`* is the name assigned to the image, and *`xPosition`* and *`yPosition`* are the x and y coordinates where the user clicked. This allows an additional degree of control over processing by providing the CGI script a way to differentiate between images and areas within the images.

<TEXTAREA>

The <TEXTAREA> input type uses its own tag. It is related to <INPUT type="TEXT"> in that it receives free form text input. (See Figure 15.3.) However, a text area allows the user to enter larger amounts of information with multiple lines.

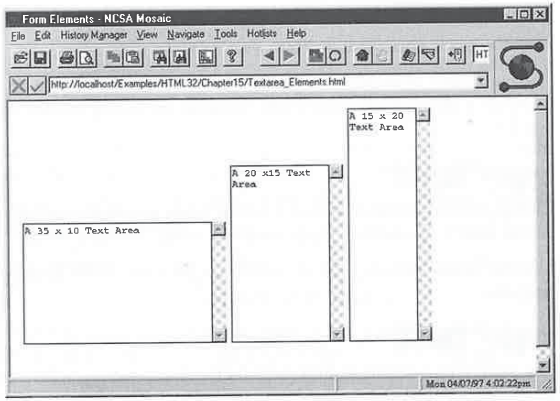


FIGURE 15.3.
A text area form element allows an unlimited amount of input using multiple lines within a scrolling box.

TIP

If you're going to allow your users to use elements such as <TEXTAREA> to submit large blocks of information, be sure to use the POST method in the <FORM> element.

The first attribute of <TEXTAREA> is the same as other input types. This attribute is name, which assigns a keyword to the field that is used to reference its value within the CGI script.

The next two attributes set the visible size of the text area—rows and cols. As you can discern from the names, rows describes how high the area is in text lines, while cols sets the width of the area in characters. Because a user can enter text beyond the horizontal or vertical constraints of the box, the user's browser will provide scrollbars to assist in editing.

A text area form element requires an opening and closing tag. Between the opening and closing tags, you can provide a default set of text to use when the form is first displayed, which is also reinstated if the Reset button is pressed. If no default text is desired, the closing tag immediately follows the opening tag.

<SELECT>

The last type of form element is a selection box. It can take the form of a drop-down list or a menu of items. (See Figure 15.4.) It is similar to the radio button input type, although it is easier to create and more flexible to use. The syntax for a selection box is

```
<SELECT name="selectionName" [size=visibleItems][multiple]>
<OPTION [selected]>Option1 Text
...Add'l options...
</SELECT>
```

where name is used the same as other form input items, size determines how many options are visible at one time, and multiple controls whether more than one item can be chosen at one time.

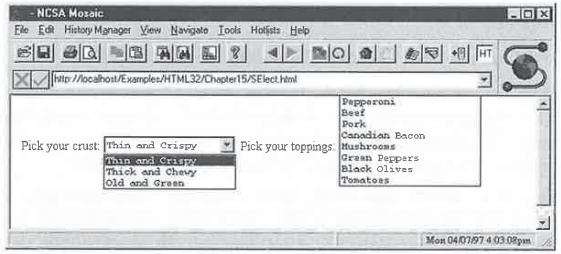


FIGURE 15.4.
The selection box on the left includes the multiple *attribute, and the one on the right does not.*

The default value for size depends on the browser and whether multiple is used. On any browser, if multiple is not used, the default size is one line. A button with an arrow on it is provided to display the rest of the list.

If multiple is used, on the other hand, the behavior depends on the browser. On some browsers, such as NCSA or Spyglass Mosaic, the default for size becomes the number of items on the list. On Internet Explorer, however, the default is four lines, with a scrollbar provided to access the rest. If you need a consistent appearance, use the size attribute.

TIP

The width of a selection box on a Web page is determined by the longest item option. There is not a way to force the box's width to a predetermined size.

Each item in the list is specified using an <OPTION> tag. The text to the right of the tag is treated the same as value is in other form items such as text boxes. Alternatively, you can use the value attribute with <OPTION> to specify the text to send to the CGI script, and then the text is treated like a label.

If the `selected` attribute is included, that item is chosen initially. More than one item can be marked with `selected` if the `multiple` attribute is also used in the `<SELECT>` tag.

A Completed Form

Now that you've been through each of the form elements and their various and sundry attributes, it's time to take a step back and look at an entire form. This form is designed to connect with a CGI script that will post the contents to a guest book database.

The guest book includes requests for a wide variety of information, including the user's name and e-mail address, requests for a reply, comments, and some demographic information.

First, create the opening form tag. Because this form has the potential to return a lot of information, use the `POST` method. The name of the script is `guest_register.pl`, which is located in `/cgi-bin`:

```
<FORM method="POST" action="/cgi-bin/guest_register.pl">
```

With the container tags for the form in place, it's time to start adding individual elements. Three things are wanted the most from this form—the user's name, e-mail address, and comments. The first two items are handled with text fields. The name is limited to 50 characters, and the e-mail address should not be longer than 35 characters:

```
Your name: <INPUT type="TEXT" name="GuestName" maxlength="50">
Email address: <INPUT type="TEXT" name="Email" maxlength="35">
```

Note the text placed to the left of the input tags. These are labels to let the user know what is desired for each box. The next field to include is a text area for comments. This is placed on its own line:

```
<BR>Comments: <TEXTAREA name="Comments" rows=5 cols=60>(No comment.)</TEXTAREA>
```

Before you start adding elements to gather demographic information, make sure the important items are emphasized. First, a pair of Submit and Reset buttons have been included right after the text area, followed by a horizontal rule to separate the section from the one below it. Then, some text is added below the horizontal rule to tell the user that completing the demographic information is purely optional:

```
<P><INPUT type="SUBMIT"><INPUT type="RESET"></P>
<HR>
<P>In addition to your comments, we'd also like to ask you a few questions
about who you are and how you found us. If you don't want to complete
this information, use the Submit button above. Thank you.</P>
```

With the first three items in place, turn your attention to some demographic information that is wanted from the user. Because this form is only an example, the demographic information will include several types of form elements so that you can see how they would look in actual usage.

Start with a checkbox and a set of radio buttons:

```
Would you like for a response to your comments?
<INPUT type="CHECKBOX" name="Response_Requested" value="Yes">
 <TABLE WIDTH="30%">
  <CAPTION>Who do you use for Internet/ World Wide Web access?</CAPTION>
  <TR>
   <TD WIDTH="5%">
   <INPUT TYPE="radio" NAME="Internet_Access" VALUE="LISP" SELECTED="selected">
   <TD>Local ISP
  <TR>
   <TD><INPUT TYPE="radio" NAME="Internet_Access" VALUE="RISP">
   <TD>Regional ISP
  <TR>
   <TD><INPUT TYPE="radio" NAME="Internet_Access" VALUE="AOL">
   <TD>America Online
  <TR>
   <TD><INPUT TYPE="radio" NAME="Internet_Access" VALUE="CIS">
   <TD>Compuserve
  <TR>
   <TD><INPUT TYPE="radio" NAME="Internet_Access" VALUE="MSN">
   <TD>Microsoft Network
  <TR>
   <TD><INPUT TYPE="radio" NAME="Internet_Access" VALUE="PROD">
   <TD>Prodigy
 </TABLE>
```

The `checked` attribute is not part of the checkbox, so the default answer for requesting a reply is "no." A table is used to format the radio buttons so that they line up in a neat vertical column with their labels adjacent. Next on the agenda is a drop-down list for the user's job title and a menu list for their job functions. Another table is used to make sure these two items appear on the same horizontal line:

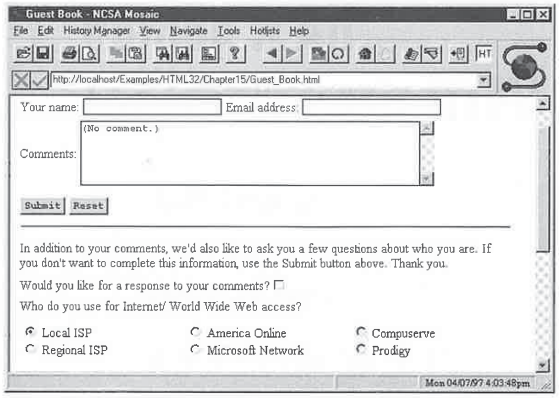
```
<TABLE width="100%">
<TR>
<TD>Your job title:<SELECT NAME="JobTitle">
  <OPTION VALUE="">Owner</OPTION>
  <OPTION VALUE="">President</OPTION>
  <OPTION VALUE="">CEO</OPTION>
  <OPTION VALUE="">Supervisor</OPTION>
  <OPTION VALUE="">Maintenance Engineer</OPTION>
  <OPTION VALUE="">Flight Attendant</OPTION>
  <OPTION VALUE="">Firefighter</OPTION>
  <OPTION VALUE="">Counterespionage Agent</OPTION>
</SELECT>
<TD>Your job duties (select all that apply):<BR>
<SELECT NAME="JobFunctions" MULTIPLE>
  <OPTION VALUE="">Upper Management</OPTION>
  <OPTION VALUE="">Middle Management</OPTION>
  <OPTION VALUE="">Lower Management</OPTION>
  <OPTION VALUE="">Receivables</OPTION>
  <OPTION VALUE="">Payables</OPTION>
  <OPTION VALUE="">Global Terrorism</OPTION>
</SELECT>
</TABLE>
```

With the addition of a little thank-you and a closing `</FORM>` tag, the form is completed. The entire finished document is shown in Listing 15.1 and Figure 15.5.

Listing 15.1. An HTML document with a form to gather guest book information.

```
<HTML>
<HEAD>
<TITLE>Guest Book</TITLE>
</HEAD>
<BODY>
<H1>Please sign our Guest Book...</H1>
<FORM method="POST" action="/cgi-bin/guest_register.pl">
Your name: <INPUT type="TEXT" name="GuestName" maxlength="50">
Email address: <INPUT type="TEXT" name="Email" maxlength="35">
<BR>Comments: <TEXTAREA name="Comments" rows=5 cols=60>(No comment.)</TEXTAREA>
<P><INPUT type="SUBMIT"><INPUT type="RESET"></P>
<HR>
<P>In addition to your comments, we'd also like to ask you a few
questions about who you are. If you don't want to complete
this information, use the Submit button above. Thank you.</P>
Would you like for a response to your comments?
<INPUT type="CHECKBOX" name="Response_Requested" value="Yes">
<BR>Who do you use for Internet/ World Wide Web access?
<TABLE WIDTH="100%">
<TR>
  <TD><INPUT TYPE="radio" NAME="Internet_Access"
             VALUE="LISP" SELECTED="selected">Local ISP
   <BR><INPUT TYPE="radio" NAME="Internet_Access" VALUE="RISP">Regional ISP
  <TD><INPUT TYPE="radio" NAME="Internet_Access" VALUE="AOL">America Online
   <BR><INPUT TYPE="radio" NAME="Internet_Access" VALUE="MSN">Microsoft Network
  <TD><INPUT TYPE="radio" NAME="Internet_Access" VALUE="CIS">Compuserve
   <BR><INPUT TYPE="radio" NAME="Internet_Access" VALUE="PROD">Prodigy
</TABLE>
<TABLE>
<TR valign="TOP">
<TD>Your job title:
<BR><SELECT NAME="JobTitle">
  <OPTION>Owner
  <OPTION>President
  <OPTION>CEO
  <OPTION>Supervisor
  <OPTION>Maintenance Engineer
  <OPTION>Flight Attendant
  <OPTION>Firefighter
  <OPTION>Counterespionage Agent
</SELECT>
<TD>
<TD>Your job duties (select all that apply):<BR>
<SELECT NAME="JobFunctions" MULTIPLE>
  <OPTION>Upper Management
  <OPTION>Middle Management
  <OPTION>Lower Management
  <OPTION>Receivables
  <OPTION>Payables
  <OPTION>Global Terrorism
</SELECT>
</TABLE>
<H3>Thank you for taking the time to answer these questions.
Your time and input is appreciated.</H3>
</FORM>
</BODY>
</HTML>
```


FIGURE 15.5.
The finished guest book form as it appears on a user's browser.



Summary

Forms are a way of gathering information from the user that is processed through a CGI script. An HTML form consists of a pair of <FORM> tags that encase one or more elements, including text boxes, menus, checkboxes, and buttons. Other HTML elements can also be used to label and format the form items, such as text, images, horizontal rules, and tables.

Forms were implemented in HTML 2.0, and the tags for creating forms are widely supported in just about every available browser. There might still be a problem in some older browsers with compatibility with a couple of tags—primarily the checkbox and radio button.

In this chapter, you learned how to include each form element in a Web page, as well as how to construct the form itself so that when it's submitted it calls the right programs on the server to process the information.

CHAPTER 16

Putting It All Together: Basic HTML

by Rick Darnell

IN THIS CHAPTER

You've spent several chapters learning about the various elements of HTML—what each element does and how to make it work. You've learned about headers, footers, text alignment, lists, tables, links, images, multimedia, image maps, and forms.

Now it's time to put all these elements together to assemble Web pages that do something useful—deliver your message to the world. If you've never worked with HTML 3.2 or are just looking for a few new ideas, this chapter is for you.

This chapter focuses on building several HTML pages that incorporate most of the standard features of HTML 3.2, including home pages, directories, and news and information items. I'll introduce basic HTML design, a topic that is further addressed in Chapters 23 through 26, which cover effective Web page design in much more detail.

NOTE

All of the examples in this chapter are from a real Web site for the Missoula (Montana) Rural Fire District. You can view this site at `www.montana.com/mrfd/`.

Basic Page Layout Issues

Before beginning to create your page, you must answer two important questions:

- What do you want to say?
- To whom do you want to say it?

Although these are two separate questions, they are hard to separate. What you want to say depends on to whom you want to say it. As you try to answer these questions, more questions arise. The following questions help focus your efforts even more:

- Are you targeting your pages to potential customers?
- Do you want to provide unique information on a special topic or issue?
- Is this a method of keeping employees and other company personnel informed of developments and news companywide?

If you don't have an audience or a message, there's a good chance you'll end up saying nothing to anyone. This is also related to the topics covered in Chapter 15, "Building and Using HTML Forms." Form follows function, and the lack of one shows up in the other.

After you know what your basic message is and how it should be organized for delivery to your audience, it's time to plug the individual parts of the message into the framework. If the Web site is the bell, the pages are the hammers that make the sound.

Before you start a new page, remember that the overall design and layout of your page is how your message is communicated to the user. Good design and layout is not an end in itself. Looking at many pages on the Web, you'll see the full gamut—from bland pages with no thought given to their appearance to pages whose only message is to show off how artistic a designer can be. As you start to play with the design, remember one cardinal rule of layout: Self-indulgence is a message-killer.

It's a good idea to flip through magazines, books, and brochures before you start. Web page design evolves from printed page design, and the printed page is a good place to start looking for ideas.

Although the number of formatting options and capabilities are growing for Web pages, they are still limited in comparison to their printed cousins. With a little creativity, however, it's possible to make your page stand out.

A FEW WORDS ABOUT A LOT OF SPACE

Space is an important part of every page. It gives breathing room in the midst of a crowd of information. Effective use of space can help a page as much as ineffective use can hurt a page.

Having too much space causes the user to waste time with the scroll bars to get to the next piece of the page. Having too little space causes everything to run together like gumbo. An extra set of eyes can help judge the overall effect. Enlist the help of friends and coworkers to critique your work. It's some of the best advice you can get free of charge.

The following five layouts use the default font the browser provides and the same image and headline size. Even with this limitation, different uses of headline, text, images, and white space result in a very different look and feel for each. Depending on your use and placement of the four page elements, each layout can take on its own personality, which you can harness to help reinforce your message.

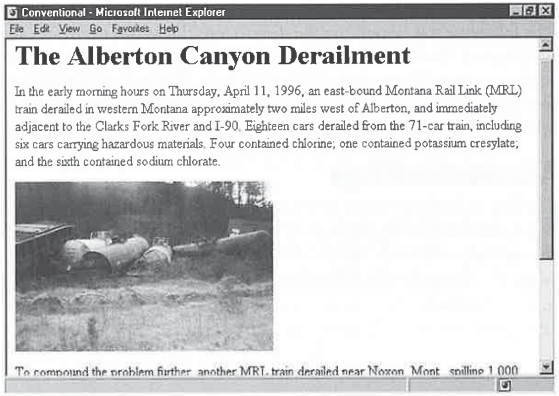
A Conventional Page

The first style of page is one that is seen all too often on the Web. It is also the easiest to construct and doesn't really require any knowledge of design or HTML. It consists of a heading at the top, followed by the bulk of the text, and perhaps an image included at the end or somewhere else along the way. (See Listing 16.1 and Figure 16.1.)

Listing 16.1. A conventional page doesn't take advantage of any of HTML's formatting capabilities. It is also the most compatible with all browsers and platforms.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
 <TITLE>Conventional</TITLE>
</HEAD>
<BODY>
<H1>The Alberton Canyon Derailment</H1>
<P>In the early morning hours on Thursday, April 11, 1996, an east-bound
Montana Rail Link (MRL) train derailed in western Montana approximately
two miles west of Alberton,  and immediately adjacent to the Clarks Fork
River and I-90.  Eighteen cars derailed from the 71-car train, including six
cars carrying hazardous materials. Four contained chlorine; one contained
potassium cresylate; and the sixth contained sodium chlorate.</P>
<P><IMG SRC="/mrfd/Alberton/albert8.jpg" WIDTH="300" HEIGHT="197"></P>
<P>To compound the problem further, another MRL train derailed near Noxon,
Mont., spilling 1,000 gallons of diesel fuel and overturning another chlorine
car.  The additional confusion caused by the diversion and rerouting of
resources slowed the initial response of personnel and equipment.</P>
<P>One of the chlorine cars and the potassium cresylate car were breached,
resulting in the release of approximately 100,000 lbs. of deadly chlorine
gas. The cloud drifted over the highway and through the town of Alberton
forcing the evacuation of more than 1,000 people in a 100 square-mile area.
Half of these were kept out of their homes for 17 days.  An estimated 350
people were treated for health and respiratory problems in Missoula hospitals,
35 miles to the east. A lone fatality occurred when a transient riding in a
box car walked into the cloud after surviving the crash.</P>
</BODY>
</HTML>
```

FIGURE 16.1.
A conventional page: thick with text, heading at the top, and picture at the bottom.



Although you wouldn't want all of the pages in your site to look like this one, it is a quick and dirty way of putting a page together to disseminate information until you can put something else together. It is also a good alternative page to use if you're developing a mirror for older browsers that might not support HTML 3.2 advanced features.

A Modern Layout

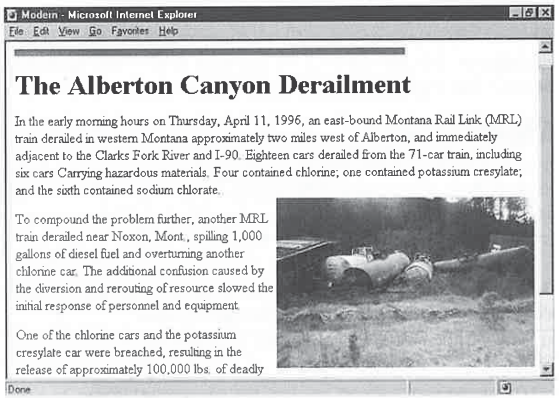
A modern layout is similar to the text-heavy conventional layout, but uses horizontal rules to mark the beginning and end of the page and images that extend into the text. (See Listing 16.2 and Figure 16.2.) A true modern layout would also include extra space between each of the text lines, but this is not yet possible with HTML 3.2.

Listing 16.2. The main feature of a modern layout is the use of left- and right-alignment on images, allowing text to flow around the picture rather than stay inline.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
 <TITLE>Modern</TITLE>
</HEAD>
<BODY>
<HR WIDTH="75%" NOSHADE SIZE="8" ALIGN="LEFT">
<H1>The Alberton Canyon Derailment</H1>
<P>In the early morning hours on Thursday, April 11, 1996, an east-bound
Montana Rail Link (MRL) train derailed in western Montana approximately
two miles west of Alberton,  and immediately adjacent to the Clarks Fork
River and I-90.  Eighteen cars derailed from the 71-car train, including six
cars carrying hazardous materials. Four contained chlorine; one contained
potassium cresylate; and the sixth contained sodium chlorate.
<IMG SRC="/mrfd/Alberton/albert8.jpg" ALIGN="RIGHT" WIDTH="300" HEIGHT="197"></P>
<P>To compound the problem further, another MRL train derailed near Noxon,
Mont., spilling 1,000 gallons of diesel fuel and overturning another chlorine
car.  The additional confusion caused by the diversion and rerouting of
resources slowed the initial response of personnel and equipment.</P>
<P>One of the chlorine cars and the potassium cresylate car were breached,
resulting in the release of approximately 100,000 lbs. of deadly chlorine gas.
The cloud drifted over the highway and through the town of Alberton forcing
the evacuation of more than 1,000 people in a 100 square-mile area. Half of
these were kept out of their homes for 17 days.  An estimated 350 people were
treated for health and respiratory problems in Missoula hospitals, 35 miles to
the east. A lone fatality occurred when a transient riding in a box car walked
into the cloud after surviving the crash.</P>
<HR WIDTH="75%" NOSHADE SIZE="8" ALIGN="RIGHT">
</BODY>
</HTML>
```

You can easily convert a conventional layout to a modern layout simply by adding `LEFT` or `RIGHT` to the image's `align` attribute. It significantly improves the appearance of the page with minimal effort.

FIGURE 16.2.
A more modern layout, with use of horizontal rules and pictures integrated into the text.



A Classic Layout

Adding columns to a page does two things. First, it shortens line length, making it easier for the user to read the text. This is especially important on a Web page, where most users are reading a screen, which causes greater eyestrain than looking at a page.

Second, columns conserve space in larger documents. Although it seems though it takes up more space, it actually reduces the total length of the document. This is part of the reason why newspapers use multiple columns.

A classic layout is one of the simplest implementations of columns in a Web page. (See Listing 16.3 and Figure 16.3.) Because HTML 3.2 doesn't directly support columns, however, tables fit the purpose quite nicely to achieve the same effect.

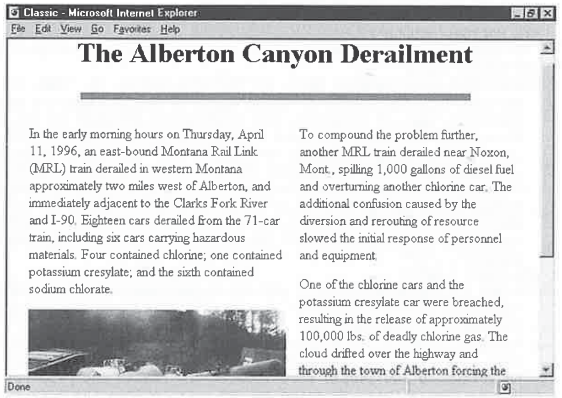
Listing 16.3. This is the first layout that makes use of tables to create columns on a Web page. The headline is placed outside of the table and centered to give it prominence.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
 <TITLE>Classic</TITLE>
</HEAD>
<BODY>
<H1 ALIGN="CENTER">The Alberton Canyon Derailment</H1>
<HR WIDTH="75%" SIZE="8" ALIGN="CENTER" NOSHADE>
<TABLE WIDTH="100%" BORDER="0" CELLSPACING="15">
 <TR VALIGN="TOP">
  <TD WIDTH="50%">
   <P>In the early morning hours on Thursday, April 11, 1996, an east-bound
   Montana Rail Link (MRL) train derailed in western Montana approximately
```



```
   two miles west of Alberton,  and immediately adjacent to the Clarks Fork
   River and I-90.  Eighteen cars derailed from the 71-car train, including six
   cars carrying hazardous materials. Four contained chlorine; one contained
   potassium cresylate; and the sixth contained sodium chlorate.
   <P><IMG SRC="/mrfd/Alberton/albert8.jpg" WIDTH="300" HEIGHT="197"></P>
  <TD WIDTH="50%">
   <P>To compound the problem further, another MRL train derailed near Noxon,
   Mont., spilling 1,000 gallons of diesel fuel and overturning another
   chlorine car. The additional confusion caused by the diversion and rerouting
   of resources slowed the initial response of personnel and equipment.</P>
   <P>One of the chlorine cars and the potassium cresylate car were breached,
   resulting in the release of approximately 100,000 lbs. of deadly chlorine
   gas. The cloud drifted over the highway and through the town of Alberton
   forcing the evacuation of more than 1,000 people in a 100 square-mile area.
   Half of these were kept out of their homes for 17 days.  An estimated 350
   people were treated for health and respiratory problems in Missoula
   hospitals, 35 miles to the east. A lone fatality occurred when a transient
   riding in a box car walked into the cloud after surviving the crash.</P>
</TABLE>
</BODY>
</HTML>
```

FIGURE 16.3.
This classic layout is often used in varying forms. It is a simple, two-column format with a centered headline and image set into the text.



The two columns should be of equal width, and only one requires the `width` attribute. However, both `<TD>` tags include the `width` specification, which helps you to avoid ambiguity during later review and revision.

A Technical Information Page

The layout for technical documents evolved out of a need for practical features. Originally, many of these pages were closer in appearance to the conventional design. This meant a lack of

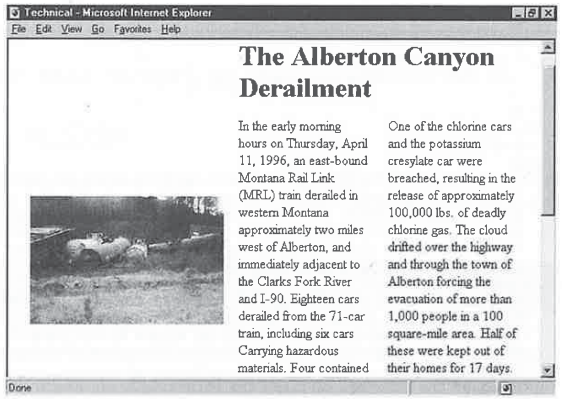
space to make notes or comments about the contents, however. By shifting to a three-column format and leaving one of the outside columns open for diagrams and illustrations, additional white space was acquired.

The resulting design is very angular, and the ample white space results in a clean and strong appearance. (See Listing 16.4 and Figure 16.4.)

Listing 16.4. The technical page design is similar to the two-column classic, except a third column is added to the table to hold images.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
 <TITLE>Technical</TITLE>
</HEAD>
<BODY>
<TABLE BORDER="0" WIDTH="100%" CELLSPACING="15">
 <TR>
  <TD>
  <TD COLSPAN="2" WIDTH="33%">
   <H1>The Alberton Canyon Derailment</H1>
 <TR VALIGN="TOP">
  <TD VALIGN="MIDDLE">
   <P><IMG SRC="/mrfd/Alberton/albert8.jpg" WIDTH="225" HEIGHT="148"></P>
  <TD WIDTH="33%">
   <P>In the early morning hours on Thursday, April 11, 1996, an east-bound
    Montana Rail Link (MRL) train derailed in western Montana approximately
    two miles west of Alberton, and immediately adjacent to the Clarks Fork
    River and I-90. Eighteen cars derailed from the 71-car train, including
    six cars carrying hazardous materials. Four contained chlorine; one
    contained potassium cresylate; and the sixth contained sodium chlorate.</P>
   <P>To compound the problem further, another MRL train derailed near Noxon,
    Mont., spilling 1,000 gallons of diesel fuel and overturning another
    chlorine car.  The additional confusion caused by the diversion and
    rerouting of resource slowed the initial response of personnel and
    equipment.</P>
  <TD WIDTH="33%">
   <P>One of the chlorine cars and the potassium cresylate car were breached,
    resulting in the release of approximately 100,000 lbs. of deadly chlorine gas.
    The cloud drifted over the highway and through the town of
    Alberton forcing the evacuation of more than 1,000 people in a 100
    square-mile area. Half of these were kept out of their homes for 17 days.
    An estimated 350 people were treated for health and respiratory problems in
    Missoula hospitals, 35 miles to the east. A lone fatality occurred when a
    transient riding in a box car walked into the cloud after surviving the
    crash.</P>
    </TABLE>
</BODY>
</HTML>
```


FIGURE 16.4.
This design is good for technical information. It combines some of the features of the conventional and modern layouts, but uses more white space.



A Formal Page

Formal pages aren't necessarily the most space-efficient designs in the world, but they do have a certain style that fits the bill for major announcements, such as the naming of a new chief or the receipt of a new fire truck.

The design is created using a table with three columns. The width of the outside two columns is 25 percent of the table width, leaving half of the table width for the center column and the text (as shown in Listing 16.5 and Figure 16.5). Then, each content element is placed in its own cell—first the headline, then the image, and finally the text.

Listing 16.5. In a formal design, three columns create the space on the left and right of the text and control the headline and initial image space.

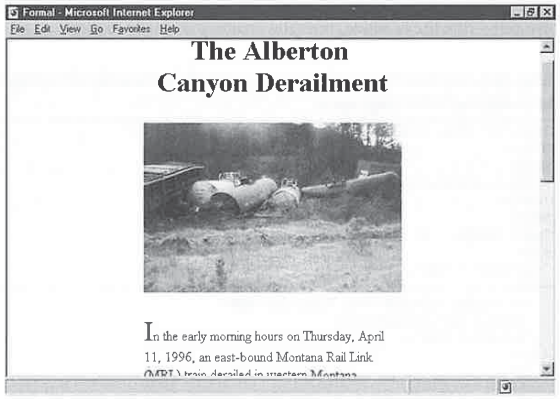
```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
 <TITLE>Formal</TITLE>
</HEAD>
<BODY>
<TABLE BORDER="0" WIDTH="100%" CELLSPACING="3">
 <TR>
  <TD WIDTH="25%">
  <TD>
   <H1 ALIGN="CENTER">The Alberton <BR>
    Canyon Derailment</H1>
  <TD WIDTH="25%">
 <TR>
  <TD>
```

continues

Listing 16.5. continued

```
  <TD>
   <P ALIGN="CENTER">
    <IMG SRC="/mrfd/Alberton/albert8.jpg" WIDTH="300" HEIGHT="197" VSPACE="24">
   </P>
  <TD>
 <TR>
  <TD WIDTH="25%">
  <TD>
   <P><FONT SIZE="+3">I</FONT>n the early morning hours on Thursday, April 11,
    1996, an east-bound Montana Rail Link (MRL) train derailed in western
    Montana approximately two miles west of Alberton, and immediately adjacent
    to the Clarks Fork River and I-90. Eighteen cars derailed from the 71-car
    train, including six cars carrying hazardous materials. Four contained
    chlorine; one contained potassium cresylate; and the sixth contained
    sodium chlorate.</P>
   <P>To compound the problem further, another MRL train derailed near Noxon,
    Mont.,spilling 1,000 gallons of diesel fuel and overturning another
    chlorine car.  Theadditional confusion caused by the diversion and
    rerouting of resources slowed the initial response of personnel and
    equipment.</P>
   <P>One of the chlorine cars and the potassium cresylatecar were breached,
    resulting in the release of approximately 100,000 lbs.
    of deadlychlorine gas. The cloud drifted over the highway and through the
    town of Alberton forcing the evacuation of more than 1,000  people in a 100
    square-mile area. Half of these were kept out of their homes for 17 days.
    An estimated 350 the east. A lone fatality occurred when a transient riding
    in a box car walked into the cloud after surviving the crash.</P>
   <TD WIDTH="25%">
</TABLE>
</BODY>
</HTML>
```

FIGURE 16.5.
This formal design is for those moments in which dignity is everything. A large initial capital letter, central placement of text and image, and lots of space give this page its grace and simplicity.



As an added touch, you can increase the size of the first letter in the text using the `<FONT size>` tag. If you require additional text formatting and fonts, you should create the text portion in a program designed for the job, such as Macromedia FreeHand, and export it to a GIF or JPEG file. However, doing so will adversely affect page download time.

Page 1: A Template

With some of this basic layout information in hand, it's time to roll up your sleeves and get to work. Using the Missoula Rural Fire District site as an example, you'll begin with a template, which will become the basis for your other pages.

NOTE

An HTML template serves as a guide to other HTML pages, much like a pattern for sewing or tracing outlines. Templates are useful for maintaining the same design and image across multiple pages, without guesswork, by providing much of the underlying structure and formatting before you begin.

The Fire District uses its site to communicate a variety of information to the public, including safety tips and date-dependent news. The District also uses the site to develop general awareness about the District and its varying roles and responsibilities in the community. As such, the Web pages need to offer a professional attitude in a friendly matter.

For this purpose, we'll opt for a design that is a cross between the classic and technical layouts. It includes ample white space on the left side, interrupted only by hanging headlines and an appropriate image, combined with a bold banner across the top.

Construction of the page begins with the basics—a structure for the document (as shown in Listing 16.6).

Listing 16.6. The beginning of the template for a set of Web pages.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<TITLE>MRFD - a title</TITLE>
</HEAD>
<BODY bgcolor="#FFFFFF">
</BODY>
</HTML>
```

Structure does not a page make, but even a castle needs a foundation before the flags fly from the towers. In short, every Web page should start at the beginning and work up from there. In addition to the structure, we'll specify a default background color in the `<BODY>` tag using the RGB hexadecimal triplet for white. The placeholder `<TITLE>`, which contains the format for the Web page titles, is also added so we don't forget to add a title for other pages later.

TIP

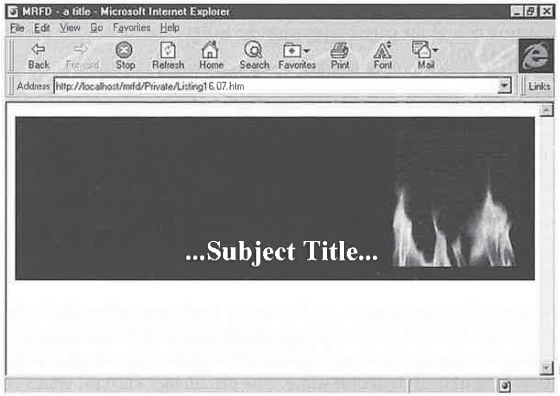
It's a good idea to begin a page title with a common identifier for your site. This makes it easier for the user to identify it if he or she includes a bookmark for a page on his or her home browser.

Now that the first bit of detail is completed, it's time to move to formatting for the top of the document. For consistency, we'll give every page the same banner across the top using a simple table with text and an image, as described in Chapter 10, "Creating Tables for Data and Page Layout." (See Listing 16.7 and Figure 16.6.)

Listing 16.7. The first part of the page includes a two-column table with one row.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<TITLE>MRFD - a title</TITLE>
</HEAD>
<BODY bgcolor="#FFFFFF">
<TABLE border="0" cellspacing="15" width="100%" bgcolor="#000000">
 <TR>
  <TD align="RIGHT" valign="BOTTOM" width="75%">
   <H1><FONT color="White">...Subject Title...</FONT></H1>
  <TD align="LEFT" valign="BOTTOM">
   <IMG src="/mrfd/images/fire2.jpg" HEIGHT="158" WIDTH="149">
</TABLE>
</BODY>
</HTML>
```

FIGURE 16.6.
The template as it appears so far, with a document title and heading for the top of the document.



You should note several things concerning the opening table tag. First, the borders are invisible (`border="0"`), which ensures that only the formatting is apparent to the user. The method of the formatting remains hidden. Next, the `cellspacing` attribute uses a larger-than-normal figure to help pad the contents of the two data cells. The table will always occupy 100 percent of the available window width with a black background (`bgcolor=#000000`).

The first cell has contents that are forced to the bottom and right margins and that occupy 75 percent of the available table width. Then, the page title is added in white to contrast against the black background. Because it's the main title, you use the `<H1>` tag.

The second cell's contents are aligned to the bottom and left margins. The `width` attribute is not used, because the browser will automatically assign it the remaining 25 percent of the table. Its content is a JPEG image with a specified height and width, as discussed in Chapter 12, "Adding Images to Your Web Page."

Because there's no way of telling where a document based on the template could appear, it uses an `src` URL, which is relative to the document root of the Web.

TIP

The height and width for the image are the same as its actual dimensions. These attributes help speed the page display time by alerting the browser to how much space is needed before the image begins to download.

TIP

The image of the flame could just as easily be an animated GIF or a Shockwave animation, as described in Chapter 13, "Integrating Multimedia and Other File Types." Place the special content in a common directory so the browser cache can recognize the file, no matter what page it appears on. This will dramatically speed download time for subsequent pages.

After the banner for the page is completed, insert the area where the content will appear. To do this, use another table with opposite column width settings to add interest and balance to the layout (as shown in Listing 16.8 and Figure 16.7).

Listing 16.8. Add the area for content using another table, with 25 percent of the width allowed for a headline and the balance for body text.

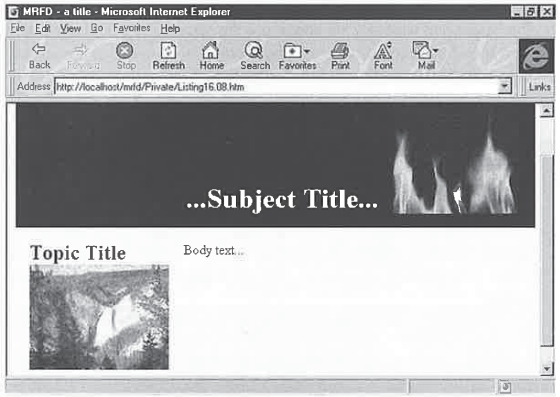
```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
```

continues

Listing 16.8. continued

```
<TITLE>MRFD - a title</TITLE>
</HEAD>
<BODY bgcolor="#FFFFFF">
<TABLE border="0" cellspacing="15" width="100%" bgcolor="#000000">
 <TR>
  <TD align="RIGHT" valign="BOTTOM" width="75%">
   <H1><FONT color="White">...Subject Title...</FONT></H1>
  <TD align="LEFT" valign="BOTTOM">
   <IMG src="/mrfd/images/fire2.jpg" HEIGHT="158" WIDTH="149">
</TABLE>
<TABLE BORDER="0" CELLSPACING="15" WIDTH="100%">
 <TR>
  <TD WIDTH="25%" VALIGN="TOP">
   <H2>Topic Title
     <IMG SRC="/ClipArt/Earth/Landscp/Wildrnss.jpg" WIDTH="162" HEIGHT="122">
   </H2>
  <TD VALIGN="TOP">
   <P>Body text...</P>
</TABLE>
</BODY>
</HTML>
```

FIGURE 16.7.
With the two main elements in place, the Web page template begins to take shape. Notice the placeholder text, which will be replaced for each Web page as it is developed from the template.

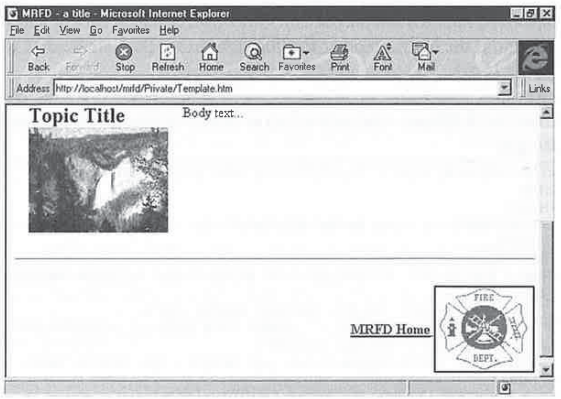


Now, the structure tags are in place, the page has a banner heading, and the table is formatted to receive content. The last step is to add a few details to finish the job, including a horizontal rule at the bottom of the page and a link back to the home page. With all of these pieces in place, the completed template file appears in Listing 16.9 and Figure 16.8.

Listing 16.9. The `template.htm` file includes all of the components for the other Fire District Web pages.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
 <TITLE>MRFD - a title</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<TABLE BORDER="0" CELLSPACING="15" WIDTH="100%" BGCOLOR="#000000">
 <TR>
  <TD ALIGN="RIGHT" VALIGN="BOTTOM" WIDTH="75%">
   <H1><FONT COLOR="White">...Subject Title...</FONT></H1>
  <TD ALIGN="LEFT" VALIGN="BOTTOM">
   <H1><IMG SRC="/mrfd/images/fire2.jpg" HEIGHT="158" WIDTH="149"></H1>
</TABLE>
<TABLE BORDER="0" CELLSPACING="15" WIDTH="100%">
 <TR>
  <TD WIDTH="25%" VALIGN="TOP">
   <H2>Topic Title
     <IMG SRC="/ClipArt/Earth/Landscp/Wildrnss.jpg" WIDTH="162" HEIGHT="122">
   </H2>
  <TD VALIGN="TOP">
   <P>Body text...</P>
</TABLE>
<HR>
<P ALIGN="RIGHT">
<A HREF="/mrfd/index.htm"><STRONG>MRFD Home</STRONG>
 <IMG SRC="/mrfd/Images/cross.gif" WIDTH="113" HEIGHT="97" ALIGN="MIDDLE">
</A></P>
</BODY>
</HTML>
```

FIGURE 16.8.
The completed file, `template.htm`, illustrates the basic appearance of any Web page created for the Fire District. Through the use of invisible borders for the table, advanced formatting techniques are implemented without distracting the user.



NOTE

If a picture is worth a thousand words, it can seem to take at least a thousand minutes to load. For that reason, most of the pages within the Fire District site include only three relatively small images—one in the banner, one under the document title, and one for the link to the home page.

Because the banner image and home page image are the same on virtually every page, the user's browser should maintain its respective files in a cache that will further speed download time. The only new file the browser needs to download for each page is the one under the document title.

Why all this emphasis on limiting images? Because a lot of ISDN and 28.8Kbps modems are on the market, as are a lot of 14.4Kbps and 9600 modems. This limit is an attempt to be sensitive to the download time for all users while still including enough images to make the pages interesting.

Save the template as `template.htm`. Every time a new page is needed, you can place a copy of `template.htm` in the appropriate directory and rename it. Then, you can open it for editing and replace the placeholder contents with the real thing. Because it's a utility file not meant for display to users, it is kept on the local drive and not on the Web site.

Creating a Real Page from the Template

After the template is created, we'll make copies that are then modified to create the real pages for our users. The first copy is destined to become one of the District's safety pages. First, we'll rename it, and then add the content. Other than changing the title and adding the appropriate content in the body, no other modifications to the page are needed. (See Listing 16.10 and Figure 16.9.)

Listing 16.10. This page was based entirely on `template.htm`. Only the title and body content was changed.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
 <TITLE>MRFD - Living in the Wildland/Urban Interface</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<TABLE BORDER="0" CELLSPACING="15" WIDTH="100%" BGCOLOR="#000000">
 <TR>
  <TD ALIGN="RIGHT" VALIGN="BOTTOM" WIDTH="75%">
   <H1><FONT COLOR="White">...Summer Fire Safety...</FONT></H1>
  <TD ALIGN="LEFT" VALIGN="BOTTOM">
   <IMG SRC="/mrfd/images/fire2.jpg" HEIGHT="158" WIDTH="149">
</TABLE>
<TABLE BORDER="0" CELLSPACING="15" WIDTH="100%">
 <TR>
  <TD WIDTH="25%" VALIGN="TOP">
```

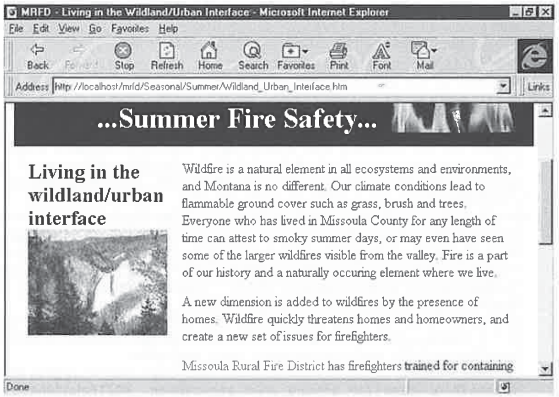


```
   <H2>Living in the wildland/urban interface
   <IMG SRC="/mrfd/images/Wildrnss.jpg" WIDTH="162" HEIGHT="122"></H2>
  <TD VALIGN="TOP">
   <P>Wildfire is a natural element in all ecosystems and environments, and
    Montana is no different. Our climate conditions lead to flammable ground
    cover such as grass, brush and trees. Everyone who has lived in Missoula
    County for any length of time can attest to smoky summer days, or may even
    have seen some of the larger wildfires visible from the valley. Fire is
    a part of our history and a naturally occuring element where we live.</P>
   <P>A new dimension is added to wildfires by the presence of homes. Wildfire
    quickly threatens homes and homeowners, and create a new set of issues for
    firefighters.</P>
   <P><FONT COLOR="Maroon">Missoula Rural Fire District</FONT> has firefighters
    trained for containing fires in the wildland/urban interface. But compared
    to a wildfire, they are very limited in manpower and equipment, and in the
    distances and terrain they protect. The first homes to burn in the recent
    Pattee Canyon fire were lost while firefighters were still en route to the
    scene.</P>
   <P>If you're in an area that has a potential for wildfire, you can call
    for a representative from <FONT COLOR="Maroon">Missoula Rural Fire
    District</FONT> to come to your home and make
    <A HREF="/mrfd/Seasonal/Summer/Wildfire_Tips.htm">suggestions for creating
    defensible space</A> around your structures. <CITE>Defensible space</CITE>
    is an <DFN>area of reduced fuel</DFN> which gives your home greater odds
    of surviving a fire. This is a service provided free of charge to district
    homeowners. For more information, call 549-6172 or
    <A HREF="mailto:mrfd@montana.com">send us a note</A>.</P>

</TABLE>
<HR>
<P ALIGN="RIGHT">
<A HREF="/mrfd/index.htm"><STRONG>MRFD Home</STRONG>
 <IMG SRC="/mrfd/Images/cross.gif" WIDTH="113" HEIGHT="97" ALIGN="MIDDLE">
</A></P>
</BODY>
</HTML>
```

Several features were included within the content. First, any mention of Missoula Rural Fire District was highlighted in maroon to emphasize who was providing the information without being intrusive. A color such as red or pink would stand out more, but would also interfere with the user trying to read. This use of the `<FONT>` tag is covered in Chapter 8, "Text Alignment and Formatting."

Next, two hyperlinks were added within the text, following the syntax discussed in Chapter 11, "Linking Documents and Images." The first was a link to a related page, `/mrfd/Seasonal/Summer/Wildfire_Tips.htm`. If the user wanted more information on this topic, he or she could click the hyperlink. The second was a link to send e-mail, `mailto:mrfd@montana.com`. If a user wanted to provide feedback, he or she had the option of doing so immediately rather than looking for a feedback page elsewhere on the site.

FIGURE 16.9.
The finished product, which was created using the template as its starting point, took only a few minutes to create.



Another feature used the logical tags for citations and definitions. The <CITE> tag was used to highlight the term *defensible space*, and the <DFN> tag was used to mark the definition within the same sentence. In most browsers, this would probably appear the same (italics). The logical tags indicate usage if the page is edited or changed at a later time.

This is a simple page, but it represents most of the content on this site. The total time used to create it—including copying and renaming the template, typing the text, and finding an appropriate image to use under the document title—was 10 minutes.

A Home Page from the Template

Creating a home page from the template is similar to creating the standard page. Some hyperlinks within the document are added, plus a graphic image for a bullet. The hyperlink at the end of the template is replaced with author and copyright information.

The District home page (shown in Listing 16.11 and Figure 16.10) is basically the same as any other page but with a few additional features. First is an unordered list that uses graphical bullets instead of the <UL> tag. This is the only additional graphic used on this page beyond the three called for in the template. It's a small image that you can download quickly and that doesn't seriously interfere with download time.

Listing 16.11. The home page is also created from the template with some additional minor style changes.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
 <TITLE>Missoula Rural Fire District — Home Page</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<TABLE BORDER="0" CELLSPACING="15" CELLPADDING="1"
➥WIDTH="100%" BGCOLOR="#000000">
 <TR>
  <TD ALIGN="RIGHT" VALIGN="BOTTOM" WIDTH="75%">
   <H1><FONT COLOR="White">...Missoula Rural Fire District<BR>
    ...<A HREF="#NAME">Point of Origin</A></FONT></H1>
  <TD ALIGN="LEFT" VALIGN="BOTTOM">
   <IMG SRC="/mrfd/images/fire2.jpg" HEIGHT="158" WIDTH="149">
</TABLE>
<TABLE BORDER="0" CELLSPACING="10" CELLPADDING="5" WIDTH="100%">
 <TR>
  <TD WIDTH="25%" VALIGN="TOP" ALIGN="CENTER">
   <H2 ALIGN="CENTER"><A NAME="TOP"></A>Missoula Rural
    <IMG SRC="Images/MRFD-BADGE.JPG" ALIGN="MIDDLE" WIDTH="127" HEIGHT="149">
    Fire District</H2>
   <P> </P>
   <P ALIGN="LEFT"><B> <I>Point of origin</I></B><BR>
        <I>Where a fire begins</I></P>
  <TD WIDTH="75%" VALIGN="TOP">
   <H3>
    <IMG SRC="Images/BL-HELMT.gif" WIDTH="61" HEIGHT="59" ALIGN="MIDDLE">
    <A HREF="#SPARKS">What's New</A></H3>
   <UL>
    <LI>
     <P><A HREF="#SPARKS1"><STRONG>Burning season open</STRONG></A></P>
    <LI>
     <P><A HREF="#SPARKS2"><STRONG>Flood plain information</STRONG></A></P>
   </UL>
   <H3><IMG SRC="Images/BL-HELMT.gif" WIDTH="61" HEIGHT="59" ALIGN="MIDDLE">
    <A HREF="/mrfd/Seasonal/Summer/index.htm">Summer safety</A></H3>
   <BLOCKQUOTE>It may not feel like it yet, but the summer season is almost upon
    us. Here are some things to  keep in mind to keep you safe.</BLOCKQUOTE>
   <H3><IMG SRC="Images/BL-HELMT.gif" WIDTH="61" HEIGHT="59" ALIGN="MIDDLE">
    <A HREF="General_Information/Property_Taxes.htm">How
    we spend the taxes you pay</A></H3>
   <BLOCKQUOTE>Every year, a part of your property taxes goes towards
    fire protection in the Missoula Rural Fire District. Here's how that
    amount is determined.</BLOCKQUOTE>
   <H3><IMG SRC="Images/BL-HELMT.gif" WIDTH="61" HEIGHT="59" ALIGN="MIDDLE">
    <A HREF="Alberton/">The Alberton Canyon Derailment</A></H3>
   <BLOCKQUOTE><FONT COLOR="Maroon">Missoula Rural Fire District
    </FONT>was a key player in this incident. The Regional Hazardous
    Materials Response Team was involved during the entire incident. MRFD
    also supplied the Operations Chief, Public Information Officer, a Liaison
    Officer and many other positions during this incident.</BLOCKQUOTE>

   <H3><IMG SRC="Images/BL-HELMT.gif" WIDTH="61" HEIGHT="59" ALIGN="MIDDLE">
    <A HREF="General_Information/Staff_Directory.htm">Who</A> and
    <A HREF="General_Information/Station_Directory.htm">where</A> we are</H3>
   <BLOCKQUOTE>MRFD covers a lot of ground, so we've provided a
    <A HREF="General_Information/Station_Directory.htm">station</A>
    and <A HREF="General_Information/Staff_Directory.htm">staff</A>
    directory to help figure out who's who and what's what.
    </BLOCKQUOTE>
   <HR>
<TR>
```

continues

Listing 16.11. continued

```
  <TD>
  <TD>
   <H3><IMG SRC="Images/BL-HELMT.gif" WIDTH="61" HEIGHT="59" ALIGN="MIDDLE">
     <A NAME="SPARKS"></A>Sparks</H3>
   <H4><A NAME="SPARKS1"></A>Burning permits available</H4>
   <BLOCKQUOTE>The Spring burning season is now open. Stop by any
    <A HREF="General_Information/Station_Directory.htm">fire
    station</A> between 8 a.m. and 5 p.m. to pick up a burning permit
    and a copy of the rules to keep your flames legal. The permit and
    advice are free of charge.</BLOCKQUOTE>
   <H4><A NAME="SPARKS2"></A>Flood plain maps posted</H4>
   <BLOCKQUOTE>Each <A HREF="General_Information/Station_Directory.htm">fire
    station</A> in the District now has a copy of the Missoula County flood
    plain maps posted for public review. If you want to know  where you'll be
    standing when the water comes, come by and take a look.</BLOCKQUOTE>
    </TABLE>
<HR SIZE="4" noshade>
<P><CITE><IMG SRC="Images/cross.gif" WIDTH="113" HEIGHT="97" ALIGN="LEFT">
 &copy;1996, 1997 Missoula Rural Fire District, Missoula, Montana.</CITE>
 <BR><CITE>Page layout and construction by Rick Darnell and Dave Herzberg.
     </CITE></P>
<ADDRESS>Send comments to <A HREF="mailto:mrfd@montana.com">MRFD</A>.</ADDRESS>
<HR>
<P ALIGN="CENTER"><EM>For more information about the fire service and
 other departments<BR> around the country, be sure to stop by</EM><BR>
  <A HREF="http://www.firelink.com/">
   <IMG SRC="Images/SMALLFIRELINK.GIF" WIDTH="140" HEIGHT="43">
  </A></P>
</BODY>
</HTML>
```

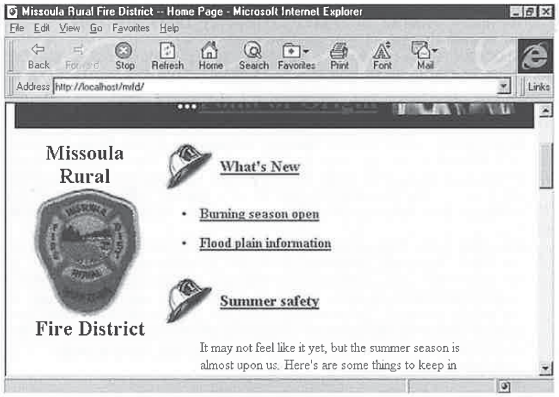


FIGURE 16.10.
The Fire District home page also uses the template as its base. To increase its color, it includes graphical bullets for each of the list items.

Each list item is a link to another page on the site and is followed by a brief summary paragraph marked with `<BLOCKQUOTE>`. This indents the text and gives further emphasis to the bullet and link.

The home page also includes a news briefs section, a quick summary of items of interest that is individually hyperlinked using anchor tags (`<A name>`) as the destination. Because this is separate from the main section of the home page, an additional row is added to contain the briefs.

The last item added to the page is copyright and author information. This is added in lieu of the hyperlink to the home page and provides the legal information that most documents should include.

A Page with a List

The next page created using `template.htm` is a directory of the fire stations that are a part of the District. After copying the template, the banner headline and page title are both changed and the lists are added for the body text, which is illustrated in Chapter 9, "Using Lists to Organize Information." (See Listing 16.12 and Figure 16.11.) A set of nested lists is also added.

Listing 16.12. The outer list in this set is a definition list that includes an ordered list as a part of each definition (<DD>).

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
 <TITLE>MRFD — Station Directory</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<TABLE BORDER="0" CELLSPACING="15" CELLPADDING="1"
➥WIDTH="100%" BGCOLOR="#000000">
 <TR>
  <TD ALIGN="RIGHT" VALIGN="BOTTOM" WIDTH="75%">
   <H1><FONT COLOR="White">...Directory...</FONT></H1>
  <TD ALIGN="LEFT" VALIGN="BOTTOM">
   <IMG SRC="/mrfd/images/fire2.jpg" HEIGHT="158" WIDTH="149">
</TABLE>
<TABLE BORDER="0" CELLSPACING="10" CELLPADDING="5" WIDTH="100%">
 <TR>
  <TD WIDTH="25%" VALIGN="TOP">
   <H2>Missoula Rural Fire District Station Directory</H2>
  <TD WIDTH="75%" VALIGN="TOP">
   <DL>
    <DT><STRONG>Station 1</STRONG> (Administration and Maintenance)
    <DD><EM>2521 South Ave. West, Missoula<BR>
      (406) 549-6174</EM>
     <OL>
      <LI>Engines 311, 312, 316
      <LI>Water Tender 317
      <LI>Staff vehicles 301, 302, 303, 305
     </OL>
```

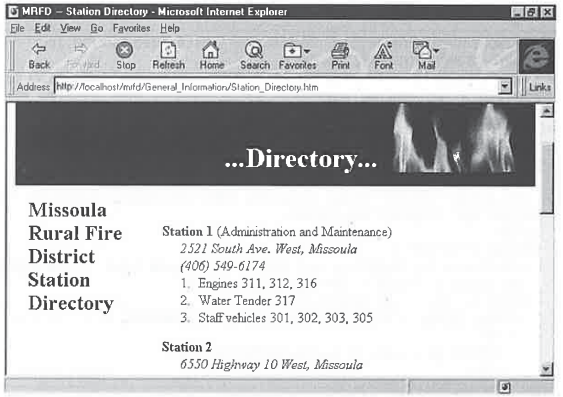
continues

Listing 16.12. continued

```
    <DT><STRONG>Station 2</STRONG>
    <DD><EM>6550 Highway 10 West, Missoula<BR>
      (406) 549-3601</EM>
     <OL>
      <LI>Engines 321, 326
      <LI>HazMat Response 328
      <LI>Staff vehicle 304
     </OL>
    </DD>

    <DT><STRONG>Station 3</STRONG>
    <DD><P><EM>Closed June 1995</EM></P>

    <DT><STRONG>Station 4</STRONG>
    <DD><EM>9480 Highway 10 East, Bonner<BR>
      (406) 258-6061</EM>
     <OL>
      <LI>Engines 341, 346
      <LI>Water Tender 347
      <LI>Truck 348
     </OL>

    <DT><STRONG>Station 5</STRONG>
    <DD><EM>12221 Highway 93 South, Lolo<BR>
      (406) 273-2551</EM>
     <OL>
      <LI>Engines 351, 356
      <LI>Water Tender 357
     </OL>
    </DD>

    <DT><STRONG>Station 6</STRONG>
    <DD><EM>8455 Mullan Road, Missoula<BR>
      (406) 542-0366</EM>
     <OL>
      <LI>Engines 361, 366
     </OL>
   </DL>
</TABLE>
<P ALIGN="RIGHT"><A HREF="/mrfd/index.htm">
<STRONG>MRFD Home</STRONG>
<IMG SRC="/mrfd/Images/cross.gif" WIDTH="113" HEIGHT="97" ALIGN="MIDDLE">
</A></P>
</BODY>
</HTML>
```

Each list entry consists of two parts—the name of the station and its address and assigned apparatus. The only list available in HTML that serves a term-definition relationship is the definition list. The list of apparatus is created using a numbered list within the definition.

FIGURE 16.11.
Several directory pages are similar to this one. Where additional information is needed for each list entry, the definition list tags are used.



A Feedback Form

Last on our list of projects for the MRFD site is a guest book. We're not looking for anyone's life story with this feature—just a few basic demographic bits of information. If more information is needed, we can always add more fields, as demonstrated in Chapter 15, "Building and Using HTML Forms."

This page continues to use tables to control its formatting, including a table within a table. This occurs in the main body of the document, where the form contains a table to control the formatting of its elements. (See Listing 16.13.)

Listing 16.13. The guest book form page gathers basic information from the user and retains formatting using tables.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 3.2//EN">
<HTML>
<HEAD>
 <TITLE>MRFD - Guest Book</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<TABLE BORDER="0" CELLSPACING="0" WIDTH="100%"
➥BGCOLOR="#000000" CELLPADDING="10">
 <TR>
  <TD ALIGN="RIGHT" VALIGN="BOTTOM">
   <H1><FONT COLOR="White">...Thanks for visiting...</FONT></H1>
  </TD>
  <TD ALIGN="LEFT" VALIGN="BOTTOM" WIDTH="110">
   <P><BR>
```

continues

Listing 16.13. continued

```
    <IMG SRC="/mrfd/Images/pointsource-posterized_92x139.jpg"
➥HEIGHT="139" WIDTH="92"></P>
  </TD>
  <TD WIDTH="25"> </TD>
 </TR>
</TABLE>
<TABLE BORDER="0" CELLSPACING="15" WIDTH="100%">
 <TR>
  <TD WIDTH="25%" VALIGN="TOP">
   <H2 ALIGN="CENTER">Guest Book<IMG SRC="/FireDept/Images/fd-foot.jpg"
➥WIDTH="128" HEIGHT="205"></H2>
  </TD>
  <TD VALIGN="TOP">
   <H3>Thanks for visiting our Web site. Please take  a few minutes to tell
    us about yourself.</H3>
   <FORM ACTION="/cgi/guestbook.exe">
    <TABLE BORDER="0" CELLSPACING="2" CELLPADDING="1">
     <TR>
      <TD>Do you live in Missoula County?</TD>
      <TD>
       <P ALIGN="LEFT">
        <INPUT TYPE="checkbox" NAME="CheckBox" VALUE="CheckBox">
        Yes</P>
      </TD>
     </TR>
     <TR>
      <TD>Are you a member of any fire service agency?</TD>
      <TD>
       <P ALIGN="LEFT">
        <INPUT TYPE="checkbox" NAME="CheckBox" VALUE="CheckBox">
        Yes</P>
      </TD>
     </TR>
     <TR>
      <TD>If so, paid or volunteer?</TD>
      <TD>
       <P ALIGN="LEFT">
        <INPUT TYPE="radio" NAME="PaidVolunteer" VALUE="Paid">
        Paid  <BR>
        <INPUT TYPE="radio" NAME="PaidVolunteer" VALUE="Volunteer">
        Volunteer</P>
      </TD>
     </TR>
     <TR>
      <TD>How did you find us?</TD>
      <TD>
       <SELECT NAME="Where">
        <OPTION VALUE="FL">FireLink</OPTION>
        <OPTION VALUE="MC">Missoula County</OPTION>
        <OPTION VALUE="MT">Montana Territories</OPTION>
        <OPTION VALUE="YL">Yahoo/Lycos/Etc.</OPTION>
        <OPTION VALUE="OT">Other</OPTION>
       </SELECT>
      </TD>
     </TR>
     <TR>
```



```
      <TD VALIGN="TOP">Anything you'd like to tell us? (What you'd like
       to see<BR>on the site, what you thought of our pages, etc.)
      </TD>
      <TD>
       <TEXTAREA COLS="20" NAME="comments" ROWS="10"></TEXTAREA></TD>
     </TR>
    </TABLE>
    <P ALIGN="CENTER">
     <INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Sign the Book">
     <INPUT TYPE="RESET" NAME="Reset" VALUE="Never mind...">
    </P>
   </FORM>
  </TD>
 </TR>
</TABLE>
<HR SIZE="8" NOSHADE>
<P ALIGN="RIGHT">
 <A HREF="/mrfd/index.htm"><STRONG>MRFD Home</STRONG>
 <IMG SRC="/mrfd/Images/cross.gif" WIDTH="113" HEIGHT="97" ALIGN="MIDDLE">
 </A></P>
</BODY>
</HTML>
```

Like the parent table, the form's table has the borders suppressed so only the formatting is seen, not the mechanism for creating it. Also, note the use of alternative labels for the submit and reset buttons. Although *Submit* and *Reset* are fine and acceptable names, sometimes it's nicer to include something a little more personal. However, the buttons still work the same.

Summary

This chapter has given you a look into the way one organization uses HTML 3.2 to build its Web pages. The process began with experimenting with the four basic design elements—headlines, text, images, and space. These elements led to a simple design that fit the needs of the Fire District and became the basis of a Web page template.

From there, various HTML 3.2 elements were added to the pages, including hyperlinks, font sizes and colors, lists, and inline images. The result is a uniform image for the Web site that requires little time to implement. This feature is especially useful because this site is maintained by people with other demands on their time.

It is possible to use combinations of graphics, tables, and frames to build more elaborate and fancy page designs, but you should get a feel from these examples for what is possible by remaining within the constraints of the standard HTML elements. Chapter 18, "Creating Sophisticated Layouts with Frames and Layers," covers extended HTML features beyond what I've discussed in this section. Chapter 19, "Introducing Cascading Style Sheets," focuses on design, showing you how to further use HTML's capabilities to deliver your message to an eager audience.

PART III

Extending HTML 3.2

IN THIS PART

CHAPTER 17

Introducing HTML 4.0

by Bob Correll

IN THIS CHAPTER

To say that HTML is evolving is undeniably true. We've had about a year to enjoy the new capabilities HTML 3.2 gave us as Web developers and gnash our teeth at what it couldn't do. Netscape and Microsoft each upped the ante with their third generation of Web browsers in the summer of 1996, seeking to implement what was then the draft version of HTML 3.2. It wasn't until January 1997 that the specification for HTML 3.2 was approved by the World Wide Web Consortium (W3C), and true to the fast pace of free-market competition, the fourth generation of browsers and the next HTML specification is here.

The fact that Netscape and Microsoft are battling for control of the browser market has tremendous influence over the shape of HTML and the rapidity of change in the specification, a standard that has been trailing the browsers in the race to the market. Get used to it if you haven't already. Without browsers, we would have no use for HTML, and despite the feeling that sometimes the cart is preceding the horse, I would argue that this external contest is, on the whole, beneficial.

Enter HTML 4.0. This chapter is an overview of the latest developments in the HTML specification from the W3C and an entry point to the other chapters that cover these items in more depth. Some of the items I introduce will undoubtedly be familiar to you, and perhaps you are already using some of them. In these areas, the W3C is catching up with HTML elements already in widespread use and making the *de facto* standard official. Other areas, such as the Object Model, are more forward-looking and lay the groundwork for the future of HTML. A full understanding of these areas is critical for you to get the most out of HTML as you develop your Web pages and sites.

What Is HTML 4.0?

HTML 4.0 is the code name for a collection of material under review by the World Wide Web Consortium HTML Working Group for potential incorporation into the next version of HTML, most likely version 4.0. This material has been released to the public, and much discussion will precede the eventual acceptance or rejection of each proposal (or part thereof).

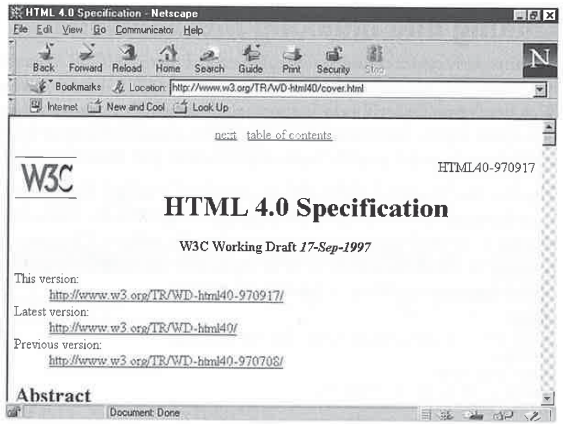
NOTE

This chapter relies heavily on the draft HTML 4.0 material from the W3C, and therefore some of the information is subject to change. You should regularly visit the W3C site at

`http://www.w3.org/TR/WD-html40/cover.html`

This site, shown in Figure 17.1, keeps you abreast of developments and modifications.

FIGURE 17.1.
You should become familiar with the Cougar page at the World Wide Web Consortium.



The 10 documents, each with a different focus, that were under consideration by the W3C as Cougar are summarized here:

- Creating "index-friendly" Web pages
- Fully extending HTML support for style sheets
- Fully extending HTML support to client-side scripting lanuages
- Standardizing frame usage
- Enhancing forms
- Inserting objects into HTML
- Enlarging supported character sets
- Allowing file uploads
- Enhancing tables
- Internationalization

As was the case with HTML 3.2, several months may go by before the process of reviewing and debating the different proposals culminates in an official W3C Recommendation. As this discussion progresses, versions 4 of Microsoft Internet Explorer and Netscape Navigator (in the form of Netscape Communicator) will be released—before the standard has been set! This will result in a period of time when neither major browser meets the precise requirements of HTML 4.0 but implements only select portions, some of which probably will not be included in the final HTML standard.

Linking and Indexing Mechanisms

The first draft under consideration, Hypertext Links in HTML, describes the current use of hypertext links and resource descriptions used in HTML 4.0. It offers several techniques for Web developers that will help indexing engines better catalog your site. I won't go into the descriptive aspects here because these are fully covered in Chapter 11, "Linking Documents and Images," and Chapter 7, "Structural Elements and Their Usage."

The first two recommended methods for making your site friendlier to indexing engines involve language. Given the internationalization of the Web, it is becoming more important that you know in which language a page was originally written. Therefore, you should define the language in the document HEAD, or as required in the BODY element if you depart from your default language, as shown in this snippet of code:

```
<HEAD>
<META HTTP-EQUIV=Content-Language CONTENT=en>
</HEAD>
<BODY LANG=en>
[content of Body is English]
<SPAN LANG=fr>
[content to be displayed in French]
</SPAN>
[body language reverts back to English]
</BODY>
```

You should also specify any language variants of the document using the LINK element to refer to your different translations. This allows search engines to offer the results in the preferred language of the user if a translation exists.

NOTE

Internationalization is a major theme of HTML 4.0. More than ever before, the fact that it is a *World Wide* Web is being recognized, and HTML 4.0 has language-identifying attributes sprinkled throughout.

The remaining recommendations suggest methods to construct your Web page in a way that improves indexing by search engines. As you might know, searching the Web can be a daunting task, and many vendors are expending tremendous amounts of energy attempting to catalog and index the vast number of Web pages currently online. You can help ensure that your site stands a chance of being properly indexed by using the META element with descriptive keywords, providing a logical content description, and specifying which areas of your site can be indexed.

Support for Style Sheets

Style sheets represent a major advancement in the ability of HTML authors to control the appearance and layout of their work. This change has been driven in part by the increasing importance of the Web to business and entertainment, whose content developers were not satisfied with the standard heading tags and rudimentary layout possibilities that HTML offered. With style sheets, you can specify different font faces, sizes, and colors; control your margins; insert padding; and much more. Some of these effects are shown in Figure 17.2, the W3C informational Web page on style.

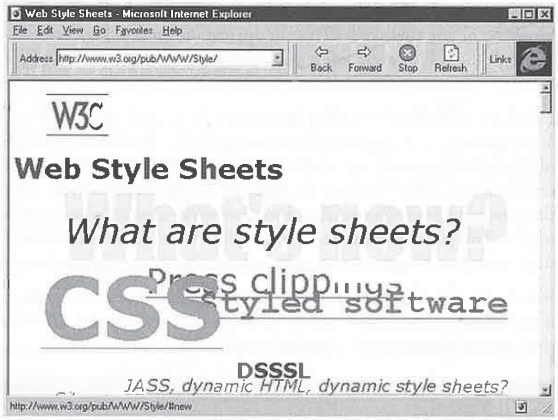


FIGURE 17.2.
You can create appealing visual effects using style sheets. This Web page is from the W3C.

On December 17, 1996, the W3C officially released its recommendation on style sheets, entitled "Cascading Style Sheets, level 1." HTML 3.2 made provisions for the introduction of style sheets with the STYLE tag but did not flesh out the necessary HTML to completely integrate them into the standard. HTML 4.0 addresses this fact and more fully describes how to incorporate style sheets into your HTML documents.

Style sheets are more fully covered in Chapter 19, "Introducing Cascading Style Sheets," Chapter 20, "Cascading Style Sheet Usage," and Chapter 21, "JavaScript Style Sheets and Other Alternatives to CSS."

Conceptual Framework for Style Sheets

Several rules of thumb drive the implementation of style sheets in HTML. The first is a commitment to support the flexible placement of style information. Instead of being forced to define a style in a single manner and in one designated location, you have the freedom to choose from several methods depending on the circumstances and your inclinations. You might want to include style information in a variety of places, either external to the HTML document or within it. The freedom to refer to an external style sheet, include style information in the HEAD element of a document, or place it inline ensures that you can find a method to suit your needs.

Second, HTML 4.0 does not specify a single style language to support. You can use cascading style sheets, level 1 (which has the recommendation of the W3C and is supported by both major browsers) or any alternative. This has opened the door to JavaScript-based style sheets as well as a few others. However, before you decide that "Joe's Friendly Style Sheets, level 42" are the ones you like best, you should ensure that your target audience's browser can interpret them.

Third, the concept of cascading style sheets (in general, as opposed to the specific Cascading Style Sheets, level 1 implementation) is supported. This allows style elements from a number of external sources to be included—thus, the term *cascading*. By using cascading style sheets of one form or another, you can put together multiple styles of your choosing, independent of the content of your Web page, and you can apply them depending upon design you are looking for. This permits you to reuse your styles with a minimum of fuss. You should refer to the documentation of your chosen style, because not all style languages support this usage.

Fourth, you can define a broad category of media (such as a PC monitor, WebTV, print-based browser, or speech-based browser) to which your style applies. This warns certain users that they are unable to support a particular style.

NOTE

This is another major theme of HTML 4.0—improving support for nonvisual browsers or those with capabilities different from most standard PC monitors.

Finally, multiple styles can be supported to give users the flexibility of choosing one that suits their needs. This permits for the "graceful degradation" of your style depending on the limitations of a diverse audience. A person with limited vision might be able to choose an alternate style that you have provided with larger fonts to improved legibility.

Now that you've seen the conceptual framework, let's look briefly at a few of the proposed elements that make this possible.

The LINK Element

Using the LINK element, you can refer to a style sheet that is external to the HTML document. You can reference multiple external style sheets if necessary (which is called cascading), with the last LINK taking precedence over those before it.

The syntax for using LINK is

```
<LINK attributes>
```

No closing tag is required. The LINK element supports the following six attributes:

- href provides the URL for the linked style sheet.
- title is used to group various LINK and STYLE elements together under a common name such as "Corporate Standard."
- rel stands for *relationship value* between the LINK element and the external style sheet and must be included for your external style to be recognized as a style sheet. The values are stylesheet and alternate stylesheet.
- rev defines the relationship opposite to REL, which would be from the style sheet to the LINK element. This is not necessary if you are linking to an external style.
- type specifies the Internet Media type, allowing users to disregard if they do not support your media type.
- media defines the medium to which your style sheet applies. The media attributes are print, screen, projection, braille, aural, and all. Multiple options are permissible when separated by commas.

Suppose that you have multiple styles you want to incorporate into your document. Using the LINK element to reference them would look like the following:

```
<LINK REL=stylesheet MEDIA=aural HREF="soundsgood.css">
<LINK REL=stylesheet MEDIA=braille HREF="feelsgood.css">
<LINK REL=stylesheet MEDIA=screen HREF="looksgood.css">
<LINK REL=stylesheet MEDIA=all HREF="allgood.css">
```

Depending on the media of the target browser, one or more of the style sheets would be applied.

The STYLE Element

The STYLE element enables you to include your style in the actual HTML document. The element must be in the document HEAD, and multiple declarations are allowed. The syntax is

```
<STYLE attributes>
[style data]
</STYLE>
```

A closing tag is required. The STYLE element has only the following three attributes:

- type is the Internet Media type, such as "text/css".
- title is similar to the title attribute for the LINK element. This allows you to group your styles together under a common name.
- media allows you to identify the media for which your style is designed. This attribute can take the same values that were used in the LINK element.

If you want to define a style to apply to your different headings, it would look like this:

```
<STYLE TYPE="text/css">
     H1 {color: red }
     H2 {color: blue}
     H3 {color: black}
</STYLE>
```

Generic Attributes

In order to define a style across a document, a method for identifying each style-capable element must be created. This allows you to apply your style to a defined object in the page. The following generic attributes have been proposed for most elements in HTML:

- id defines a unique identifier that applies to the entire document.
- class allows you to define a class of elements that will be formatted as you define in your style declaration.
- style provides rendering information that is specific to the element to which it is attributed.

The SPAN and DIV Elements

What if you want to change the style of some text in the middle of a paragraph? The text is not a complete element and would not have a unique ID. Therefore, the SPAN element has been proposed to allow you to apply a style to text that is otherwise not an element. After defining the SPAN style in the STYLE element, you can use SPAN wherever you would use the <EM> tag.

The DIV element is proposed to handle situations when you have multiple elements to which you want to apply a style. It is used wherever the <P> tag is used, and it can be nested.

Support for Scripting

Although you might have used JavaScript and VBScript for some time, you might be interested to know that the SCRIPT element in HTML 3.2 was simply a placeholder reserved for future use. Your ability to incorporate these technologies into your Web pages has been at the discretion of the large browser vendors—Netscape and Microsoft. This has caused considerable trouble in terms of standardization, as each company has attempted to evangelize its chosen language. You now have a more fully developed scripting interface into HTML in which you can insert client-side scripting.

This is certainly good news, but perhaps the real significance is the fact that scripting will be the glue that holds many of the proposed elements of HTML 4.0 together. For example, scripting will be able to manipulate style sheets, and a large part of what is called Dynamic HTML relies upon scripting to provide the dynamism.

Because HTML does not define a standard scripting language, implementation across browsers between the differing scripts might not be uniform, but it appears that JavaScript is currently the most widely supported script.

NOTE

The SCRIPT element illustrates another focus of HTML 4.0—building the framework for the inclusion of a language or object while remaining neutral to the implementation. This neutrality has its good and bad points. It allows you the flexibility to choose a language you desire, but it does not ensure that all browsers will support it.

Knowing how to use the SCRIPT tag and taking advantage of these events is dependent on your knowledge of a chosen scripting language. Therefore, I encourage you to read Chapter 28, "Integrating JavaScript and HTML," and Chapter 30, "Integrating ActiveX and VBScript," for further details.

The SCRIPT Element

Multiple instances of the SCRIPT element can be located in both the document HEAD or BODY. The SCRIPT element has the following syntax:

```
<SCRIPT attributes>
[script and HTML content]
</SCRIPT>
```

Here are the available attributes:

- type is the MIME type that specifies the scripting language, such as "text/javascript" or "text/vbscript".
- language is the name of the scripting language, which is deprecated in favor of the type attribute.
- src allows you to give a URL for access to external script.

NOTE

Accessing external scripts requires you to set up a MIME attribute for the type of document that contains the script.

Intrinsic Events

Intrinsic events are easy to define. A user has done something that you can track and have the Web page respond to through the appropriate script. Having the capability to track these events allows you to build dynamic Web pages. The current list includes the following events:

- `onLoad` occurs when the page has finished loading or all frames defined within a frameset have loaded. You can use this attribute only within the `BODY` or `FRAMESET`.
- `onUnload` occurs when a user exits the document. This attribute is available only within the `BODY` or `FRAMESET` elements.
- `onClick` occurs when the user clicks on an anchor or form field. The anchor refers to a hypertext link and the form field consists of buttons, checkboxes, radio buttons, reset buttons, and submit buttons.
- `onMouseOver` occurs when the mouse is moved over an anchor or `TEXTAREA` element.
- `onMouseOut` occurs, conversely, when the mouse has moved out of an anchor or `TEXTAREA` element.
- `onFocus` occurs when a form field is selected by tabbing or clicking with the mouse.
- `onBlur` happens when a form field loses its focus.
- `onSubmit` means, simply, that a user has submitted a form.
- `onSelect`, as opposed to `onFocus`, occurs when the text has been selected within a single or multiline field.
- `onChange` occurs when a form field has lost its focus and the data has been changed.

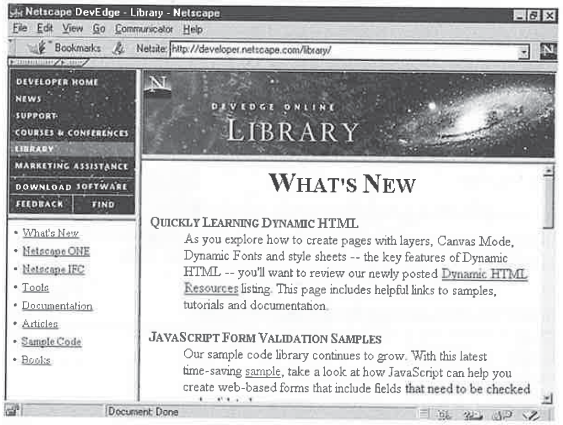
Intrinsic event handling and Dynamic HTML are covered in Chapter 22, "Dynamic HTML."

Frame Implementation

Frames—devices used to divide a Web page into separate regions for navigation and layout purposes—were ushered in by the release of versions 3 of Netscape Navigator and Microsoft Internet Explorer during the summer of 1996. Figure 17.3, a Web page from Netscape's site, is an excellent example of using frames to divide a Web page into regions that each have a distinct navigational or layout purpose. Despite their widespread use, frames were not a part of the official HTML 3.2 specification, which followed six months later.

HTML 4.0 rectifies that fact by standardizing the use of frames and also introducing a new element called `IFRAME`, for *inline frame*. Users of Microsoft's Internet Explorer will undoubtedly recognize `IFRAME` as the floating frame introduced in version 3 (which is not supported by Netscape). Support for this might be short-lived, because the same document that proposes it also discusses an alternative, based on the `OBJECT` element, and never comes to a definitive conclusion.

FIGURE 17.3.
The use of frames greatly enhances page layout possibilities.



Frame syntax and usage is covered in Chapter 18, "Creating Sophisticated Layouts with Frames and Layers."

The Inline Frame

The usage of `IFRAME` is similar to that of a normal frame but distinct in a few areas. With a normal frame, you replace the body tag with a frameset and then create frames within that structure. Inline frames are designed to actually reside inside the `BODY` element of a Web page, with text and other elements flowing around them accordingly. The syntax for `IFRAME` is

```
<IFRAME attributes>
    [HTML content]
</IFRAME>
```

Here are the available attributes:

- `src` is the address of a document to be displayed in an inline frame.
- `name` names the frame so that you can direct content into it.
- `frameborder` displays a border around the frame. Values are `"1"`, which is the default and turns the border on, and `"0"`, which turns the border off.
- `marginwidth` inserts the specified number of pixels between the frame border and content.
- `marginheight` inserts the specified number of pixels between the frame border and content.

- `scrolling` allows scrollbars to be displayed. Values are `"auto"` (default), `"yes"`, and `"no"`.
- `width` determines the frame width in pixels.
- `height` determines the frame height in pixels.
- `align` is an additional attribute that allows you to position the frame in relation to the line of text where it is located. Allowable values are `left`, `middle`, `right`, `top`, and `bottom`.

An example of usage might be to display another local Web page or link to an external Web site in order to create the effect of a picture within a picture, similar to what many televisions can do.

Listing 17.1. The main page that calls the inline frame.

```
<HTML>
<HEAD>
<TITLE>Inline Frame Test</TITLE>
</HEAD>
<BODY>
This is a test for the inline frame
<IFRAME src="iframe1.htm" name="iframe1" frameborder=1 scrolling=no
 width=100 height=50 align=middle></IFRAME>
Notice how it's aligned in the middle of the text and that text
automatically wraps around it.
<P><IFRAME src="http://www.samspublishing.com" name="iframe2"
frameborder=1 width=600 height=200 align=top></IFRAME>
<P>Here I've directed the Sams.net web site into the frame.
</BODY>
</HTML>
```

In Listing 17.1, I created two inline frames within the `main.htm` file. The first is a small frame that calls another HTML file and displays it within the inline frame. The second case links to the Sams.net Publishing Web site and displays it within a larger inline frame. The output of Listing 17.1 is shown in Figure 17.4.

NOTE

As of this writing, the inline frame feature works only with Microsoft Internet Explorer 3 and 4.

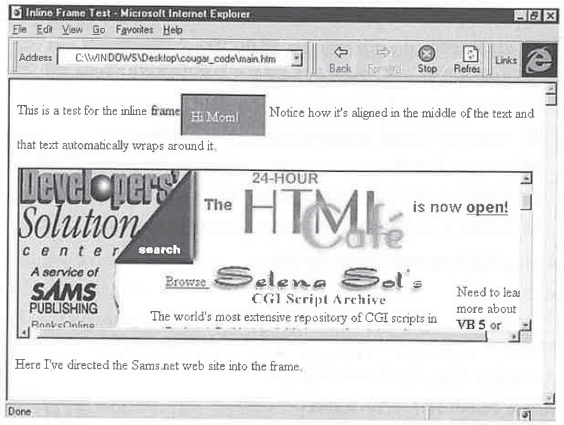


FIGURE 17.4.
Two inline frames—one containing another HTML document and the second linking to a Web site.

Form Extensions

HTML forms have existed for some time now, and the current proposal seeks to improve interactivity by including intrinsic events and providing better support for speech-based browsers. Three new elements have been proposed, and this section briefly describes the attributes associated with each.

The LABEL Element

The `LABEL` element provides speech-based browsers with the means to describe an element, and it incorporates intrinsic events to enhance interactivity. (See the "Support for Scripting" section earlier in this chapter.) Labels used to enclose a single form control are called *implicit* while those that are defined separately are called *explicit*.

The syntax is

```
<LABEL attributes>
[FORM Controls or field label text]
</LABEL>
```

Here are the available attributes:

- `for` is the ID of a form control.
- `id` is a document-wide identifier.
- `class` is a list of class names to be used in style sheets.
- `style` is element-specific style information.

- title provides additional information in relation to a particular element.
- dir specifies the directionality of the text.
- lang indicates the language used.
- disabled is used to disable form controls.
- accesskey is used to provide a keyboard shortcut.
- onClick is used to execute a script when an element is clicked.
- onFocus is used to execute a script when an element receives the focus.
- onBlur is used to execute a script when an element loses the focus.

An implicit label would look like this:

```
<LABEL>
     City
     <INPUT TYPE="TEXT" NAME="CITY>
</LABEL>
```

An explicit label would look like this:

```
<LABEL FOR="CITY">City</LABEL>
<INPUT TYPE=TEXT NAME=CITY ID=CITY>
```

Using an explicit label allows you to place the label and the associated input element in separate locations on the page, such as in different columns of a table.

The BUTTON Element

The new BUTTON element enables you to create more than the two standard buttons available now: Submit and Reset.

Here are the BUTTON element attributes:

- id is a document-wide identifier.
- class is a list of class names to be used in style sheets.
- style is element-specific style information.
- title provides additional information in relation to a particular element.
- dir specifies the directionality of the text.
- lang indicates the language used.
- disabled is used to disable form controls.
- tabindex is an integer that sets the tab order.
- onClick is used to execute a script when an element is clicked.
- onFocus is used to execute a script when an element receives the focus.
- onBlur is used to execute a script when an element loses the focus.

The FIELDSET Element

The FIELDSET element has been proposed in order to group related fields together to give speech-based browsers the capability to describe the different groups and let the user move from group to group.

Here are the attributes:

- id is a document-wide identifier.
- class is a list of class names to be used in style sheets.
- style is element-specific style information.
- title provides additional information in relation to a particular element.
- dir specifies the directionality of the text.
- lang indicates the language used.

HTML forms are more fully covered in Chapter 15, "Building and Using HTML Forms."

Inserting Objects

The draft currently under consideration by the W3C discusses ways to insert additional types of content (such as Java Applets or ActiveX Controls) into HTML documents. What about the APPLET tag? It is still supported but obviously is specific to Java Applets; no other objects can use the APPLET tag to gain entry into a Web page. Instead of creating a new element for every object or plug-in that could conceivably be used in a Web page, the W3C has chosen to create a single OBJECT element that functionally replaces the IMG element for inserting media into an HTML document and can be extended if necessary.

The OBJECT Element

The OBJECT element has the following syntax:

```
<OBJECT attributes>
     [Parameters]
     [alternate content for those without the capability to view your media-type]
</OBJECT>
```

Here are the available attributes:

- id is a document-wide identification.
- declare allows you to imply an object without actually creating or instantiating it until needed.
- classid is a URL identifying the class identifier or implementation of an object.
- codebase is a URL pointing to the location of the code.
- data is another URL that points to any data an object might require.
- type identifies the Internet Media type for the data as referenced in the DATA attribute.

- codetype identifies the Internet Media type of the code as referenced in the CLASSID attribute.
- standby allows the display of a short message to be shown while the object is being loaded.
- align specifies how to align the object relative to the current text line or aligned as a separate entity. Possible values are texttop, middle, textmiddle, baseline, textbottom, left, center, and right.
- width specifies the width allotted to the object within the browser window.
- height specifies the height allotted to the object within the browser window.
- border shows a border around the object when the value is "1", and it shows no border when set to "0".
- hspace is similar to what is used in an inline image. It allows you to insert extra space to "cushion" the object.
- vspace is similar to what is used in an inline image. It allows you to insert extra space to "cushion" the object.
- usemap attributes the object's URL if the object is a client-side image map.
- shapes is used in conjunction with objects that have anchors or links defined by shaped areas.
- name is used with HTML forms that might contain an object, in order to determine whether that object should send data to the server when the form is submitted.
- alt is alternate content to be displayed if the user can't or doesn't want to display the object.
- title is the title of the object to be displayed.

Here is an example of how to insert a QuickTime movie into a Web page:

```
<OBJECT data=Apollo13.mov
        type="video/quicktime"
        alt="Problem"
        title="Apollo 13">
    <img src=Bang.gif alt="Problem">
</OBJECT>
```

The image, bang.gif, is displayed if the user can't view the QuickTime movie.

The PARAM Element

The PARAM element, which is a list of named parameters to be passed to initialize an object, is associated with the OBJECT element.

The syntax is

```
<PARAM attributes>
```

You should note that there is no closing tag.

Here are the attributes:

- name defines the property name.
- value is the data you will be passing to the object.
- valuetype defines the type of value to be passed. Allowable values are REF, which indicates a URL; OBJECT, which points to the URL of an OBJECT element in the document; and DATA, the default value, which is passed directly to the OBJECT as a string.

Other Items Under Consideration

WD-entities, Additional Named Entities for HTML extends HTML support for all characters in ISO 8859-1, Adobe Symbol font characters representable by glyphs, characters required for internationalization, and characters that lie outside of ISO 8859-1 but are included in CP-1250.

RFC 1867, Form-based File Upload in HTML recommends two changes to HTML 3.2 and would add a new MIME media type in order to enable the user to submit a file for upload from an HTML form.

RFC 1942, HTML Tables was also used as the basis for and considered a superset of the HTML 3.2 Recommendation. Improvements to the HTML 4.0 specification are focused on supporting style sheets and improving accessibility issues. You can turn to Chapter 10, "Creating Tables for Data and Page Layout," for further information.

RFC 2070, Internationalization of the Hypertext Markup Language discusses the issue of internationalization of HTML and is more fully covered in Chapter 39, "Internationalizing the HTML Character Set and Language Tags."

Dynamic HTML

Dynamic HTML is not mentioned by name in HTML 4.0, but the mechanisms have been put in place to change the way you create—and experience—Web pages. The combined effects of the proposed style sheet implementation, improvement of the HTML support for scripting languages, and the use of a wider variety of objects to be created within a Web page gives you the power to dynamically manipulate items in a Web page without accessing the server.

When you use style sheets and take advantage of the generic attributes that have been proposed, you are able to uniquely identify virtually everything in a Web page. Having done this, you can call on your favorite scripting language to change an element (such as a font face or the color of a heading) in response to a user action (such as onMouseOver, which is an intrinsic event).

However, HTML 4.0 is language-neutral. It creates a standard that specifies the interface between scripting/programming languages and HTML with complete neutrality and leaves implementation issues to be hammered out by other parties. Netscape and Microsoft might be able to claim compliance with the HTML 4.0 standard while still not supporting rival scripting languages or style sheet implementations.

Summary

As you have seen, HTML 4.0 is an ambitious attempt to improve HTML support for a wide range of HTML elements and techniques, and this chapter has just touched the surface. Aside from "housekeeping" improvements, HTML 4.0 seems to have three main themes: accessibility, dynamism, and neutrality. Whether it takes the form of internationalizing HTML or providing means for nonvisual browsers (or those with differing capabilities) to meaningfully interpret a Web page, the push is on to be able to reach the largest audience possible. In addition, through the use of style sheets, scripting, and the Object Model, Dynamic HTML is made possible.

Most striking is the move to integrate a wide variety of media types, styles, and scripts—much of which is outside of HTML proper—with HTML without attempting to restrict which type the developer can use. This doesn't mean Netscape will ever buy into ActiveX or Microsoft will stop trying to market its programming and scripting technologies to developers, but HTML is steering a course between these two giants, hoping to preserve the openness of HTML.

CHAPTER 18

Creating Sophisticated Layouts with Frames and Layers

by John Jung

IN THIS CHAPTER

There are times when using the basic HTML tags that you've learned simply isn't enough to get what you want. Sometimes you really want your Web page to stand out and truly be different. Many people turn to Java and JavaScript in such situations. Although those are generally adequate solutions, they aren't the only ones. A number of HTML extensions that are being proposed for the next update to HTML are available to you. Two such concepts and extensions are frames and layers.

Frames

Frames are the implementation of a concept currently under discussion at the World Wide Web Consortium (W3C). With frames, you can separate Web pages into distinct sections. These sections can be any size within the browser window and, if allowed, can be resized. Also, the content in one frame can be independent of other frames if the Web author so desires. It's also possible to have the user's actions in one frame affect the content of other frames.

What Are Frames Good For?

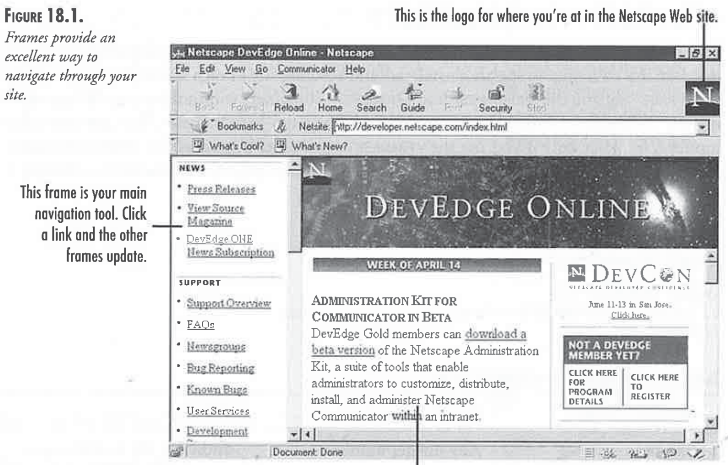
Because the contents in each frame are separate and distinctive of each other, you get better control over the information presented. For example, a commercial Web site could put the logos of its big advertisers in separate frames. Whatever the user does at such a site won't make the logos disappear. This way, you can provide free advertising to your big sponsors and not worry about someone missing it.

But frames give you far more functionality, rather than just control over what is presented. With the frames extension, you can change the content of other frames, which makes it possible for you to use frames as a navigation tool. You could display the main menu for your site in one frame and the contents in another. When the user makes a selection on the main menu, the contents frame will be updated. In fact, many such cool Web sites do precisely this (as shown in Figure 18.1).

Considerations of Frame Use

Frames can be a useful tool, but you shouldn't just rush out and start using them. You need to take some considerations into account before using frames. First, frames are a relatively new idea. As a result, some older browsers don't even support frames. For example, Lynx, the text-only Web browser, provides limited support for frames. As a result, if you use frames, many Lynx users will have problems getting around your site. Additionally, moderately older browsers have a poor user interface for navigating through frames. Netscape Navigator 2.0 was the first Web browser to support frames. Everything was fine, unless you wanted to go back a frame configuration or two.

FIGURE 18.1.
Frames provide an excellent way to navigate through your site.



Another consideration in using frames is whether they're really necessary. Although frames are a good tool, they aren't appropriate for all Web sites. If you're creating your own personal Web page, you can probably get away with using frames. After all, you want the Web page to impress only yourself and maybe your friends. You're not trying to ply any goods, so you're not really losing out on anything. If your company has a history of being a "hip" and "with-it" type of company, you could also use frames. If you're selling airplanes or power tools, however, you probably don't need them. So when should you use frames? Look at the following criteria. If you meet any of them, you should consider using frames:

- It's a personal Web page.
- Your company has a maverick/independent image.
- Your target audience will most likely be using the latest browser.

Syntax of Frame Creation

Like working with most other building blocks of HTML, creating frames isn't terribly difficult. In fact, I found tables a bit harder to create than frames. The frame extension is made up of three new HTML tags and one new attribute. The easiest of the three tags to understand is the `<NOFRAMES></NOFRAMES>` container. Basically, only Web browsers that don't support frames will display whatever is between those tags. You can use these tags to tell users what browser you prefer them to use so that they can view your frames.

<FRAMESET>...</FRAMESET>

The <FRAMESET>...</FRAMESET> container is the starting point for designing your frame layout. Use this container to specify the relative and absolute size of each frame. You can use only the <FRAME> tag and the <NOFRAMES> and <FRAMESET> containers within these tags. When creating any Web page with frames, you must first define a top-level frameset. You can create this top-level frameset only if you use the <FRAMESET> container instead of the <BODY> container. Table 18.1 details the attributes for the <FRAMESET> tag. Generally, the syntax for the <FRAMESET></FRAMESET> group of tags is as follows:

```
<FRAMESET frame_configuration...>
<FRAME SRC=URL1...>
<FRAME SRC=URL2...>
...
<FRAME SRC=URLn...>
...Additional <FRAMESET></FRAMESET> tags as needed…
</FRAMESET>
```

Table 18.1. Attributes for <FRAMESET>.

Attribute Name	*Acceptable Values*	*Purpose*
BORDER	Any integer number	Indicates the border thickness for all child frames.
COLS	A number that may or may not have a percentage sign (%) or asterisk (*)	Defines the number and size of the columns you want to create. A number without a percentage sign indicates the size of the column in pixels. A number with a percentage sign indicates the width of the frame relative to the width of the browser. An asterisk indicates that the frame is sized proportionately to other frames. Specify multiple columns by quoting the size in a comma-separated list.
FRAMEBORDER	1 or 0	Specifies whether frames are displayed with a border. A value of 1 indicates the presence of a frame border and 0 disables the frame border. Individual frames can override this attribute.

Attribute Name	*Acceptable Values*	*Purpose*
ROWS	A number that may or may not have a percentage sign (%) or asterisk (*)	Defines the number and size of the rows you want to create. A number without a percentage sign indicates the size of the row in pixels. A number with a percentage sign indicates the height of the frame relative to the width of the browser. The asterisk indicates that the frame is sized proportionately to other frames. Specify multiple columns by quoting the size in a comma-separated list.

The <FRAME> Tag

After you specify a frameset, you must define the content of those frames, which is where the <FRAME> tag comes in. The <FRAME> tag controls various attributes for a particular frame, such as the content, color, and border width. For every row or column you define in the frameset, you must have a corresponding <FRAME>. The first frame, specified by either COLS or ROWS, will use the first <FRAME> tag (as shown in Listing 18.1). You can create complicated frame layouts using the <FRAMESET> container tags instead of the <FRAME> tag. (See the section "Creating Fancy Frame Layouts," later in this chapter.) Table 18.2 completely lists all the attributes that the <FRAME> tag uses.

Listing 18.1. Code to create a frame layout.

```
<HTML>
<HEAD>
<TITLE>Sample Frameset</TITLE>
</HEAD>
<FRAMESET ROWS="50%,50%">
  <FRAME SRC="agenda.html">
  <FRAME SRC="minutes.html">
</FRAMESET>
</HTML>
```

Table 18.2. Attributes for <FRAME>.

Attribute Name	*Acceptable Values*	*Purpose*
FRAMEBORDER	1 or 0	Specifies whether frames are displayed with a border. A value of 1 indicates the presence of a frame border and 0 disables the frame border. Individual frames can override this attribute.
MARGINHEIGHT	Any integer number	Indicates the height of the top and bottom margins in pixels.
MARGINWIDTH	Any integer number	Indicates the height of the left and right margins in pixels.
NAME	Any string beginning with a letter	Identifies the name of the frame for reference. There are four reserved names: _blank, _parent, _self, and _top. The _blank name opens a new window with the specified URL. The _parent name opens the URL in the parent frame of the current frame. If you call a _self frame, the URL replaces the frame the link was originally in. The _top name displays the URL in the full window.
NORESIZE	None	Simply acts as a toggle switch. When present, it prevents the user from resizing the frame.
SCROLLING	Yes, No, or Auto	Specifies whether a scrollbar appears in the frame. When set to Auto, the browser determines whether a scrollbar should be created. Auto is the default behavior.
SRC	Any URL	Specifies the file to be displayed within the frame. If you do not specify a SRC attribute, the space where the frame would appear will be blank.

The New Attribute

The final addition to HTML that the frames extensions made was to add a new extension. That extension, TARGET, was added to the Anchor element. This attribute takes a single string that specifies the name of the frame to display the contents. The <FRAME> tag, using the NAME attribute, defines the name of the frame (as shown in Listing 18.2). This enables you to change the contents of a frame by using hypertext links. For example, Figure 18.2 shows a framed Web page and the names for each. If you click a link in the left frame, named left, the right frame, named right, is updated appropriately. The HTML source for the left frame is shown in Listing 18.1. Frequently using the TARGET attribute updates the frame. The TARGET attribute isn't part of the <FRAMESET> or <FRAME> tag definition. Rather, it's an attribute for the <A> tag, used for navigation purposes. The TARGET attribute is entirely optional, and not using it would make each frame update independently. The last link in the left frame of Figure 18.2 updates the contents of its own frame.

NOTE

To create a new window, specify an invalid frame name.

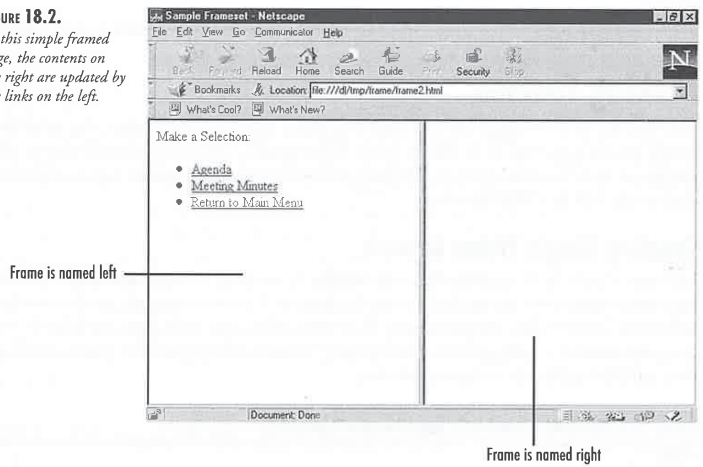


FIGURE 18.2.
In this simple framed page, the contents on the right are updated by the links on the left.

Listing 18.2. The HTML source for the entire Web page shown in Figure 18.2.

```
<HTML>
<HEAD>
<TITLE>Sample Frameset</TITLE>
</HEAD>
<FRAMESET COLS="50%,50%">
  <FRAME SRC="left.html">
  <FRAME SRC="right.html">
</FRAMESET>
</HTML>
```

Listing 18.3. HTML source for the frame `Left` shown in Figure 18.2.

```
<HTML>
<HEAD>
<TITLE>This title doesn't show up in the browser window</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
Make a Selection:
<UL>
<LI><A HREF="agenda.html" TARGET="right">Agenda</A></LI>
<LI><A HREF="minutes.html" TARGET="right">Meeting Minutes</A></LI>
<LI><A HREF="mainmenu.html" TARGET="_self">Return to Main Menu</A></LI>
</UL>
</BODY>
</HTML>
```

Making Frames

Now you know what frames are and what arguments they use. But it takes a lot more than simply knowing the tags to be able to create frame layouts. You need to be able to put what you've just read about into practice. I'll give you some simple frame layouts and show you how they would look in a Web browser.

Creating Simple Frame Layouts

Let's start out on applying your frame knowledge by creating a simple Web page. It should have two frames—one on top and one on the bottom. You won't care about the content in each frame because those are just regular Web pages. Also, you won't have any links that update each other; I already covered that. Figure 18.3 shows what your Web page should look like, and Listing 18.4 shows how it was done.

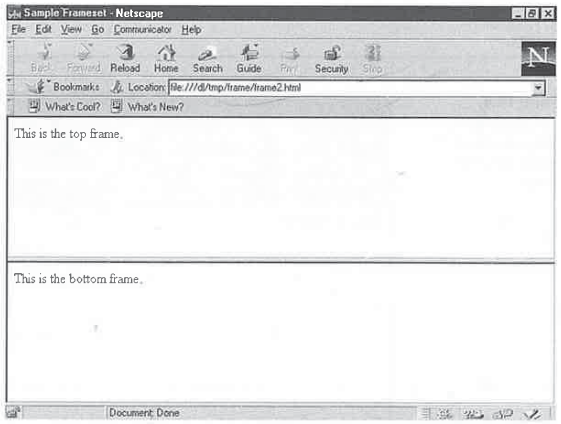
Listing 18.4. The HTML source for the page shown in Figure 18.3.

```
<HTML>
<HEAD>
<TITLE>Two Frame Web Page</TITLE>
</HEAD>
```



```
<FRAMESET ROWS="50%,50%">
  <FRAME SRC="top.html">
  <FRAME SRC="bottom.html">
</FRAMESET>
</HTML>
```

FIGURE 18.3.
This simple frame layout has only two frames.



Maybe that was a bit too easy. I think you could figure out the appropriate values to plug into the `ROWS` attribute. Also, with two frames, it's not hard to guess which Web page goes into which frame. Let's try something more difficult. Let's try three frames—two next to each other and the third under both of them. (See Figure 18.4.) Take a look at Listing 18.5 to see how it should be done. Basically, you first create a `<FRAMESET>` that encapsulates the general look of the Web page, such as splitting it horizontally in this case. Next, instead of using a `<FRAME>` to specify the Web page to be presented in the frame, you specify another `<FRAMESET>`. This nested `<FRAMESET>` further subdivides the region, allowing you to display even more Web pages. Because you are splitting the top frame region, the nested `<FRAMESET>` will be split vertically. This nested `<FRAMESET>` receives its contents from the two `<FRAME>` tags between the beginning and ending tags. Finally, to specify the Web page for the bottom frame, you simply use a regular `<FRAME>` tag.

Listing 18.5. HTML source for the page shown in Figure 18.4.

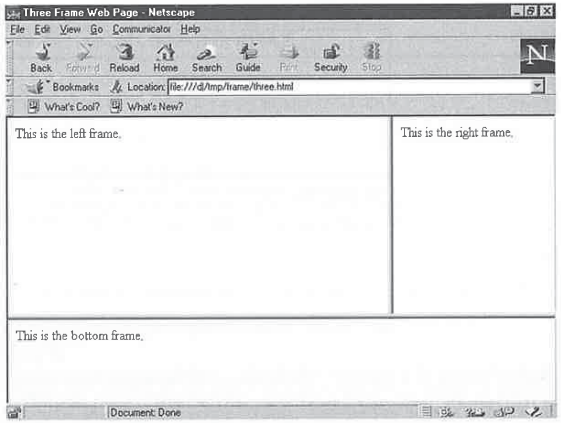
```
<HTML>
<HEAD>
<TITLE>Three Frame Web Page</TITLE>
</HEAD>
```

continues

Listing 18.5. continued

```
<FRAMESET ROWS="70%,30%">
  <FRAMESET COLS="70%,30%">
    <FRAME SRC="left.html">
    <FRAME SRC="right.html">
  </FRAMESET>
  <FRAME SRC="bottom.html">
</FRAMESET>
</HTML>
```

FIGURE 18.4.
This frame layout, which can be a little tricky, uses three frames.



The important part to remember about this example is that you nested <FRAMESET> tags, which is how you were able to have the left and right frames. You now have the basics of creating nice, functional layouts. With these ideas, you can create truly advanced frame layouts if you think about how to lay out a page beforehand. The next section deals with creating sophisticated frame layouts.

Creating Fancy Frame Layouts

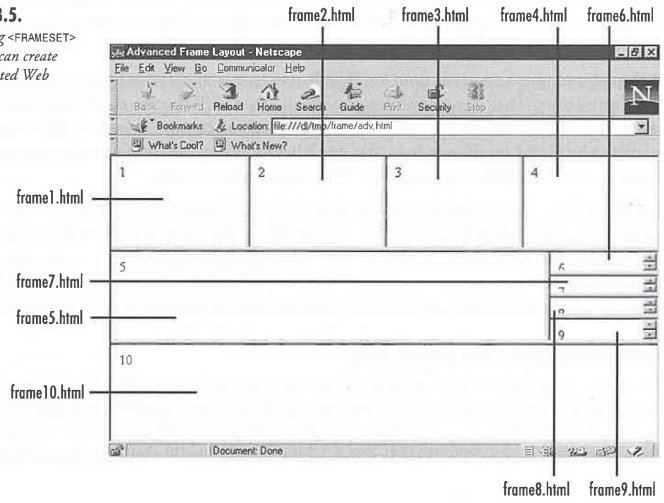
It's quite possible, by using frames, to have really cool-looking frame layouts. Earlier, I briefly mentioned that you could nest <FRAMESET> tags. This procedure is the heart of making a truly spectacular Web page. Because <FRAMESET> uses the currently available window space, nesting enables you to further subdivide the window. For example, consider how Figure 18.5 was created. Or, you can cheat and look at Listing 18.6. Basically, you split the window into three horizontal frames, leaving the bottom frame alone. You subdivide the top frame into four nested frames, all of them next to each other. You then split the middle region into two nested frames,

leaving the left frame region alone. Finally, you split the middle left frame region into four frames, one on top of each other.

Listing 18.6. HTML source for the frame layout in Figure 18.5.

```
<HTML>
<HEAD>
<TITLE>Advanced Frame Layout</TITLE>
</HEAD>
<FRAMESET ROWS="33%,33%,33%">
  <FRAMESET COLS="25%,25%,25%,25%">
    <FRAME SRC="blank1.html">
    <FRAME SRC="blank2.html">
    <FRAME SRC="blank3.html">
    <FRAME SRC="blank4.html">
  </FRAMESET>
  <FRAMESET COLS="80%,20%">
    <FRAME SRC="blank5.html">
    <FRAMESET ROWS="25%,25%,25%,25%">
      <FRAME SRC="blank6.html">
      <FRAME SRC="blank7.html">
      <FRAME SRC="blank8.html">
      <FRAME SRC="blank9.html">
    </FRAMESET>
  </FRAMESET>
  <FRAME SRC="blank10.html">
</FRAMESET>
</HTML>
```

FIGURE 18.5.
By nesting <FRAMESET> tags, you can create sophisticated Web pages.



Layers

Layers are a very new set of tools that help make neat-looking Web pages. Layers, created by Netscape, enable Web authors to define any number of *sub-Web pages*—pages whose content is independent of each other and the main Web page. They can show any URL, just like a Web page, or a frame, but also have much flexibility. Frames can be positioned, sized, and moved around with various JavaScript code. Further, you can put layers on top of each other, and have only certain ones showing their content. Hence, they're called "layers." When you use layers correctly, you can create some outstanding Web pages. Currently, only Netscape Navigator 4.0 will support the layers, although Internet Explorer 4.0 will probably do so as well.

Differences Between Frames and Layers

"Layers sound a lot like frames," I hear you saying. Not at all. Layers and frames are different in numerous respects. First, you can't simply have one frame; you must have at least two. That means that the frame layout takes up the entire Web browser window. Layers, on the other hand, are separate entities that take up only a certain amount of space. If the Web author desires, he or she can use a layer that takes up the entire Web browser window space. But perhaps more importantly, layers have two functions that frames simply lack.

First, you can control the visibility of a layer. In other words, you can show or hide any and all layers. This is quite different from frames, which simply must take up the entire Web browser window space. You cannot create a frame that takes up only a certain portion of the browser window. Furthermore, you can use JavaScript (as discussed in the section "Layers and JavaScript" in this chapter) to change layer visibility so that—depending on what a user does with JavaScript—a different layer shows.

A second and very important function that frames lack is the precise layout control that you have with layers. That is, with frames, as with all other HTML tags, the browser determines where each tag goes in the Web browser. With layers, you can specify exactly where you want a layer to be placed on the window. You don't have to worry about a layer showing up in different locations on different computers. *You* control where it goes.

Finally, with layers, you also have the built-in ability to change their positions. So, if you were to use JavaScript, you could control not only which layers are seen but where they appear. This gives you unparalleled control over how your Web page looks. Additionally, you could use this ability to move layers to create neat animations. For example, you could create a set of images of an airplane, put each on a separate layer, and use JavaScript to control which layer is visible.

Considerations for Layer Use

For all the neat functionality of layers, there are some reasons you don't want to use them immediately. The primary reason is that, currently, only Netscape Communicator supports layers. Although there are a lot of Netscape Navigator users, there aren't necessarily a lot of Communicator owners. The fact that Netscape Communicator is released doesn't mean that all Navigator users will rush out and use it. As a result, by using layers, you could be aiming for a rather small market. Almost all corporate sites should avoid layers until other browsers start to support them. You can probably get away with using layers in personal Web pages, but don't be surprised if you get some complaints. The primary reasons you would want to use layers are the same as those for wanting to use frames. If you meet any of the following criteria, you should probably use layers:

- You are creating a personal Web page.
- Your company has a maverick or independent image.
- Your target audience will most likely be using the latest browser.

Another consideration in using layers is that JavaScript handles most of its functionality. That means that along with further alienating some users, you should watch the content. It's too easy for developers to overload their JavaScript Web pages with graphics and sound. Layers makes it all the more tempting to put up really cool Web pages and not consider the user. Far too many Web pages today, without layers, have large Java applets and large images. It can take a considerable amount of time, even with intranet-level connections, to retrieve some of these pages. This situation can only get worse if people don't think about the consequences of using layers.

An additional consideration for the usage of layers is that the W3C hasn't agreed on its specifications. Although this isn't surprising for cutting-edge HTML code, the standards committee has also shelved discussion on layers, so it's possible that the W3C has completely lost interest in the idea of layers. Therefore, it's possible that the W3C will simply drop the proposed layer tags entirely. You have no guarantee that layers *will* be dropped, but you also have no guarantee that they won't.

Yet another consideration in using layers is how much work you want for yourself. If you intend to use layers extensively, you must remember that it will create a bit of work. Because layers can be anywhere on the screen, they are out-flow layers, which means that any HTML code after a layer will appear exactly where it would go if the layer tags weren't used. For example, suppose you give a title to a Web page, define a layer, and then add a bulleted list of items. The bulleted list of items would come immediately below the title of the Web page, not after the layer. You might want to use an in-flow layer for such a purpose.

What Are In-Flow and Out-Flow Layers?

In this section, I talk about how to create layers. The new layer specifications allow for two different types of layers: out-flow and in-flow layers. An out-flow layer—probably what you, like most people, will use—is positioned exactly where the Web author wants it to be. To create an out-flow layer, use the `<LAYER>...</LAYER>` container tags. An in-flow layer is offset from the last HTML tag or text. To create in-flow layers, use the `<ILAYER>...</ILAYER>` container tags. The `<ILAYER>` tag has exactly the same argument as the `<LAYER>` tag. Because the syntax for using both are the same, for simplicity's sake I'll talk about only out-flow layers.

Attributes and Methods of Layers

Layers are of a similar nature to frames in that they divide the Web browser window into distinct objects. You can't change frames dynamically. Therefore, when a user selects a particular link, you should have a frame change its size. Layers are simply the next progression from frames, allowing you to modify their attributes based on user events. In creating layers, you can set a number of possible attributes. In addition, each layer you create has a number of predefined methods associated with it. *Methods* is just another name for *functions* or *subroutines*, if you prefer. Neither layer attributes nor methods were created for frames.

Layer Attributes

You can set a certain number of attributes for each layer you create. These attributes are just like regular HTML tag attributes, with one big difference: For almost every attribute available, a corresponding layer property is available. I'll talk more about layer properties in the next section, "Layer Properties and Methods." Table 18.3 lists all the attributes for the `<LAYER>...</LAYER>` container tags. The general `<LAYER>` tag can be broken down as follows:

```
<LAYER layer_attributes>
...HTML code or plain text...
</LAYER>
```

Table 18.3. Attributes for <LAYER> and <ILAYER> container tags.

Attribute Name	*Acceptable Values*	*Purpose*
`ABOVE`	Any layer name	Indicates that the current layer is above the specified layer name.
`BACKGROUND`	Any URL	Specifies the background image to be displayed in the layer.
`BELOW`	Any layer name	Indicates that the current layer is below the specified layer name.
`BGCOLOR`	`#RRGGBB`	Identifies the background color for the layer. `RR`, `GG`, and `BB` are hexadecimal numbers that give the intensity of the red, green, and blue values of a color. You can specify only two digits for the hexadecimal numbers.
`CLIP`	`Integer,Integer, Integer,Integer`	Specifies the left, top, right, and bottom coordinates by four-integer numbers for the layer in pixels. Whatever is in the layer contained in the `CLIP` rectangle is displayed.

Attribute Name	*Acceptable Values*	*Purpose*
`LEFT`	Any integer value	Specifies the left edge of the layer in pixels from the left side of the browser window. For `<ILAYER>`, this attribute specifies the offset from the previous HTML tag or text.
`NAME`	Any string beginning with a letter	Identifies the layer for Java or JavaScript referential use.
`TOP`	Any integer value	Specifies the top edge of the layer in pixels from the top of the browser window. For `<ILAYER>`, this attribute specifies the offset from the previous HTML tag or text.
`VISIBILITY`	`SHOW`, `HIDE`, or `INHERIT`	Determines whether the layer is visible. The `INHERIT` value causes the layer to take the visibility attribute of its parent.
`WIDTH`	Any integer value	Indicates the width of the layer in pixels.
`ZINDEX`	Any integer value	Indicates the index value for the layer for stacking and displaying.

Layer Properties and Methods

Because Netscape intended layers to enable you to create dynamic Web pages, layers have a few extra features. All of these features provide hooks for scripting languages into accessing aspects of the layer. The script can change values of some properties or call some layer methods. Table 18.4 shows all the properties for all layers. Table 18.5 gives you a list of methods available to all layers.

Table 18.4. The <LAYER> tag properties.

Attribute Name	*Can Be Changed?*	*Acceptable Values*	*Description*
`above`	No	Any layer name	Specifies the parent layer of the current layer for nested layers.
`background`	Yes	Any URL	Specifies the background image to be displayed in the layer.

continues

Table 18.4. continued

Attribute Name	*Can Be Changed?*	*Acceptable Values*	*Description*
`below`	No	Any layer name	Specifies the child layer of the current layer for nested layers.
`clip.top`, `clip.left`, `clip.right`, `clip.bottom`, `clip.width`, `clip.height`	Yes	All properties accept any integer value	Control the various aspects of the clipping rectangle.
`bgColor`	Yes	`#RRGGBB`	Identifies the background color for the layer. `RR`, `GG`, and `BB` are hexadecimal numbers that give the intensity of the red, green, and blue values of a color. You can specify only two digits for the hexadecimal numbers.
`height`	No	Any integer value	Indicates the height of the layer in pixels.
`layers`	No	An array	Stores the index and name for each defined layer.
`left`	Yes	Any integer value	Specifies the number of pixels, from the left edge of the browser window, where the layer should start.
`name`	No	Any string beginning with a letter	Identifies the layer for Java or JavaScript referential use.
`sibling.Above`	No	Any layer name	Indicates that the current layer is above the specified layer name. A null value indicates the top-level layer.

Attribute Name	*Can Be Changed?*	*Acceptable Values*	*Description*
`sibling.Below`	No	Any layer name	Indicates that the current layer is below the specified layer name. A null value indicates the bottom-most layer.
`top`	Yes	Any integer value	Specifies the number of pixels from the top of the browser window, where the layer starts.
`visibility`	Yes	`show, hide, or inherit`	Determines whether the layer is visible. The `INHERIT` value causes the layer to take the visibility attribute of its parent.
`width`	No	Any integer value	Indicates the width of the layer in pixels.
`zINDEX`	Yes	Any integer value	Indicates the index value for the layer for stacking and displaying. Layers with a higher `zINDEX` value will be stacked above lower `zINDEX` numbered layers.

Table 18.5. The <LAYER> tag methods.

Method Name	*Parameters*	*Parameter Data Type*	*Description*
`offset`	`(x,y)`	`integer, integer`	Causes the layer to be offset on the left by the specified x value. The top is similarly offset by the specified y value.
`moveAbove`	`(layer_name)`	`string`	Adjusts the stacking array to place the current layer above the specified layer. Both layers will have the same parent.

continues

Table 18.5. continued

Method Name	*Parameters*	*Parameter Data Type*	*Description*
`moveBelow`	`(layer_name)`	`string`	Adjusts the stacking array to place the current layer below the specified layer.
`moveTo`	`(x,y)`	`integer, integer`	Places a layer at a precise screen location. You specify the exact position at which you want the upper-left corner of the layer to be placed.
`resize`	`(width, height)`	`integer, integer`	Resizes the current layer. The first value is the desired width of the layer and the second is the desired height.

Layers and Languages

One nicer aspect of layers is that it has built-in support for languages. You can use just about any scripting language, such as JavaScript, to expand the capabilities of layers. You can dynamically turn layers on or off when the user moves his or her mouse. You can exploit the features of both the language and layers. The first part in accessing the features of layers is to be able to look at and change its properties. You can do this by using the following syntax:

```
document.layername.property
```

Here, `document` is the required string, but `layername` and `property` are variables. The `layername` represents the name that you've given to the layer, and `property` is the property of the layer that you want to access. (See Table 18.4.) Similarly, you can access the methods of a particular layer with the following syntax:

```
document.layername.method(params)
```

Once again, `document` is a required string, and `layername` is the name of the layer. This time, `method` is the name of the method you want to access, and `params` are the parameters you're passing. Table 18.5 completely lists all the methods and their parameters built into all layers. I'll talk more about using properties and methods later in this chapter.

Creating Layers

So now that you know the technical syntax of creating layers, it should be a snap to create them, right? Well, not always. Sometimes, it helps to try out what you know before you put it into your Web page. As a result, I'll go through a couple of examples of using layers. I'll show you a Web page with some layers and then show you how it was done.

Basic Layer Use

In some ways, creating layers is much easier than creating frames. With layers, you can specify the exact size and position you want the layer to be. Also, all HTML code between the `<LAYER>...</LAYER>` container is processed as usual. So, let's start by trying to create three layers, each next to each other. Figure 18.6 shows you the look to strive for, and it was created with the code in Listing 18.7. In this example, you use three out-flow layers and simply specify their background color. Out-flow layers are used because they give you very precise control over where the layers are placed. In this example, the layers are simply positioned next to each other, so that it is easier to see all of them.

Listing 18.7. HTML source for the page shown in Figure 18.6.

```
<HTML>
<HEAD><TITLE>Sample of Three Layers</TITLE></HEAD>
<BODY BGCOLOR="#FFFFFF">
<H1><CENTER>Three Layers Next to Each Other</CENTER></H1>
<LAYER NAME="red" LEFT=50 TOP=50 WIDTH=50 HEIGHT=50 VISIBILITY=SHOW
  BGCOLOR="#FF0000">
</LAYER>
<LAYER NAME="green" LEFT=100 TOP=50 WIDTH=50 HEIGHT=50 VISIBILITY=SHOW
  BGCOLOR="#00FF00">
</LAYER>
<LAYER NAME="blue" LEFT=150 TOP=50 WIDTH=50 HEIGHT=50 VISIBILITY=SHOW
  BGCOLOR="#0000FF">
</LAYER>
</BODY>
</HTML>
```

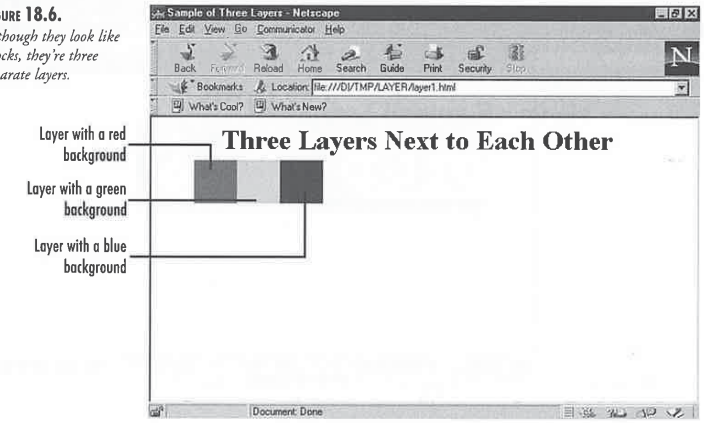


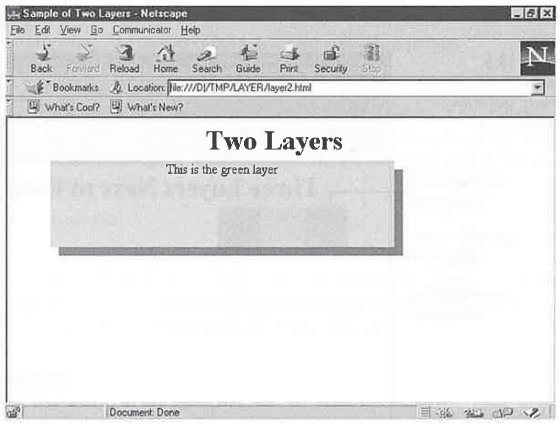
FIGURE 18.6. *Although they look like blocks, they're three separate layers.*

That wasn't terribly hard, was it? Let's go on to something a little bit trickier—stacking the order of the layers. This is an important exercise to understand because you can use it to script. You could easily have multiple layers on top of each other and simply choose which one you want on top. Figure 18.7 shows two layers, one slightly offset from the other, which create a drop-shadow effect. Listing 18.8 shows you how it was done. In this example, you first create a layer that will act as the shadow of the second layer. Because it's a shadow, use a gray color as the background, although black would work as well. Next, create an out-flow layer for the layer that's getting the shadow, and position it up and to the left from its shadow. This makes only a small part of the shadow layer appear from underneath the green layer.

Listing 18.8. HTML source for the page shown in Figure 18.7.

```
<HTML>
<HEAD><TITLE>Sample of Two Layers</TITLE></HEAD>
<BODY BGCOLOR="#FFFFFF">
<H1><CENTER>Two Layers</CENTER></H1>
<LAYER NAME="grey" LEFT=60 TOP=60 WIDTH=400 HEIGHT=100 VISIBILITY=SHOW
  BGCOLOR="#888888" zINDEX=3>
</LAYER>
<LAYER NAME="green" LEFT=50 TOP=50 WIDTH=400 HEIGHT=100 VISIBILITY=SHOW
  BGCOLOR="#00FF00" zINDEX=1>
  <CENTER>This is the green layer</CENTER>
</LAYER>
</BODY>
</HTML>
```

FIGURE 18.7.
By controlling the stacking order of the layers, you can create nice special effects.



This might seem rather simple and straightforward—because it is. Layers aren't that hard to understand or to use. Their syntax is similar in nature to that of frames. As a result, if you know how to manage frames well, you should be able to manage layers pretty well, too.

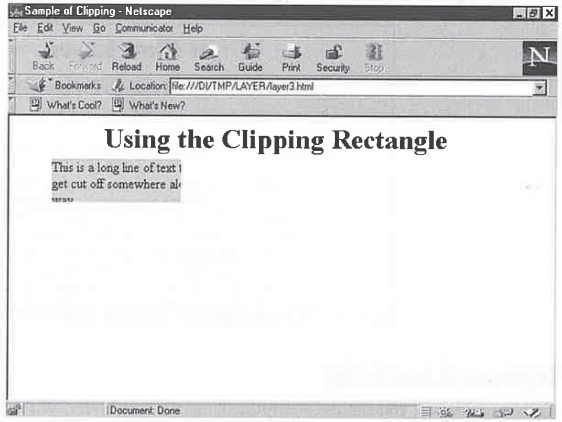
Layers and Clipping Rectangles

Another useful aspect of layers is the ability to define clipping rectangles. A clipping rectangle is a specified region in which only contents in that region are displayed. You could have a layer with lots of HTML tags but only display part of it. When the user makes a particular selection, you could simply move the clipping rectangle. Take a look at Figure 18.8 and notice that some of the text is cut off. That's the clipping rectangle in action, even though you don't see a border around it. You can see how I created that figure by looking at Listing 18.9.

Listing 18.9. HTML source for the page shown in Figure 18.8.

```
<HTML>
<HEAD><TITLE>Sample of Clipping</TITLE></HEAD>
<BODY BGCOLOR="#FFFFFF">
<H1><CENTER>Using the Clipping Rectangle</CENTER></H1>
<LAYER NAME="green" LEFT=50 TOP=50 WIDTH=200 HEIGHT=200 VISIBILITY=SHOW
  CLIP="0,0,150,50" BGCOLOR="#00FF00">
This is a long line of text that <B>will</B> get cut off somewhere along the
 way.
</LAYER>
</BODY>
</HTML>
```

FIGURE 18.8.
The clipping rectangle cuts off some of the text.



In-Flow Layers

So far, I've shown you how to use only out-flow layers. What about the layers that simply start off where the last bit of HTML code ends? These in-flow layers are equally easy to manage and create. Look at Figure 18.9 and you'll notice that some of the text appears to be on a colored block. That green block is actually an in-flow layer. Notice that the text before and after the layer continues as usual, with no break-up. Listing 18.10 shows you how the in-flow layer was created.

Listing 18.10. HTML source for the page shown in Figure 18.9.

```
<HTML>
<HEAD><TITLE>Sample of In-Flow Layers</TITLE></HEAD>
<BODY BGCOLOR="#FFFFFF">
<H1><CENTER>In-Flow Layers</CENTER></H1>
This is a line of text that will be followed by
<ILAYER VISIBILITY=SHOW WIDTH=50 HEIGHT=50 BGCOLOR="#00FF00">
<B><U>a layer</U></B>
</ILAYER>
, and then some more text.
</BODY>
</HTML>
```

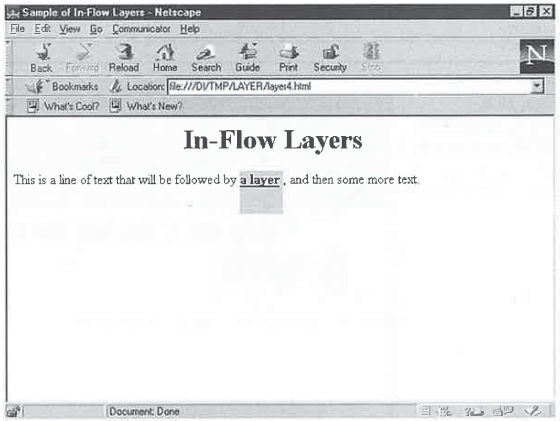


FIGURE 18.9.
Some of this text is actually part of an in-flow layer.

Layers and JavaScript

As I've mentioned, you can use a scripting language with layers. With this combination, you can create an extremely impressive looking Web page. It makes it possible to have floating help

that describes what each link does. Also, you can provide a great deal more interaction with the user than regular HTML allows.

Figure 18.10 gives an example of a Web page with links on the left side. When the user moves his or her mouse over each link, the help layer on the right changes. The right layer gives specific information for the purpose of the link on the left. Listing 18.11 shows you the entire Web page that I used to accomplish this feat. This trick is accomplished by using a separate layer for each link. When the cursor is over a particular link, all layers are hidden except for the appropriate one. Also, when the mouse is moved off the link, the generic description layer is displayed.

Listing 18.11. HTML source for the page shown in Figure 18.10.

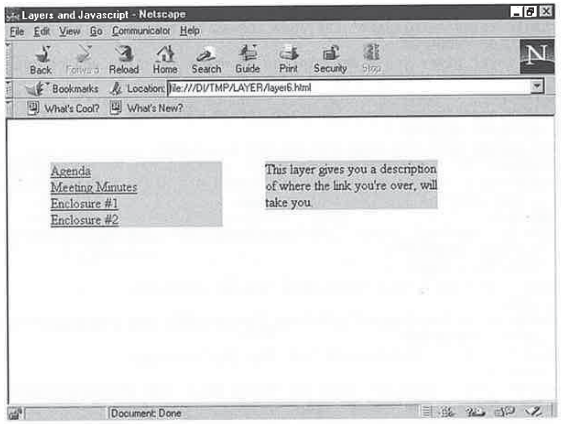
```
<HTML>
<HEAD><TITLE>Layers and JavaScript</TITLE></HEAD>
<BODY BGCOLOR="#FFFFFF">
<LAYER name="helplayer0" LEFT=300 TOP=50 WIDTH=200 VISIBILITY=SHOW
  BGCOLOR="#00FF00">
This layer gives you a description of where the link you're over, will take
  you.
</LAYER>
<LAYER name="helplayer1" LEFT=300 TOP=50 WIDTH=200 VISIBILITY=HIDE
  BGCOLOR="#FFFF00">
The agenda from the last meeting are available.
</LAYER>
<LAYER name="helplayer2" LEFT=300 TOP=50 WIDTH=200 VISIBILITY=HIDE
You can read the last meeting minutes.
</LAYER>
<LAYER name="helplayer3" LEFT=300 TOP=50 WIDTH=200 VISIBILITY=HIDE
  BGCOLOR="#00FFFF">
  BGCOLOR="#FF00FF">
Take a look at enclosure #1 from the last meeting.
</LAYER>
<LAYER name="helplayer4" LEFT=300 TOP=50 WIDTH=200 VISIBILITY=HIDE
  BGCOLOR="#F0F0F0">
Take a look at enclosure #2 from the last meeting.
</LAYER>
<LAYER NAME="linklayer" LEFT=50 TOP=50 WIDTH=200 VISIBILITY=SHOW
  BGCOLOR="#00FF00">
  <A HREF="agenda.html" onMouseOver="changeLayer(1)"
  onMouseOut="changeLayer(0)">Agenda</A><BR>
  <A HREF="minutes.html" onMouseOver="changeLayer(2)"
  onMouseOut="changeLayer(0)">Meeting Minutes</A><BR>
  <A HREF="encl01.html" onMouseOver="changeLayer(3)"
  onMouseOut="changeLayer(0)">Enclosure #1</A><BR>
  <A HREF="encl02.html" onMouseOver="changeLayer(4)"
  onMouseOut="changeLayer(0)">Enclosure #2</A><BR>
</LAYER>
<SCRIPT>
function hideLayers ()
{
```

continues

Listing 18.11. continued

```
  document.layers["helplayer0"].visibility = "hide";
  document.layers["helplayer1"].visibility = "hide";
  document.layers["helplayer2"].visibility = "hide";
  document.layers["helplayer3"].visibility = "hide";
  document.layers["helplayer4"].visibility = "hide";
}
function changeLayer (n)
{
  hideLayers();
  document.layers["helplayer" + n].visibility = "inherit";
}
</SCRIPT>
</BODY>
</HTML>
```

FIGURE 18.10.
The text on the left is contained within its own layer.



The key parts of this listing are the `onMouseOver` and `onMouseOut` attributes in the anchor elements. Netscape proposed these extensions to provide a better mechanism for event handling. You'll also notice that all I'm really doing is hiding all the layers and then revealing the correct layer. Therefore, when the user positions the mouse over a particular link in the left layer, the corresponding right layer is exposed. Furthermore, a "default" generic layer explains how to get more help on each link. So, whenever the user moves off of any link in the left layer, the generic layer is displayed. With this approach, you don't need to keep track of which layer is currently visible.

Obviously, this is just the tip of the iceberg for the functionality of layers and JavaScript. You really modified only one attribute—the `visibility` attribute—and used JavaScript code to modify the values. It's possible to expand this Web page to do much more. You could simulate nested menus with floating help simply by adding the `moveTo()` or `offset()` methods. Or, you could have a layer on the screen and, depending on the user input, resize it to show more information. You could even create a simple game just by messaging around with layers, their visibility, and their positions.

Summary

In this chapter, I covered some of the cooler HTML extensions that Netscape proposed. I talked about frames and layers and the pluses and minuses of each. Also, I gave you a comprehensive list of the attributes and syntax for creating such elements. I provided you with some examples on how to create sample Web pages with these elements. You can take these examples and expand on them, based on what you learned here.

What's Next?

The next chapter focuses on a new development created for HTML: cascading style sheets. Style sheets are a way of providing a uniform look and feel to all your Web pages. For corporate Web sites, this takes a lot of the work needed to make a site look consistent. But cascading style sheets also give you some extra functionality that you might not have thought about.

CHAPTER 19

Introducing Cascading Style Sheets

by Molly E. Holzschlag

IN THIS CHAPTER

Style sheets have been available for several years, but only recently has their power been recognized with a new eagerness to push the design of Web pages to new limits. Style sheets were at first overlooked in lieu of proprietary extensions offered by Netscape, because Netscape's solutions were available to more people for viewing and style sheets weren't. With style sheets, Web designers are gaining the ability to control how both text and design elements are laid out on pages through HTML, with a level of sophistication beginning to approach many word processors.

Style sheets are a means of controlling the way HTML tags are formatted. Because many Web designers have come from a graphic design or desktop publishing background, style control is part of the natural approach to design. But style tools were unsupported by browsers for HTML. Even before the invention of style sheets, HTML specifically avoided advanced style options, offering instead only very standard or rudimentary means of creating style elements such as color, headers, and margins. Special effects such as text shadowing required graphic images, which in turn slowed down pages with extra data passed from server to client.

Cascading refers to the fact that not only can multiple styles be used in an individual HTML page, but the browser will follow an order (a cascade) to interpret the information. This means that, out of the three types of currently available style sheets, a designer can choose to use all three simultaneously. The browser then looks for information in an orderly and predetermined fashion, delivering the pervading style sheet.

Style Sheets and Style Control

Before style sheets, style control with HTML was clumsy at best, and nonexistent at worst. Style sheets give a Web designer better ways of working with critical issues such as fonts, and the designer also has a means to control other traditional graphic design issues such as typeface, style and size, font weight, and leading (pronounced "ledding").

Internet Explorer set the pace by introducing style sheets in the Internet Explorer 3.0 release, and now Netscape has incorporated style sheets into its Communicator 4.0 release.

TIP

Ironically, when a page using style sheets without other HTML formatting is viewed by Netscape 3.0 or below or Internet Explorer 2.0 or below, the page reverts to the old reliable, and nasty gray background with demoted fonts and other styles. (See Figures 19.1 and 19.2.)

FIGURE 19.1.
A page using cascading style sheets, as it appears in Internet Explorer 3.0.

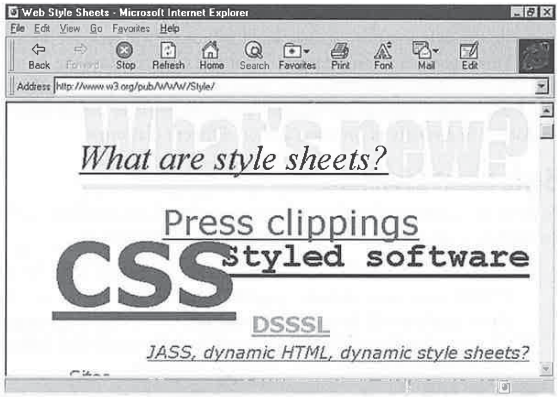
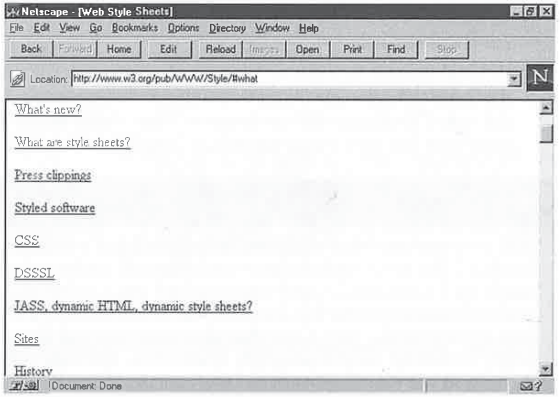


FIGURE 19.2.
The same page reverts to bare-bones defaults in any Netscape browser below version 4.0.



This doesn't mean that certain challenges aren't inherent in cascading style sheets. These challenges include issues such as the difference between various fonts loaded on different machines. Font style, including face and weight, can be called by a style sheet, but if the font and font weight isn't on the visitor's machine, the browser cannot interpret the style you've set up.

Conveniently, there are workarounds for this type of problem. Wise designers stack information into the style argument strings, and the browser will seek out the information it can interpret. This is good, but some control is lost as a result because the designer can never be absolutely certain that a page is going to remain consistent regardless of where it's viewed. As always, a designer should be careful and, whenever possible, test the results of his or her work by viewing it in a variety of browsers and on different platforms.

Style Sheet Essentials

Style sheets can be used in three primary ways: the inline method, the individual page or embedded method, and linking to a master or external style sheet.

- **Inline style sheets.** This approach exploits existing HTML tags within a standard HTML document and adds a specific style to the information controlled by that tag. An example would be controlling the indentation of a single paragraph using the `style="x"` attribute within the `<p>` tag. Another method of achieving this is with the `<span>` tag and the `style="x"` attribute combined.
- **Embedded style sheets.** This method allows the control of individual pages. It uses the `<style>` tag, along with its companion tag, `</style>`. This information is placed between the `<html>` tag and the `<body>` tag, with the style attributes inserted within the full `<style>` container.
- **External (linked) style sheets.** All that is required is to create a style sheet file with the master styles you want to express—using the same syntax you would with embedded style. This file uses the `.css` extension. Then, simply be sure that all of the HTML documents that require those controls are linked to the style sheet.

NOTE

With embedded and linked style sheets, the attribute syntax is somewhat different from standard HTML syntax. Attributes are placed within curly brackets; where HTML uses an equals sign (=), a colon (:) is used instead; and individual, stacked arguments are separated by a semicolon rather than a comma. Also, several attributes are hyphenated, such as `font-style` or `margin-left`. A style sheet string would then look like this: `{font-style: arial, helvetica; margin-left}`. Still, as with HTML, style sheet syntax is very structured. As you work with the examples in this chapter, you should become quite comfortable with the way style sheets work.

The following examples introduce you to the basics of each type. Although you'll be using some style sheet syntax to create the examples, detailed syntax will be covered later in the chapter.

Using Inline Style

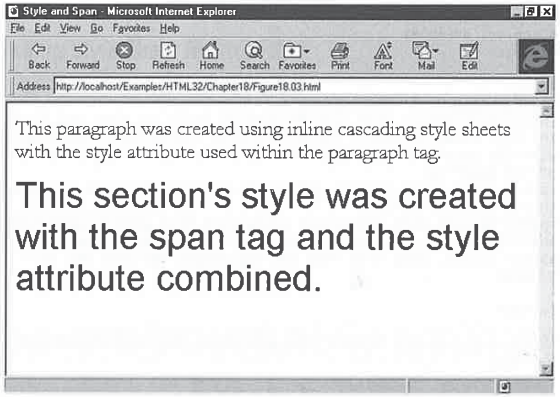
Creating inline style sheets is a simple matter of adding a style attribute to individual tags within the document.

Look at the following lines of HTML, paying attention to how the `style` attribute and the `<span>` tags are used:

```
<p style="font: 18pt garamond">
This paragraph was created using inline cascading style sheets
with the style attribute used within the paragraph tag.
</p>

<span style="font: 32pt arial">
This section's style was created with the span tag and the style
attribute combined.
</span>
```

When viewed on a browser that supports cascading style sheets, the lines appear as those in Figure 19.3.

FIGURE 19.3.
The results of inline style sheet commands.



TIP

The `<div>` (division) tag can be used like the `<span>` tag for inline control. The `<div>` tag is especially helpful for longer blocks of text, whereas `<span>` is most effective for adding style to smaller stretches of information, such as sentences, several words, or even individual letters within a word.

In a sense, the inline method of style sheet control defeats the ultimate purpose of cascading style sheets. The main point of the technology is to seek style control of entire pages or even entire sets of pages. The inline method treats the capabilities more like a <FONT> tag, and should be used only where touches of style are required.

Using the Individual Page Style

This section focuses on what is known as *embedding* a style sheet. Embedding sets the standards for an entire page. Look at the following basic document with embedded style, noting the addition of the <style> tags between the </head> and <body>.

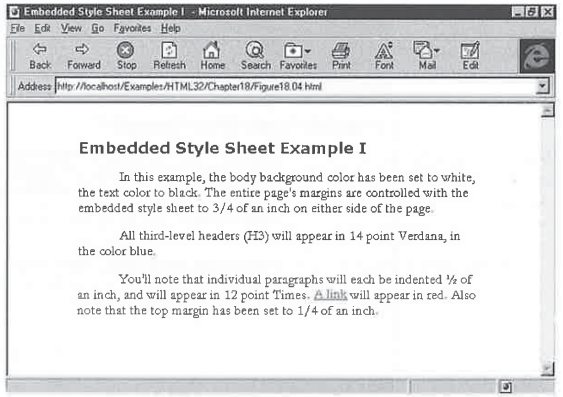
```
<html>

<head>
<title>Embedded Style Sheet Example I </title>
</head>

<style>
BODY {background: #FFFFFF; color: #000000;
margin-top:.25in; margin-left:.75in; margin-right:.75in}
H3 {font: 14pt verdana; color: #0000FF}
P {font: 12pt times; text-indent: 0.5in}
A {color: #FF0000}
</style>

<body>
<h3>Embedded Style Sheet Example I</h3>
<p>In this example, the body background color has been set to white,
the text color to black. The entire page's margins are controlled
with the embedded style sheet to 3/4 of an inch on either side
of the page.</p>
<p>All third-level headers (H3) will appear in 14 point Verdana,
in the color blue.</p>
<p>You'll note that individual paragraphs will each be indented _ of an inch,
and will appear in 12 point Times. <a href="other.htm">A link</a> will
appear in red. Also note that the top margin has been set to 1/4 of an inch.</p>
</body>

</html>
```

When displayed on a browser with style sheets, the result should look like the results in Figure 19.4.

Note the paragraph tags used as containers. Style sheets won't recognize a paragraph until there is an opening and closing paragraph tag. Moreover, if you choose to use inline style, you need to use the container code for paragraphs as well.

Embedded style sheets will probably be the type of style sheet you find yourself using most often. Because of their page-by-page control, embedded style sheets enable a designer to modify the look and feel of pages within a site. However, if strong uniformity is required, linking to a master style sheet is in order.

FIGURE 19.4.
The results of individual, or embedded, style sheet commands.



Creating the Cascading Style Sheet Master File

Creating linked style sheets is the focus of this section. This type of style sheet is a master document, controlling all documents that are linked to it. First, create a file that contains a set of <style> tags as well as any other style sheet definition:

```
<style>

BODY {background: #FFFFFF; color: #000000; margin-top:.25in;
margin-left:.75in; margin-right:.75in}
H3 {font: 14pt verdana; color: #0000FF}
P {font: 12pt times; text-indent: 0.5in}
A {color: #FF0000}

</style>
```

This file is saved with a .css extension. For this example, we'll call it style.css. This file can now be used to control the style attributes it defines, in as many individual HTML pages as you want to link to it.

After you've created the master file, you need to add a line to the document it affects so the styles can be used when the page is loaded. After you've opened the HTML file, place the following line somewhere between the <head> tags:

```
<link rel=stylesheet href="style.css" type="text/css">
```

All of the data within the actual html file will now interpret the styles you've set forth in the .css file. An example of an HTML page using externally linked style sheet controls is shown in Figure 19.5.

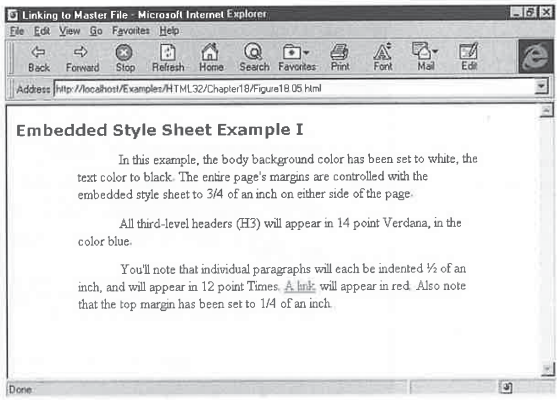


FIGURE 19.5.
This page calls on an externally linked style sheet for style control.

TIP

If your server requires you to register the MIME (Internet Media) type for style sheets, note that the suffix is .css, the MIME type is `text/css`, and it is considered an 8-bit (ASCII) file type.

Combining Style Sheet Techniques

Designers who have complex requirements will benefit from mixing style sheet techniques. Mixing takes full advantage of the cascade element of style sheets. When multiple techniques are combined, the browser looks for the information in the following cascading order:

- Linked style sheets will be employed globally.
- If an embedded style sheet exists on a page with a linked style sheet, the embedded style sheet will override a linked style sheet.
- Inline styles will override the preceding two points.

This way, you can set a single style sheet for an entire site, change a few pages here and there for individual style with embedded style, and use inline style to override both.

With so many sites, it's difficult to keep up with which style overrides which, and it can become confusing! Designers and developers using style sheets recommend that you select a favorite technique and stick with it; apply other style sheet techniques only if they are required for distinct assignments.

Text-Specific Style Attributes

Some graphic designers dedicate entire careers to the study of typography. It is undeniably an art, comprised of being familiar with countless font faces and related attributes such as weight, size variations, styles, families, and how to artistically use all of these aspects in attractive combinations.

The face of the Web can't change for the better if designers don't understand some text-based fundamentals before putting style sheets to use. The remainder of this chapter concentrates on the text and other formatting attributes that are available for use in all methods of style sheets. We'll begin by working with fonts:

- `font-family`. This attribute controls the face of the font by arguing for these items:
  - The name of the typeface you want to use. Here is a sample-style specification for Times:

    ```
    {font-family: Times}
    ```
  - The addition of a series of alternative fonts in those cases mentioned early in the chapter. Because an individual computer might not have the installed font of your preference, you might want to add a similar alternative:

    ```
    {font-family: times, garamond}
    ```
  - The name of a typeface family name:

    ```
    {font-family: serif}
    ```

  The following typefaces, or type *families*, can be called for with style sheets:
  - Serif. Serif fonts are a good choice for long sections of text. Popular serifs include Times New Roman and Garamond. Serifs are actually the little accents at the ends of the lines, which make up the letterform, such as the little horizontal line at the top and bottom of an uppercase *I*.
  - Sans Serif. This font family includes popular choices such as Arial, Helvetica, and Avante Garde. *Sans* means *without*, so sans serif faces don't have the little adornments.
  - Cursive. These are script fonts—fonts that appear as though they have been hand-written.
  - Fantasy. Fantasy fonts are decorative in nature and very useful for stylish headings and titles. They are typically not practical for body text.
  - Monospace. You're most likely to be familiar with monospace fonts from the days of the typewriter. Monospace means that every letter takes an equal amount of space. An *a* takes as much space as an *m*. Most fonts are proportional; each letter takes up a space that is proportional to the individual letter's size and style, rather than forcing it to fit in an exact amount of space.

NOTE

The standard design guideline for selecting serif versus sans serif fonts is to use serifs for body text, and use sans serifs for headers or small blocks of text.

However, although studies suggest that serif fonts are easier to read, sans serif fonts are becoming increasingly popular as common text fonts within Web browsers. It's an interesting phenomenon, but no one quite understands why it's occurring! Designers have to use good judgment, basing the use of serif or sans serif fonts on whether the pages are easy to read as well as attractive.

TIP

Using a typeface family as a default is an excellent idea, because it covers the designer's font choices as completely as possible. Even if a specific font face is unavailable on a given computer, it's likely that a similar one in that font's family is available. A savvy designer will place his or her first choice first, second choice second, and so forth, with the family name at the end. If I'm arguing for two serif fonts, my final string would be `{font-family: times, garamond, serif}`.

- `font size`. Sizing in style sheets gives the designer the choice to size his or her fonts using five size options:
  - Points. To set a font in point size, use the abbreviation `pt` immediately next to the numeric size:
    `{font-size: 12pt}`.
  - Inches. If you'd rather set your fonts in inches, simply place `in` and the numeral size, in inches, of the font size you require:
    `{font-size: 1in}`.
  - Centimeters. Some designers might prefer centimeters, represented by `cm` and used in the same fashion as points and inches:
    `{font-size: 5cm}`.
  - Pixels. Pixels are argued with the `px` abbreviation:
    `{font-size: 24px}`.
  - Percentage. You might want to set a percentage of the default point size:
    `{font-size: 50%}`

  Designers will most likely find themselves more comfortable with the point values for setting font sizes. However, if you prefer another method, that's fine. If you choose a method, stick to it. Consistency is a smart approach to the creation of your own individual design and coding style.

- `font-style`. This attribute typically dictates the style of text, such as placing it in italics. The appropriate syntax is as follows:

  `{font-style: italic}`

  Another font style is bold, but, interestingly, no current style sheet attribute can achieve this. The only other legal attribute for style is `normal`, which simply places the typeface in normal default status. It's unclear when support for more font styles will become available, although it seems only natural that they will at some point be added to the specifications.
- `font-weight`. The thickness of a typeface is referred to as its *weight*. As with font faces, font weights rely on the existence of the corresponding font and weight on an individual's machine. A range of attributes is available in style sheets, including the following:

  ```
  extra-light
  demi-light
  light
  medium
  extra-bold
  demi-bold
  bold
  ```

  Before assigning font weights, make sure that the font face to which you are applying the weight has that weight available. Always check your work on a variety of platforms and machines to see whether you can achieve strong design despite the fact that some machines might not support the font or the font weight in question.
- `text-decoration`. This attribute decorates text with options such as the following:

  ```
  none
  underline
  italic
  line-through
  ```

NOTE

With cascading style sheets, designers can use the `{text-decoration: none}` attribute and argument to globally shut off underlined links. In embedded and linked style sheet formats, the syntax would follow the `A` value: `A {text-decoration: none}`. For inline style, simply place the value within the link you want to control: `<a style="text-decoration: none" href="my.html">this link has no underline</a>`.

- `line-height`. To set the leading of a paragraph, use the line-height attribute in points, inches, centimeters, pixels, or percentages in the same fashion you would when describing `font-size` attributes:

  ```
  P {line-height: 14pt}
  ```

  This attribute is the same as *leading*, which refers to the amount of line spacing between lines of text. This space should be consistent, or the result is uneven, unattractive spacing. The line-height attribute allows designers to set the distance between the baseline, or bottom, of a line of text.

After learning some of the basics, it's time to see how style sheets are actually used.

Using Text Attributes in Embedded Style Sheets

Now you can employ attributes discussed in the previous section in an embedded style sheet. Begin with a fresh page in your HTML editor, and a set of basic structure tags with `<style>` tags.

```
<html>

<head>
<title>Text Style Example: Embedded Style Sheets</title>
</head>

<style>
</style>

<body bgcolor="#ff9933">
</body>

</html>
```

Then, add the style syntax between the `<style>` tags. You'll see the font family, size, weight, and style specified for headers; family, size, and style in paragraphs; and underlining removed from links with the text-decoration attribute. Note the stacking of font faces and families to ensure as close a match as possible to the intended style:

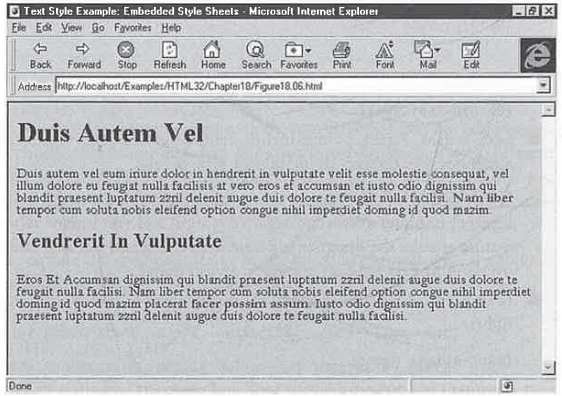
```
H1 {font-family: lucida handwriting, arial, helvetica, cursive, san-serif ;
➥font-size: 16pt; font-style: normal}
H2 {font-family: lucida handwriting, arial, helvetica, cursive, san-serif;
➥font-size: 14pt; font-style: normal}
P {font-family: garamond; font-size: 12pt; font-style: normal;
➥line-height: 11pt}
A {text-decoration: none; font-weight: bold}
```

With the style in place, you can add the body of the document. I've included some dummy content just for illustration purposes.

```
<h1>Duis Autem Vel</h1>

<p>
Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie
consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan
et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis
dolore te feugait nulla facilisi. <a href="other.htm"> Nam liber </a> tempor
cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim.
</p>

<h2>Vendrerit In Vulputate</h2>

<p>
Eros Et Accumsan dignissim qui blandit praesent luptatum zzril delenit augue
duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis
eleifend option congue nihil imperdiet doming id quod mazim placerat
<a href="other1.htm"> facer possim assum. </a>  Iusto odio dignissim qui blandit
praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.
</p>
```

When the file is viewed on a browser, you can see how the headers, paragraphs, and links have obeyed the style sheet's commands, as demonstrated in Figure 19.6.

FIGURE 19.6.
This is an example of using an embedded style sheet. A variety of text controls are applied to the text on this page.



NOTE

The gibberish used in place of text is referred to as *greeking*. It's not really Greek, but a latinesque style of word placement designers use when creating mock-ups and dummy text.

Remember that a linked style sheet is simply the information in an embedded style sheet placed in its own file and given the extension `.css`. Then, all pages that you want to draw from the information in that file are simply linked to that file.

Now you have a page that has successfully employed simple embedded style sheets. The next section covers margin, indent, and text-alignment attributes to help add texture and white space to the page.

Margins, Indents, and Text-Alignment Attributes

All critical elements of controlling page layout, margins, indents, and text alignment can help bring a sophisticated look to your pages. Here are some examples:

- `margin-left`. To set a left margin, use a distance in points, inches, centimeters, or pixels. The following sets a left margin to ¾ of an inch:

  ```
  {margin-left: .75in}
  ```

- `margin-right`. For a right margin, select from the same measurement options provided for the `margin-left` attribute:

  ```
  {margin-right: 50px}
  ```

- `margin-top`. Top margins can be set using the same measurement values used for other margin attributes:

  ```
  {margin-top: 20pt)
  ```

- `text-indent`. Again, points, inches, centimeters, or pixel values can be assigned to this attribute, which serves to indent any type of text:

  ```
  {text-indent: 0.5in)
  ```

  Internet Explorer allows for negative margin and text indent values. This exciting feature enables the designer to create interesting and unusual effects, including overlapping text for contemporary, stylish design.

- `text-align`. This feature allows for justification of text. Values include left, center, and right:

  ```
  {text-align: right}
  ```

Text alignment is a powerful layout tool, and designers will enjoy being able to place text in a variety of alignments without having to rely on tables, divisions, or other less graceful HTML workarounds (they still exist). Designers should remember that justification of text requires a fine eye. Left justification is the only reasonable choice for long selections of text, because it is much more readable. Right justification comes in handy for short bursts of text, such as pull-quotes. Centered text should be used sparingly. Even though it seems natural to want to center text, it is actually more difficult to read.

To add margin control to the previous example, add the following line to the beginning of the style section:

```
BODY {margin-left: 0.75in; margin-right: 0.75in; margin-top: 0.10in}
```

The new file should match what you see in Figure 19.7.

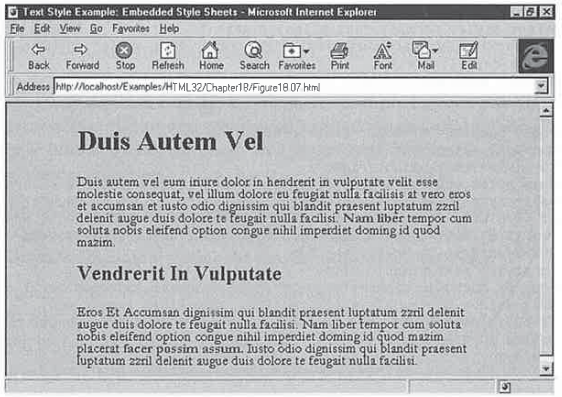


FIGURE 19.7.
Margins create attractive white space.

Although margin values are added to the entire page with the `BODY` attribute, you can add margins to any HTML tag you choose. For example, if you want to control the headers with different margins, place the margin values in the string next to the header of your choice. Similarly, you can adjust margins on individual paragraphs by adding the margin values you seek to the paragraph string.

For a practical example, replace the style definitions in the previous example with the following attributes, which include margin and justification syntax for the borders of the page:

```
BODY {margin-left: 0.75in; margin-right: 0.75in; margin-top: 0.10in}
H1 {font-family: lucida handwriting, arial, helvetica ; font-size: 16pt;
➥font-style: normal; text-align: left}
H2 {font-family: lucida handwriting, arial, helvetica; font-size: 14pt;
➥font-style: normal; text-align: right}
P {font-family: garamond; font-size: 12pt; font-style: normal;
➥line-height: 11pt}
A {text-decoration: none; font-weight: bold}
```

Because individual paragraphs should be justified in this example, that information is placed inline, and the browser will know what to do. The full HTML code looks like this:

```
<html>

<head>
<title>Text Style Example: Embedded Style Sheets</title>
</head>

<style>
BODY {margin-left: 0.75in; margin-right: 0.75in; margin-top: 0.10in}
H1 {font-family: lucida handwriting, arial, helvetica ; font-size: 16pt;
➥font-style: normal; text-align: left}
```

```
H2 {font-family: lucida handwriting, arial, helvetica; font-size: 14pt;
➥font-style: normal; text-align: right}
P {font-family: garamond; font-size: 12pt; font-style: normal;
➥line-height: 11pt}
A {text-decoration: none; font-weight: bold}
</style>

<body bgcolor="#ff9933" link="FF0033">
<h1>Duis Autem Vel</h1>
<p style="text-align: right">
Duis autem vel eum iriure dolor in hendrerit in vulputate <a href="other.htm">
velit esse molestie </a>consequat, vel illum dolore eu feugiat nulla facilisis
at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum
 zzril delenit augue duis dolore te feugait nulla facilisi.</p>
<h2>Vendrerit In Vulputate</h2>
<p style="text-align: left">
Eros Et Accumsan dignissim qui blandit praesent luptatum zzril delenit augue
duis dolore te feugait nulla facilisi. Nam liber tempor cum
<a href="other1.htm">soluta nobis eleifend </a> option congue nihil imperdiet
doming id quod mazim placerat facer possim assum.</p>
</body>

</html>
```

The look and feel of this page is becoming much more interesting, as shown in Figure 19.8.

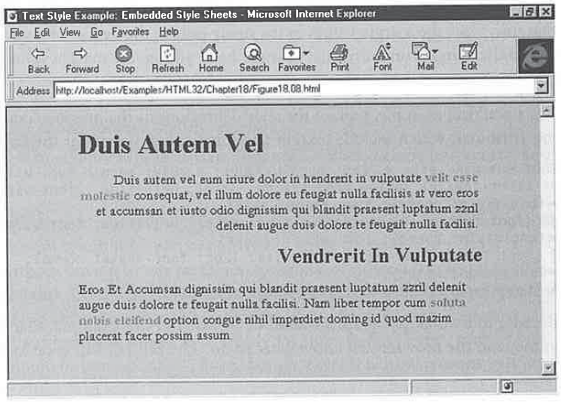


FIGURE 19.8.
Visual texture makes a Web page more interesting, even without graphics.

Another method to justify individual paragraphs is by defining classes. I'll introduce style sheet class variations toward the end of this chapter.

Color and Background

If you've checked these pages with the examples on the CD-ROM, or noticed the `<body>` tag syntax, you have noticed the background color defined within traditional HTML rather than in the `BODY` attribute available to style sheets. Internet Explorer 3.0+ has a bug in the software that results in the browser ignoring this basically legal attribute, and the bug appears to also exist in Netscape's 4.0 pre-release. As a workaround, I define a background color for the body in a traditional fashion.

All bugs and browser headaches aside, background color can be added to actual attributes including other HTML tags used in style sheet formats. For example, you can have a paragraph or a header splashed with color by simply placing the appropriate syntax in that area.

Furthermore, you can change the text color in any field you specify. This is particularly satisfying for Web designers who are constantly seeking to employ browser-based color to enliven pages rather than relying on time-consuming graphics.

The syntax required to create background color—again, with the one exception that Internet Explorer versions below 4.0 ignore the attribute if placed in the `BODY` string—is the style sheet `background:` convention and a hexadecimal triplet color argument:

```
{background: #FFFFFF}
```

Similarly, background graphics can be called upon using this attribute. Merely replace the hex argument with a URL:

```
{background: http://myserver.com/cool.gif}
```

For text color, simply use the `color` attribute and a hex argument:

```
{color: #FF6633}
```

Internet Explorer allows the use of color names in the case of `color` and `background` arguments. Such color names include black, silver, gray, white, maroon, red, purple, fuchsia, green, lime, olive, yellow, navy, blue, teal, and aqua. My advice is to stick to hexadecimal values. Not only do you have a better selection, it also helps keeps your coding consistent and professional-looking.

To add splashes of background color to fields in the file you've created, use the following style syntax in our ongoing sample file. The result is shown in Figure 19.9:

```
BODY {margin-left: 0.75in; margin-right: 0.75in; margin-top: 0.10in}
H1 {font-family: lucida handwriting, arial, helvetica; font-size: 16pt;
font-style: normal; text-align: left}
H2 {font-family: lucida handwriting, arial, helvetica; font-size: 14pt;
font-style: normal; text-align: right; background: #99CCCC}
P {font-family: garamond; font-size: 12pt; font-style: normal;
line-height: 11pt}
A {text-decoration: none; font-weight: bold}
```

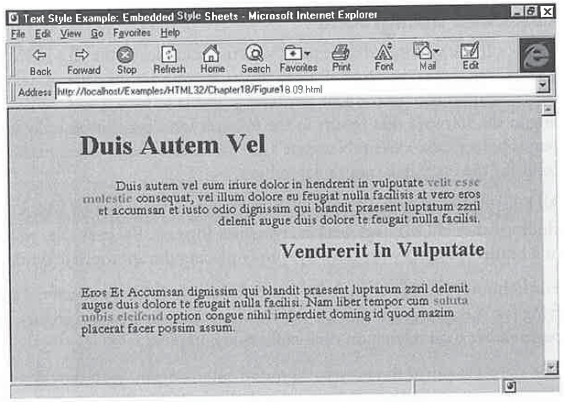


FIGURE 19.9.
Notice the splash behind the text. Is it distracting or enhancing? Designers must choose background splashes carefully.

TIP

You'll need to view these examples on a computer with a color monitor to see the full color effect.

Contrast gained by using background splashes can be very effective, but designers can quickly get in trouble by overusing them. Be careful and consistent with your color palettes, and select splashes that provide visual accents rather than detract the eye from what is important. You'll note that I selected the header to highlight; in another case, maybe only a line or phrase of text would be important.

To liven up sections of text, follow the same style guidelines to set for background color, but use the `color:` attribute:

```
H2 {font-family: lucida handwriting, arial, helvetica; font-size: 14pt;
➥font-style: normal; text-align: right;
background: #99CCCC; color: #FF0033}
```

View it in your browser. You now have an interesting, textured page completely designed with style sheets!

Additional Functionality

Two additional techniques can assist you in making style sheets more functional. The first is *grouping*, which allows for the reduction of attributes and arguments by creating logical groups. Another way to expand function is through the use of *classes*. This technique allows you to assign variations to individual HTML tags, giving you tremendous flexibility in terms of creating variations within page attributes.

Grouping Style Sheets

To group style sheets, you can follow these guidelines:

- Group multiple tags together. If you want to assign the same attributes to all header styles, you can group them together. Here's an example without grouping:

```
H1 {font-family: arial; font-size 14pt; color: #000000}
H2 {font-family: arial; font-size 14pt; color: #000000}
H3 {font-family: arial; font-size 14pt; color: #000000}
```

  Here's the same example grouped:

```
H1, H2, H3 {font-family: arial; font-size 14pt; color: #000000}
```

- Group attributes by dropping them into specific families of information. Without grouping, an example of font attributes and arguments would look like this:

```
BODY {font-family: arial, san-serif; font-size: 12pt;
➥line-height: 14pt; font-weight: bold; font-style: normal}
```

  With grouping, I can simply name the attribute `font:` and then stack the arguments like this:

```
BODY {font: bold normal 12pt/14pt arial, san-serif}
```

When grouping attributes, be sure to remember that attribute order is significant. Font weight and style must come before other font attributes; the size of the font will come before the leading, and then you can add additional information to the string. Note that there are no commas between the attributes, except in the case of font families.

Grouping attributes can be done with margins, using the `margin:` argument followed by the top, right, and left margin values in that order. Be sure to specify all three values, unless you want the same value applied to all three:

```
BODY {margin: .10in .75in .75in}
```

Note again that there are no commas between the attributes.

Assigning Classes

To get the most variation in style, assign classes to individual HTML tags. This is done very simply by adding a named extension to any HTML tag. If you have two headers and two paragraph styles that you want to attribute, you can name each one and assign style to the individual paragraphs. You then call on the name within the specific HTML tag in the body of the document:

```
H1.left {font: arial 14pt; color: #FF0033; text-align: left}
H2.right {font: arial 12pt; color: #FF6633; text-align: right}
```

In the HTML, you would place the class name:

```
<h1 class=left>This is my Left Heading</h2>
```

All of the H1 headers you name class=left will have the H1.left class attributes. Similarly, the H2.right headers named class=right will have the attributes defined for that class.

Grouping and class allow for very flexible use of style sheets. Begin with the following style definition in the sample file:

```
BODY {margin: 0.10in 0.50in 0.50in}
H1.left {font: 16pt lucida handwriting; text-align: left}
H2.right {font: 14pt lucida handwriting; text-align: right; color: #FF0033}
P.left {font: 12pt/11pt garamond; text-align: left}
P.right {font: 12pt arial; text-align: right; margin: 0in .75in .50in}
A {text-decoration: none; font-weight: bold}
```

Then, within each of the appropriate tags, add the class names to complete the process:

```
<H1 class=left>Duis Autem Vel</h1>

<p class=right>
Duis autem vel eum iriure dolor in hendrerit in vulputate <a href="other.htm">
velit esse molestie </a>consequat, vel illum dolore eu feugiat nulla
facilisis at vero eros et accumsan et iusto odio dignissim qui blandit
praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.
</p>

<h2 class=right>Vendrerit In Vulputate</h2>

<p class="left">
Eros Et Accumsan dignissim qui blandit praesent luptatum zzril delenit augue
duis dolore te feugait nulla facilisi. Nam liber tempor cum
<a href="other1.htm">soluta nobis eleifend</a> option congue nihil imperdiet
doming id quod mazim placerat facer possim assum.
</p>
```

An incompatible browser will ignore the attributes it doesn't recognize, while a browser such as Internet Explorer 4 will display the file shown in Figure 19.10.

If you're placing margins in the BODY attribute as well as in the P attribute, be sure your paragraph margins are larger than those you've selected for the entire body of text. Otherwise, the point is moot! The browser will ignore the lesser margins and use the greatest available value.

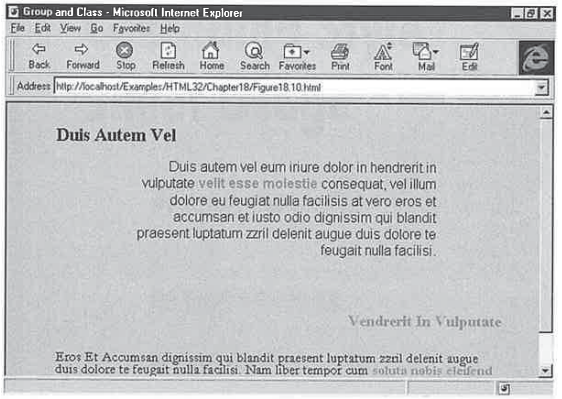


FIGURE 19.10.
The resulting page is a collection of style that defies the standard HTML.

Summary

Style sheets are both sophisticated and infantile in their behavior. The concept is extremely advanced, allowing designers to do things never dreamed possible in the young history of HTML. However, the actual application is in its infancy. As with many layout options available to designers, this is more the fault of browser technology than the reality of the tool.

If you begin to use style sheets to work with your design, you are certain to be challenged by the limitations of browsers. That problem will last for some time, but invariably in your favor is the fact that you started learning and using the technology early, giving you an edge. I encourage you to think about the power of the system rather than the limitations, because those limitations will gradually subside. Your skills can only improve with time and practice.

Moreover, there's the thrill of being able to actually design without ever opening up a graphics program. Color, font, text control—they're all yours now to use and enjoy. The next chapter gives a few more examples and moves into page design with style sheets for you to examine and work from. This includes mastering text-heavy documents with style sheet control.

CHAPTER 20

Cascading Style Sheet Usage

by Molly Holzschlag

IN THIS CHAPTER

Because style sheets are so flexible, this chapter of the book is included so that you can spend a little time designing a variety of templates for a hands-on feel. More importantly, these designs are applicable to a range of common Web design jobs. By changing some of the style attributes, such as fonts, colors, alignment, headers, or paragraph styles, you can stock up on a very diverse collection of style sheet–based offerings.

All of these examples use embedded style sheets. In some cases, the style is extrapolated so that you can see how to make a linked style sheet to control numerous pages with the same style elements. There's a lot of variation within the style sheets themselves. Sometimes tables are used to lay out a page, and other times the page is a straightforward HTML document. This chapter also includes demonstrations on the use of grouping and class, as well as examples of how to override style sheet calls by reverting to traditional HTML tags.

TIP

It cannot be said enough: Know your audience. This will help determine how much you can realistically employ style sheets.

This chapter should serve not only as a practical taste of what style sheets can do, but also as inspiration to spend time on your own using style sheets in creative design applications. You will see how to use classes to create an untold variety of styles within the same page.

Business Page

The first template is a business page. This can be used for commercial business purposes or for an intranet application. The design consists of a splash of color on the left margin that contains a summary or subtitle, with the headline and text on the right. The headline color matches the background color on the left to add an extra sense of consistency to the page.

The style container to create this page looks like this:

```
<style>
BODY {background: #CCCCCC; color: #000000; margin-top:0.00in;
➥margin-left: 0.20in; margin-right:0.20in}
H1 {font: 24pt Garamond; color: #669966; text-align: right}
P {font: 12pt Verdana; color: #FFFFFF; text-align: right;
➥text-indent: 0.5in}
P.1 {font: 11pt verdana; color: #000000; text-align: right;
➥text-indent: 0.5in}
</style>
```


CAUTION

A bug in both Internet Explorer 3.0 and Navigator 4.0 causes the browser to ignore body colors in the style sheet. To be safe, you must put the color in the <body> tag. It's a good idea to put the color attributes in the style sheet as well, because the problem is in fact a bug and not a style sheet idiosyncrasy. It's always a good idea to do both anyway. Many people don't upgrade browsers in a timely fashion, and reasonable backwards compatibility is always a good idea.

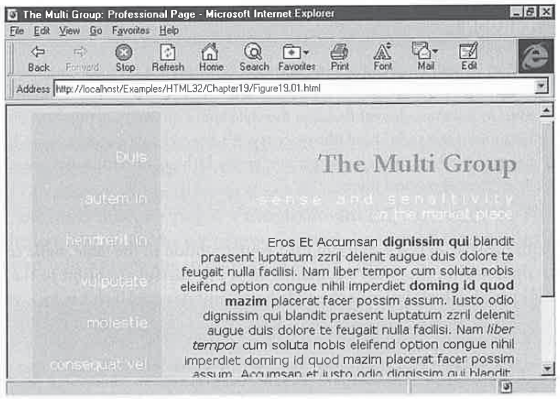
With the style in place, you can turn your attention to the table, table cells, and content. Place all three within the <BODY> container tags. Pay careful attention to the use of table cell color definitions, which override the style background just as if this were a standard HTML page.

```
<table border=0 width=575 cellpadding=10 cellspacing=0>
  <tr>
    <td valign=top width=150 bgcolor="#99CC99">
      <p style="line-height: 0.5in">Duis autem in hendrerit in vulputate
       molestie consequat vel feugiat nulla facilisis
    </td>
    <td valign=middle width=425>
      <h1>The Multi Group</h1>
      <p>s e n s e . a n d . s e n s i t i v i t y<br>
      <i>in</i> the market place</p>
      <p class=1>
       Eros Et Accumsan <b>dignissim qui</b> blandit praesent
       luptatum zzril delenit augue duis dolore te feugait
       nulla facilisi. Nam liber tempor cum soluta nobis
       eleifend option congue nihil imperdiet<b> doming id
       quod mazim</b> placerat facer possim assum. Iusto odio
       dignissim qui blandit praesent luptatum zzril delenit
       augue duis dolore te feugait nulla facilisi. Nam
       <i>liber tempor </i>cum soluta nobis eleifend option
       congue nihil imperdiet doming id quod mazim placerat
       facer possim assum. Accumsan et iusto odio dignissim
       qui blandit. Eros Et Accumsan dignissim qui blandit
       praesent luptatum zzril delenit augue duis dolore
       <b>te feugait nulla</b> facilisi. Nam <i>liber
       tempor</i> cum soluta nobis.</p>
    </td>
  </tr>
</table>
```

View the assembled product in a style sheet–compliant browser. Figure 20.1 shows the results.

The Multi Group example is very consistent with a splash or front page. Suppose you are designing different levels of a company's Web site. You can use this style at each level, embedding the same style in those specific pages, or linking those pages to a master style sheet.

FIGURE 20.1.
A business-oriented splash page.



TIP

Don't forget the limitation of font styles. Not all users will have the same fonts on their computers that you have on yours. These examples are based on using fonts in one machine—mine. You'll need to look at the fonts on your system for compatibilities that will complement the style.

Employee Newsletter

The following steps help you to implement a simple HTML newsletter layout, which relies on justification to create an interesting and dynamic look. Each story is separated by a horizontal rule, and the headings alternate between the left and right margins.

The embedded style sheet focuses on color and alignment implemented through classes:

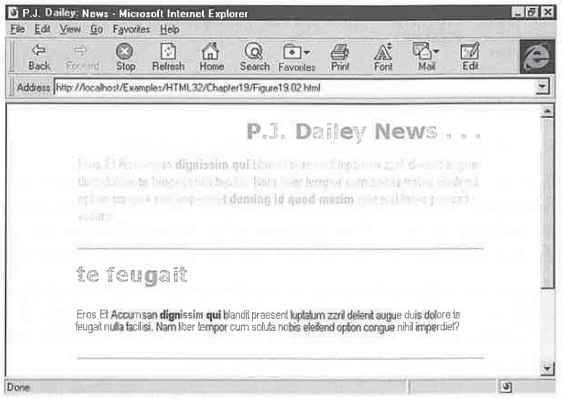
```
<style>

BODY: {color: #000000; margin-left: 0.75in; margin-right: 0.75in;
➥margin-top: 0.00in}
H1.right {font: 18pt "lucida handwriting"; color: #999966;
➥text-align: right}
H1.left {font: 18pt "lucida handwriting"; color: #999966;
➥text-align: left}
P {font: 12pt "arial narrow"; color: #CCCC99; margin-indent: 0.5in}
P.1 {font: 11pt/11pt "arial narrow"; color: #FFFFFF}
HR {color: #CCCC99}

</style>
```


The styles are implemented in the body without the use of tables or other tricks to force the formatting.

```
<h1 class=right>P.J. Dailey News . . .</h1>
<p>Eros Et Accumsan <b>dignissim qui</b> blandit praesent
luptatum zzril delenit augue duis dolore te feugait
nulla facilisi. Nam liber tempor cum soluta nobis
eleifend option congue nihil imperdiet<b> doming id
quod mazim</b> placerat facer possim assum.</p>
<hr>
<h1 class=left>te feugait</h1>
<p class=1>Eros Et Accumsan <b>dignissim qui</b> blandit
Praesent luptatum zzril delenit augue duis dolore te feugait
nulla facilisi. Nam liber tempor cum soluta nobis
eleifend option congue nihil imperdiet?</p>
<hr>
<h1 class=right>liber tempor</h1>
<p>Eros Et Accumsan <b>dignissim qui</b> blandit praesent
luptatum zzril delenit augue duis dolore te feugait
nulla facilisi. Nam liber tempor cum soluta nobis
eleifend option congue nihil imperdiet<b> doming id
quod mazim</b> placerat facer possim assum.</p>
```

View the assembled results in your browser, which should match the example in Figure 20.2.

FIGURE 20.2.
An interoffice newsletter example.



The P.J. Dailey page provides a great example of how to combine headers and body text. With very few exceptions, you don't want to separate a header from the related text with a horizontal rule. This is a mistake that many people make. Furthermore, this example shows how horizontal rules can successfully be used without appearing cliché.

The addition of style breaks the typical, and ugly, use of horizontal rules on the Web. When you define color and have plenty of white space, the problems seen with most rules disappear. At all costs, avoid using horizontal rules that stretch from margin to margin. Also, the use of headers and text together creates a rationale for not using the rules at all. If a paragraph break is what you're after, use extra spacing between the paragraph and the next line of text—whether a header or another paragraph.

Creating a Linked Style Sheet

Because the P.J. Dailey example could be applied to many pages, I'll show you how to create a linked page. All you need to do then is link the individual pages to this page, and all pages will pick up the styles defined within the sheet.

Begin by taking the embedded style out of the originating page, and place it into a file by itself:

```
<style>
BODY: {color: #000000; margin-left: 0.75in; margin-right: 0.75in;
➥margin-top: 0.00in}
H1.right {font: 18pt "lucida handwriting"; color: #999966;
➥text-align: right}
H1.left {font: 18pt "lucida handwriting"; color: #999966;
➥text-align: left}
P {font: 12pt "arial narrow"; color: #CCCC99; margin-indent: 0.5in}
P.1 {font: 11pt/11pt "arial narrow"; color: #FFFFFF}
HR {color: #CCCC99}
</style>
```

Save the file as `pj.css`. Then place the following link within the `<HEAD>` tag on every page you want to have influenced by this style:

```
<link rel=stylesheet href="pj.css" type="text/css">
```

Because you're linking styles, it's not necessary to embed this information in any of the individual pages that will be influenced by the link. However, if you want to alter individual pages that will take on most of the attributes of the linked style sheet yet contain a few individual changes, you can embed a style sheet into that page with the minor changes.

If you want to alter only one or two smaller sections in a handful of pages, drawing from the linked page and embedded style, you can use inline style to cover that smaller section. The CSS-compliant browser will read inline before embedded and embedded before linked, resulting in a cascading implementation of your style.

Price List

The following example uses a bordered table within the page as a grid. Pay specific attention to the classes defined for the table cells. This further demonstrates the power and flexibility of style sheets and their ability to influence any of the HTML tags. Here's the style definition:

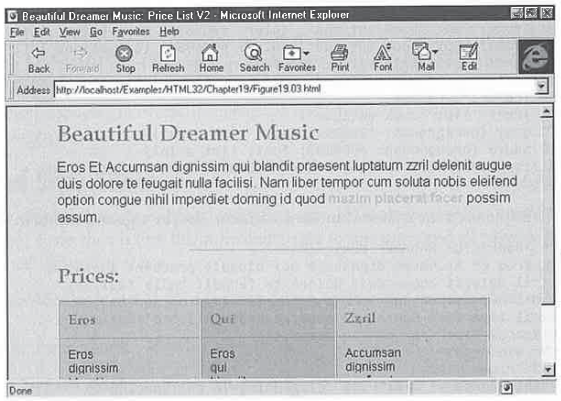
```
<style>
BODY {background: #CCCC99; color: #000000; margin-left: 0.5in;
➥margin-right: 0.5in}
H1 {font: 20pt "book antiqua"; color: #666633; text-align: left}
H2 {font: 18pt "book antiqua"; color: #666633; text-align: left}
P {font: 12pt arial; color: #000000}
B {font: 12pt "book antiqua"; color: #666633; font-weight: bold}
HR {color: #CC9999}
TD {font: 11pt "book antiqua"}
TD.grey {background: #999999; font: 11pt arial}
TD.mauve {background: #CC9999; font: 11pt arial}
A {text-decoration: none; color: #CC9999; font-weight: bold}
</style>
```

To implement the style within the document, use the tags and classes in the following manner:

```
<h1>Beautiful Dreamer Music</h1>
<p>Eros Et Accumsan dignissim qui blandit praesent luptatum
zzril delenit augue duis dolore te feugait nulla facilisi.
Nam liber tempor cum soluta nobis eleifend option congue
nihil imperdiet doming id quod <a href="nil.htm">mazim
placerat facer</a> possim assum.</p>
<hr width=45%>
<h2>Prices:</h2>
<table border=1 width=500 cellpadding=10 cellspacing=0>
  <tr>
    <td class=mauve width=50><b>Eros</b></td>
    <td class=grey width=50><b>Qui </b></td>
    <td class=mauve width=50><b>Zzril </b></td>
  </tr>
  <tr>
    <td class=mauve width=50>
      Eros<br>
      dignissim <br>
      blandit <br>
      luptatum
    </td>
    <td class=grey width=50>
      Eros<br>
      qui <br>
      blandit <br>
      luptatum
    </td>
    <td class=mauve width=50>
      Accumsan<br>
      dignissim <br>
      praesent <br>
      zzril
    </td>
  </tr>
  <tr>
    <td colspan=3 align=right>
      <i>eros et accumsan dignissim, st.</i>    Total: $18,000.00
    </td>
  </tr>
</table>
</div>
```

Save the completed file and compare it with the example in Figure 20.3.

FIGURE 20.3.
Price sheet page, version 1.



Price List in a Reverse Style

Now let's have some fun by reversing the headers and body text of this example. The following steps show how dramatic variation can be achieved quite simply with the same style sheet with the following changes to the style:

```
<style>
BODY {background: #CCCC99; color: #000000; margin-left: 0.5in;
➥margin-right: 0.5in}
H1 {font: 20pt "arial"; color: #666633; text-align: left}
H2 {font: 18pt "arial"; color: #666633; text-align: left}
P {font: 12pt "book antiqua"; color: #000000}
B {font: 12pt "arial"; color: #666633; font-weight: bold}
HR {color: #CC9999}
TD {font: 11pt "arial"}
TD.grey {background: #999999; font: 11pt "book antiqua"}
TD.mauve {background: #CC9999; font: 11pt "book antiqua"}
A {text-decoration: none; color: #CC9999; font-weight: bold}
</style>
```

Next, apply the style to the body of the document:

```
<body bgcolor="#CCCC99">
<h1>Beautiful Dreamer Music</h1>
<p>Eros Et Accumsan dignissim qui blandit praesent luptatum zzril delenit augue
duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis
```

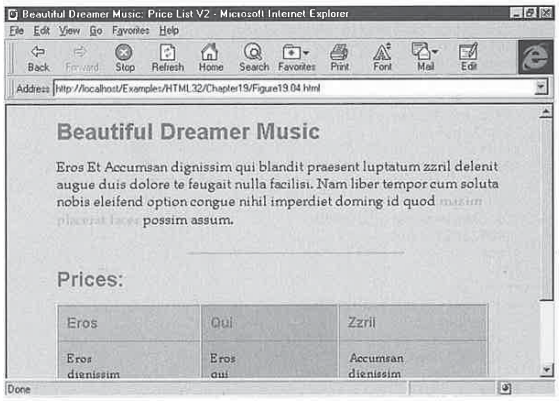


```
eleifend option congue nihil imperdiet doming id quod <a href="nil.htm">mazim
placerat facer</a> possim assum.</p>
<hr width=45%>
<h2>Prices:</h2>
<table border=1 width=500 cellpadding=10 cellspacing=0>
  <tr>
    <td class=mauve width=50>
      <b>Eros</b>
    </td>
    <td class=grey width=50>
      <b>Qui </b>
    </td>
    <td class=mauve width=50>
      <b>Zzril </b>
    </td>
  </tr>
  <tr>
    <td class=mauve width=50>
      Eros<br>
      dignissim <br>
      blandit <br>
      luptatum
    </td>
    <td class=grey width=50>
      Eros<br>
      qui <br>
      blandit <br>
      luptatum
    </td>
    <td class=mauve width=50>
      Accumsan<br>
      dignissim <br>
      praesent <br>
      zzril
    </td>
  </tr>
  <tr>
    <td colspan=3 align=right>
      <i>eros et accumsan dignissim, st.</i>    Total: $18,000.00
    </td>
  </tr>
</table>
</div>
```

Save the file and view it in your browser. It should appear similar to the example in Figure 20.4.

Some people use comment tags around their style sheet calls. This hides the style sheets from non-style sheet browsers. You can then design the page to be viewable in a non-compliant browser, but still allow for readability and some semblance of style.

FIGURE 20.4.
Price sheet page, version II.



Creating a Party Invitation Using Style Sheets

Want to get colorful? The following tags use style sheets to achieve a fun and colorful design without the use of any graphics:

```
<style>
BODY {color: #FFFFFF; margin: 0.0 0.05 0.05}
H1 {font: 25pt "LasVegasD"; color: #FF3333; text-align: right}
H2 {font: 20pt "LasVegasD"; color: #FF3333; text-align: left}
P {font: 13pt "verdana"; color: #333399}
HR {color: #FF3333}
A {color: #FFCC33; text-decoration: none}
</style>
```

The page is formatted with a table in addition to the application of styles:

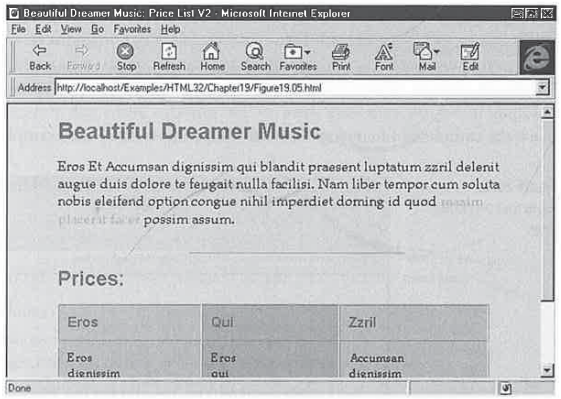
```
<table border=0 width=580 cellpadding=10 cellspacing=0>
  <tr>
    <td width=400 bgcolor="#FFFF00">
      <h1>Viva Las Vegas Party!!!</h1>
      <p><b>Las Vegas-style</b> party for all employees of AJbest Books. Eros
       Et Accumsan dignissim qui blandit praesent luptatum zzril delenit augue
       Duis dolore te feugait nulla facilisi.</p>
       <p><b>Nam: </b>liber tempor cum soluta nobis</p>
       <p><b>Nam: </b>liber tempor cum soluta nobis</p>
       <p><b>Nam: </b>liber tempor cum soluta nobis</p>
    </td>
    <td width=170>
      <h2>win big . . .</h2>
      <p align=right>Eros Et Accumsan dignissim qui
```



```
       <a href="http://win.htm/">blandit praesent</a> luptatum zzril
       delenit augue duis dolore te feugait nulla facilisi. Nam liber
       tempor!</p>
      <hr>
      <p align=right><i>email: <a href="mailto:helene@ajbooks.biz">
       helene@ajbooks.biz</a></i>
    </td>
  </tr>
</table>
```

Save the file and view it in your browser. Figure 20.5 shows the layout of this bright, online invitation.

FIGURE 20.5.
A business party invitation.



If you're beginning to be intrigued by the possibilities, you might want to check out some additional references and examples. For an official view, look at what the World Wide Web Consortium has to say at `www.w3.org/ub/WWW/Style/`, which includes a large list of links to other sites. Another good resource is located at `www.htmlhelp.com/reference/css/references.html`.

Kid's News

Here is another example of a standard HTML design using rules that creates an attractive, vivacious, and very fun page through the use of a few basic styles:

```
<style>
BODY {color: #FFFFFF; margin: 0.05in 0.75in}
H1 {font: 20pt "Kids"; color: #663366; text-align: right}
H2 {font: 18pt "Kids"; color: #663366; text-align: left}
```

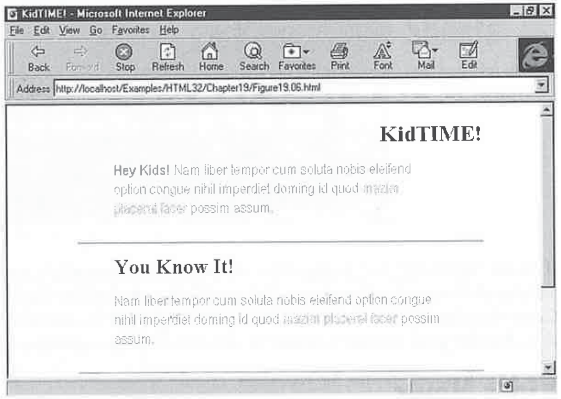
```
P {font: 13pt "arial narrow"; color: #009933}
HR {color: #FF9900}
A {color: #FF9900; text-decoration: none}
</style>
```

Next, create the layout and content with HTML in the body:

```
<h1>KidTIME!</h1>
<blockquote>
<p><b>Hey Kids!</b> Nam liber tempor cum soluta nobis eleifend option congue
nihil imperdiet doming id quod <a href="nil.htm">mazim placerat facer</a>
possim assum.</p>
<hr>
<h2>You Know It!</h2>
<p>Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet
doming id quod <a href="nil.htm">mazim placerat facer</a> possim assum.</p>
<hr>
<h2>Cool KATsss</h2>
<p>Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet
doming id quod <a href="nil.htm">mazim placerat facer</a> possim assum.</p>
</blockquote>
```

View the completed file in your browser, and compare it to the example in Figure 20.6.

FIGURE 20.6.
Light and bright kid's page.



With the completed style sheet in hand, it's time to create the preceding fun kid's magazine in the linked style.

Take the embedded style out of the originating page, and place it into a clean file:

```
<style>
BODY {color: #FFFFFF; margin: 0.05in 0.75in}
H1 {font: 20pt "Kids"; color: #663366; text-align: right}
H2 {font: 18pt "Kids"; color: #663366; text-align: left}
```



```
P {font: 13pt "arial narrow"; color: #009933}
HR {color: #FF9900}
A {color: #FF9900; text-decoration: none}
</style>
```

Name this file `kids.css`. Place the following link in the `<HEAD>` of every page you want to have influenced by this style:

```
<link rel=stylesheet href="pj.css" type="text/css">
```

Then, save and test your files to be sure you've typed everything correctly.

Online Book Cover

This task shows you a very sophisticated look and feel gained through style. Pay special attention to another way of using class. In this case, I've defined attributes, not tags, and given them a name in the `.namegoeshere` style. This allows me to call that class within any HTML section I choose; therefore, the style that string contains will be used. Also note the use of graphics. Even with three graphics on the page, the total HTML and graphic combination results in only 11KB.

Begin with the style, paying attention to the defined classes `.HEAD` and `.SUB`:

```
<style>
BODY {font: 16pt "garamond"; font-style: "italic";color: #66CCCC;
➥ text-align: center; background: #000000}
.HEAD {font: 15pt "arial narrow"; font-weight: bold; text-align: center;
➥ font-style: none}
.SUB  {line-height: 20pt}
</style>
```

Next, add the table and contents, noting the use of class each step of the way:

```
<table border=0 width=500 cellpadding=5 cellspacing=0>
  <tr>
    <td width=200 class=head>
      d a r e   t o   d r e a m
    </td>
    <td width=200>
      <img src="images/wow1.gif" width=100 height=147>
    </td>
  </tr>
  <tr>
    <td width=200>
      <img src="images/wow2.gif" width=75 height=116>
    </td>
    <td class=sub width=200>
      . . . a journey into<br> lives <br> and lifetimes
    </td>
  </tr>
  <tr>
    <td width=200>
      a series of poems<br>
      ~ by ~ <br>
```

```
      women of war
    </td>
    <td width=200>
      <img src="images/wow3.gif" width=175 height=110>
    </td>
  </tr>
</table>
```

Save the file and view it in your browser. Compare the results with the example in Figure 20.7.

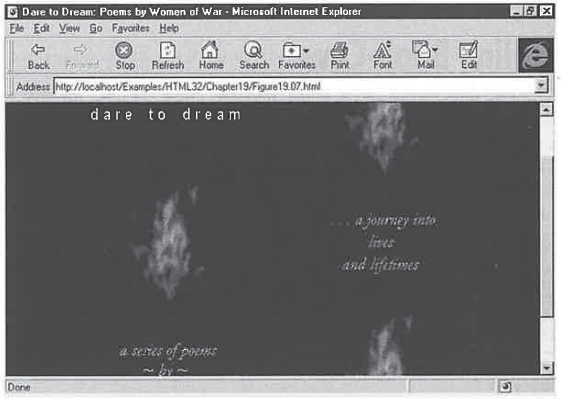


FIGURE 20.7. *An elegant, artistic book cover.*

Summary

By now, you should have a taste of what style sheets can do, how they work, and the many flexible ways of using them.

As always, you have to determine how to apply style sheets in your own work. For the purposes of this chapter and the previous chapter, you should have enough information and examples to get you started. Remember that although style sheets are gaining support, usage, and popularity, they still aren't predominant enough for you to depend on them exclusively. Either design the page so that it looks good regardless of the browser, or provide a mirror set of pages in non-style sheet versions.

If you are definitely considering working with style sheets more regularly, you should visit the World Wide Web Consortium at `www.w3.org` for very explicit discussions regarding style sheet development, implementation into browsers, and use in design.

CHAPTER 21

JavaScript Style Sheets and Other Alternatives to CSS

by Will Kelly

IN THIS CHAPTER

JavaScript style sheets are another option to control formatting in a Web document. Style sheets are a real boon to Web publishing because they offer more control over formatting than just HTML. JavaScript style sheets utilize JavaScript to specify element styles in an HTML document.

NOTE

Chapter 19, "Introducing Cascading Style Sheets," provides a strong introduction to the basics of style sheets. JavaScript style sheets are introduced here.

JavaScript style sheets are still coming into use on the Web. More sites are due to spring up as Netscape Communicator becomes more stable and becomes a shipping product. In the past, the release of a Netscape Navigator browser was accompanied by brand new Netscape Extensions to HTML. The release of Netscape Communicator also brought with it JavaScript style sheets.

When the Web took off as a communications medium, one of the biggest complaints from communicators was the lack of formatting options available in Hypertext Markup Language. HTML, though a powerful tool for multiplatform content, has never been known to be designer friendly. This is changing for the better, just like everything else on the Web.

Netscape Communications Corporation was the first commercial company to seize upon the lack of creative options in HTML with the introduction of its proprietary HTML extensions, called Netscape Extensions. These extensions gained popularity with designers who wanted more control over formatting HTML documents. Netscape continued to push the envelope for designers with new extensions with each release of Netscape Navigator. Though Netscape releases new HTML extensions with each release of its browser, it is also the brains behind JavaScript style sheets.

With the release of Netscape Communicator, Netscape debuted JavaScript style sheets, the next era in formatting control for HTML documents.

This chapter provides an introduction to JavaScript style sheets and their major elements.

What Are JavaScript Style Sheets?

JavaScript style sheets are style sheets that use JavaScript to set style properties. They aren't the same as cascading style sheets (CSS), though they serve the same role as CSS. You need to know JavaScript to use JavaScript style sheets to format your documents; you manipulate the style sheets through the JavaScript language. The JavaScript style sheet syntax offers control over the specific style for a particular HTML element. For example, suppose you are developing a corporate home page and want to format the page with the company's colors, which are dark blue, white, and black. You can set individual styles to cover the corporate colors. You can specify headings to display in blue. You could specify paragraph text to display in black.

You might already have been introduced to style sheets through word processing and desktop publishing. This crossover in publishing technology is understandable considering the migration of designers and other publications professionals to the Web. As a technical writer, I spent my fair share of time producing manuals with word processing style sheets. The power of style sheets offered the ability to combine multiple formatting elements in one style. I could create styles that could cover the font, font size, margins, and any other formatting element I needed to set. When JavaScript style sheets became a reality, I was ecstatic to learn that the power of style sheets was introduced to Web design and publishing.

NOTE

JavaScript style sheets are a particular application of JavaScript. If you need to learn more about JavaScript, refer to Chapter 28, "Integrating JavaScript and HTML."

JavaScript style sheets enable you to set classes that can cover multiple formatting requirements. The simple example I just mentioned talked only about colors, but you can carry the styles further. For the paragraph text, you can set a class for the headings that could cover the font, bolding, and centering of the headings. This class, which you could name something like `CORPHD`, can help you maintain a consistent organizational identity between online and print publications. It is also a labor-saver because it sets all the styles at once, rather than having to set each style individually. The `CORPHD` class takes care of the work of three tags:

- The `<FONT>` tag to set the font of the heading
- The `<B>` tag to set the heading as bold
- The `<CENTER>` tag to specify the centering of the heading

You implement JavaScript style sheets via in-line style sheets to control the format of existing HTML tags present in your Web project.

Netscape Communicator supports both JavaScript and cascading style sheets. Microsoft Internet Explorer 3.0 and 4.0 support cascading style sheets.

Older browsers like NCSA Mosaic and Cello don't support style sheets. Then what happens when a user running an older browser hits my site, which is equipped with JavaScript style sheets? The browser will be able to interpret only the HTML code present in the Web page.

New HTML Tags for Style Sheets

JavaScript style sheets don't replace HTML. They format HTML. But just as you need an HTML tag to support any element on a Web page, HTML tags support JavaScript style sheets. These tags enable the browser to interpret the style sheet and thus the document formatting. Three new HTML tags were introduced to support JavaScript style sheets: the `<STYLE>` tag, which indicates a document is formatted with a style sheet; the `<LINK>` tag, which specifies a

link to an external style sheet in a document; and the <SPAN> tag, which specifies the beginning and end of a styled piece of text.

The <STYLE> Element

The opening and closing <STYLE> tags indicate that a style sheet is being used to format the HTML document. Between the <STYLE> and </STYLE> tags, you can establish styles for use in an HTML document, including styles for elements, classes, IDs, and general styles.

The <STYLE> tag is just not particular to JavaScript style sheets; it is also used for cascading style sheets. When you use the <STYLE> tag to support a JavaScript style sheet, you need to specify the TYPE attribute as "text/javascript".

NOTE

The <STYLE> tag is always implemented within the HEAD element.

Listing 21.1 shows an example of how to implement the <STYLE> tag in an HTML document.

Listing 21.1. The <STYLE> tag in action.

```
<HTML>
<HEAD>
<STYLE TYPE= "text/javascript">
tags.BODY.marginRight=10
tags.BODY.marginLeft=10
CorpHD.align="center"
CorpBD.align="left"
</STYLE>
</HEAD>
```

The <LINK> Element

The <LINK> element links to an external style sheet to format the HTML document. Listing 21.2 shows an example of how to implement the <LINK> element.

Listing 21.2. The <LINK> tag in action.

```
<HTML>
<HEAD>
<TITLE>The <LINK> Tag in Action</TITLE>
<LINK REL=STYLESHEET TYPE= "text/JavaScript"
HREF= "http://samplesite.com/exstyles" TITLE= "example"
</HEAD>
```


NOTE

The <LINK> tag is always implemented within the HEAD element.

The <SPAN> Element

The <SPAN> element indicates the beginning and end of styled text. Text within the <SPAN> and </SPAN> tags is the text to which the style is applied. Listing 21.3 shows an example of what the <SPAN> tags can do in an HTML document.

Listing 21.3. The <SPAN> tag in action.

```
<P><SPAN style=color= 'red' fontWeight= 'bold' fontStyle= 'italic'>
Blivits Communication </SPAN>is a full service technical and Web
communications firm composed of communicators drawn from business and government.
We do documentation, Websites, and most any communications piece</P>
```

Style Sheets and Existing HTML Tags

The introduction of the JavaScript style sheet as a formatting tool, besides using newer HTML tags, has also introduced new attributes for existing HTML tags. These new attributes primarily focus on style issues. The new attributes follow:

- STYLE: The STYLE attribute is an important new attribute because it enables you to specify the style for a specific element.
- CLASS: The CLASS attribute enables you to define classes for styles.
- IDS: The IDS attribute enables you to specify exceptions to styles. Stylistic exceptions come into play when you want to designate a slight change to the default style for a class. An example would be italicizing a particular word or words when the rest of the text is just in a normal typeface.

NOTE

The IDS attribute is not the same as the IDS JavaScript object. The IDS HTML attribute specifies a unique style for the HTML element, whereas the IDS JavaScript object enables you to create a unique style identification.

JavaScript Style Sheet Properties and Values

This section gets into the nitty-gritty of JavaScript style sheets. You format via properties that actually specify the format. For instance, the font properties govern the formatting of fonts in a Web document, including font type, size, and appearance.

We've already introduced you to some of the basics behind style sheets in Chapter 19. Many of those same rules apply here, except that JavaScript is the programming language controlling the formatting.

Font Properties

One chief formatting complaint in days of yore, when you could only produce Times New Roman as a body font for HTML documents, was the lack of choice in font styles. This has improved with advancements in HTML and proprietary HTML extensions. Style sheets such as JavaScript style sheets also contribute to the improvements in font formatting. In fact, one of the most common applications of style sheets is formatting fonts.

The `fontSize` Property

The `fontSize` property controls the setting of font size and related characteristics. Four possible values apply to all elements:

- `absolute-size`: The `absolute-size` value contains the index to a table of font sizes computed by the rendering program for the style sheet. It includes the following possible values: `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, and `xx-large`.
- `relative-size`: The `relative-size` value is designated in reference to the table of font sizes and the parent element's font size. The relative-size value has the value of either `smaller` or `larger`.
- `length`: The `length` value is the absolute value of the line height.
- `percentage`: The `percentage` value is the percentage of the line height of the parent element.

The `fontFamily` Property

The `fontFamily` property designates the font family that is used for the style, such as Courier or Arial. The `fontFamily` value is the only possible value for the property. The property can apply to all elements.

Sometimes a font is not available to a user. Configuration management is impossible in a worldwide enterprise environment such as the World Wide Web. You need to be aware of five generic font families, including serif, sans-serif, cursive, monospace, and fantasy. I am not going to start a lesson on typography; I'll keep it simple. The five generic family names are guaranteed to indicate a font on every system. This guarantee is platform-independent, so it covers Windows, Macintosh, Unix, OS/2, and the other client operating systems that are plugged into the Internet.

The `fontStyle` Property

The `fontStyle` property controls the style of fonts in the HTML document. This property has five possible values: `normal`, `italic`, `italic small-caps`, `oblique`, `oblique small-caps`, and `small-caps`. The `small-caps` value can also encompass the `italic` and `oblique` values. On some systems, capital letters of a smaller font size will not be rendered correctly if you do not correctly set the resolution for the output medium.

Text, Classes, and ID Properties

JavaScript style sheets also enable you to set text properties, such as text decoration and line height.

The `lineHeight` Property

The `lineHeight` property sets the distance between the baselines of two adjacent lines. This property applies only to block-level elements. The `lineHeight` property has three possible values:

- `number`: The `number` value specifies the line height as computed by the font size of the current element multiplied by the numerical value. The numerical value is not the same as the percentage value because of how it inherits. Inheritance takes place when the numerical value is specified; then child elements will inherit the factor and not the resultant value.
- `length`: The `length` value is the line height's absolute value.
- `percentage`: The `percentage` value is the percentage of the parent element's line height.

The `textDecoration` Property

The `textDecoration` property describes the decorative elements applied to an element's text. This property has five possible values: `none`, `underline`, `overline`, `line-through`, and the ever-fabulous `blink`. If the element has no text, the `textDecoration` property has no effect, which would happen with elements such as IMG and Bold.

The `verticalAlign` Property

The `verticalAlign` property designates the vertical positioning of text. This property has nine possible values:

- `baseline`: The `baseline` value aligns the element's baseline with the parent element's baseline.
- `sub`: The `sub` value displays the element in subscript style.
- `super`: The `super` value displays the element in superscript style.
- `top`: The `top` value aligns the top of the element with the tallest element present on the line.

- `text-top`: The `text-top` value aligns the top of the element with the parent element's font top.
- `middle`: The `middle` value aligns the element's vertical mid-point with the baseline plus half the parent's x-height. The x-height is defined as the height of the character x.
- `bottom`: The `bottom` value aligns the element's bottom with the line's lowest element.
- `text-bottom`: The `text-bottom` value aligns the element's bottom with the bottom of the parent element's font.
- `percentage`: The `percentage` value refers to the element's line height.

The `textTransform` Property

The `textTransform` property indicates the text case of a font. The property has four possible values and applies to all elements:

- `capitalize`: The `capitalize` value displays the first character of each word in uppercase.
- `uppercase`: The `uppercase` value displays all letters of the element in uppercase type.
- `lowercase`: The `lowercase` value displays all the element's letters in lowercase.
- `none`: The `none` value neutralizes any and all inherited values.

The `textAlign` Property

The `textAlign` property specifies how text is aligned within the element. This property has four possible values: `left`, `right`, `center`, and `justify`.

The alignments that the values represent are relative to the width of the element. If the client system does not support the `justify` element, it will supply a replacement. In the case of the western languages that dominate so much of the World Wide Web, the substitution will be the `left` value.

The `textIndent` Property

The `textIndent` property designates the indent that appears before the first formatted line of text. This property has two possible values:

- `length`: The `length` value defines the length of the indent as a unit-based numerical value.
- `percentage`: The `percentage` value defines the length of the indent as a percentage of the width of the parent element.

NOTE

An indent is never inserted in the middle of an HTML element that is broken by another element, such as the `<br>` tag.

Block-Level Element Format Properties

The block-level elements, as defined by JavaScript style sheets, govern such elements as headings and paragraphs as boxes that can include borders, paddings, and margins.

The Margin Properties

The margin properties encompass the following:

- `marginLeft`
- `marginRight`
- `marginTop`
- `marginBottom`
- `margins()`

These properties specify the margin of an element. The margin is specified via individual values. The `margins()` value sets the margins for all four sides at once. These margins designate the minimal distance between adjoining element borders.

The Padding Element Properties

The padding element properties encompass the following:

- `paddingTop`
- `paddingRight`
- `paddingBottom`
- `paddingLeft`
- `paddings()`

The padding element properties specify how much space to insert between the border and the content of the document. You can set the values for padding by specifying values.

The Border Element Properties

The border element properties encompass the following:

- `borderTopWidth`
- `borderRightWidth`
- `borderBottomWidth`
- `borderLeftWidth`
- `borderWidths()`

The border properties specify the border width around an object in pixels, em units, or points. By specifying values for the elements, you can set the border width.

The borderStyle Property

The borderStyle property specifies the style of a border around a block-level element. This property has three possible values: none, solid, and 3D.

The borderColor Property

The borderColor property specifies the color of the element's border. You can specify the color to be a named color or a six-digit hexadecimal color value.

The width Property

The width property designates the width of an element. This property is typically applied to text elements, but it is best used when applied to in-line images and other media insertions. The width property has three possible values: length, percentage, and auto.

The height Property

The height property specifies the height of the element. This property works best when you apply it to in-line graphic images and similar media insertions. You can also apply this property to textual elements, however. The height property has only two values: length and auto.

The align Property

The align property specifies the element's alignment. This element can include text or an in-line image. The align property has three possible values: left, right, and none.

NOTE

The align property is somewhat related to cascading style sheets: it corresponds to the CSS float property. The term float is a reserved term in JavaScript and can't be used for a property name.

The align property is most often implemented with in-line images and can make an element float to either the right or left. It also controls how other content wraps around the property.

The clear Property

The clear property specifies whether an element allows floating elements on all of its sides. It has four possible values: none, left, right, and both.

Color and Background Properties

You can set color and background properties for block-level elements just as you can set them for an entire HTML document.

The color Property

The color property specifies the element's text color. The color specified is the foreground color, rather than the background color. Text is always at the foreground of a document. This property's only possible value is the color value.

The backgroundImage Property

The backgroundImage property specifies the background image of an element. The url value is its only possible value.

Classification Properties

The classification properties classify properties into appropriate categories and don't set element appearance.

The display Property

The display property specifies whether an element is one of the following possible values:

- in-line element is like the <B> HTML tag.
- block-level is like the <P> HTML tag.
- block-level list item is like the <LI> HTML tag.

The initial value for the property is drawn from the HTML specification. If the none value is implemented for this property, the display of the element is turned off.

The listStyleType Property

The listStyleType property specifies the formatting of list items in HTML documents. List items would be classed as items with a display value of list-item. This property has nine possible values: disc, circle, square, decimal, lower-roman, upper-roman, lower-alpha, upper-alpha, and none.

You can specify this property on any element. It will inherit down the tree normally.

NOTE

The list is displayed only on elements with list-item as its display value.

The whiteSpace Property

The whiteSpace property specifies how white space inside the element should be handled. This property has two possible values: normal and pre. The normal value specifies that white space is collapsed. The pre value works the same way as the <PRE> HTML tag.

Precedence Order

The precedence order is determined for tags, classes, and IDs via an algorithm. The algorithm works as follows:

- It finds all declarations applying to the specific element/property.
- It sorts all declarations by their explicit weight.
- It sorts by the origin. Via this sorting, the style sheets developed by the given Web site's author override the reader's style sheet. This override works as a default.
- It sorts by the selector's specificity, with more specific selectors overriding the general selectors.

Other Alternatives to CSS

JavaScript style sheets aren't the only game in town as far as alternatives to cascading style sheets. The World Wide Web Consortium (`http://www.w3.org`) is currently developing the next version of Hypertext Markup Language with input from the online community. The new version includes a specification for styles. Netscape Communicator introduced Dynamic Fonts with Preview Release 3. This powerful feature, though not a style sheet per se, enables developers to have more control over the fonts in their HTML documents than has ever been offered before.

Cougar

The new version of HyperText Markup Language, code named *Cougar*, introduces many new features to HTML including its own version of styles and style sheets. The standards are all in the draft stage. With this style specification, you no longer have to extend HTML code when you need new presentation styles to get the job done. Now, you can include style rules as a group in the document header, as part of individual HTML elements in the document, or as part of associated style sheets.

This new style sheet draft has influenced both CSS and JavaScript style sheets by providing the specs for the `<LINK>`, `<STYLE>`, and `<SPAN>` elements. The basic concepts of style sheets as set out by Cougar are as follows:

- Style information placement: One of the charms of style sheets is the developer's ability to reuse them. Style sheets are placed in separate files and are not embedded into an HTML document like traditional Web formatting.
- Style sheet language independence: The Cougar specification for style sheets doesn't tie HTML into a particular style sheet language. Remember, JavaScript style sheets use the JavaScript language to control formatting. With this specification, you have more options to use other languages to specify formatting.
- Cascading style sheets: This feature of the specification enables you to use style information from multiple sources to format an HTML document.

- Media dependencies: This feature enables you to define styles for media types in an HTML document.
- Alternative styles: The Cougar style specification provides readers with alternative styles for viewing style sheet–formatted Web content.

Point your browser to `http://www.w3.org/pub/WWW/TR/WD-style` to learn more about this emerging standard.

Dynamic Fonts

I have the tendency to be a font monger. Not that I use them all the time, but there is a certain sense of security that they are available and at my beck and call. When I was poring over the specs for Netscape Communicator (`http://home.netscape.com/inf/comprod/products/communicator/index.html`), I was happy to learn about Communicator's support of Dynamic fonts. Dynamic fonts enable you to include fonts dynamically in your HTML documents. This dynamic link is accomplished via an author-defined link to a URL containing the font files.

To learn more about Dynamic fonts, and what they have to offer the Web developer, consult the following address:

```
http://developer.netscape.com/library/documentation/communicator/jsss/index.htm
```

Future of JavaScript Style Sheets

JavaScript style sheets are closely tied to Netscape Communicator and other Netscape Web publishing technologies. Netscape will drive JavaScript style sheet development to compete against Microsoft's support of cascading style sheets. Marketing is certainly driving the technology, but it is also good to know that Netscape Communicator supports CSS, making Netscape prepared to support either way.

Summary

This chapter covered the basics of JavaScript style sheets and offered an introduction to some of the other options to cascading style sheets that are becoming available for Web development.

JavaScript style sheets offer Web developers control over formatting Web pages that they could never have had with plain HTML. The introduction of JavaScript style sheets will also help the Web mature even more as a communications and creative medium because more options are now available to the artists and Web developers developing content for the Web.

CHAPTER 22

Dynamic HTML

by Bruce Campbell

IN THIS CHAPTER

Introduction to Dynamic HTML

HTML, as defined in the 3.2 specification, standardizes the creation process of static documents that are accessed from the Web and consumed by a Web audience. Scripting languages, programming languages, user controls, and plug-in applications are incorporated into Web pages by Web page designers to add animation, database access, sound, video, and interactive applications. Dynamic HTML is an enhancement to HTML that provides a group of technologies designed to create and display more interactive Web pages within the `.htm` or `.html` file itself, avoiding the complexity of requiring additional plug-in applications, add-on controls, and multiple Web server requests. Microsoft is leading the way in providing Dynamic HTML enhancements by including the Dynamic HTML technologies in the Internet Explorer 4.0 Web browser. Microsoft continues to work closely with the World Wide Web Consortium, or W3C (`http://www.w3.org/pub/WWW/`), the governing body of the HTML standard, to provide an open standard that other Web browser developers can incorporate into their applications. Netscape (`http://www.netscape.com`) is incorporating many features of Dynamic HTML into its upcoming Communicator product. In fact, Netscape has drafted additional functionality for Dynamic HTML that shows innovative and useful characteristics. As the W3C incorporates the innovations from Microsoft and Netscape, other groups and individuals are sure to further refine Dynamic HTML as well.

Included in Dynamic HTML is a new object model designed to make HTML objects more accessible to scripting functions that take advantage of the existing HTML `<SCRIPT></SCRIPT>` tag pair. Using the new object model to create Web pages and a new Web browser to surf Dynamic HTML-based Web content, developers can get a head start on learning the promises Dynamic HTML holds for the Web.

Benefits of the New Object Model

The new object model provides the following benefits examined in detail in the four HTML examples provided in this chapter:

- Access to all page elements for dynamically changing page appearance
- Instant client page update without additional server services
- An exciting event model tracking mouse usage and page state
- HTML text ranges that can be changed on the fly to dramatically change content

Any Web page element can be identified as an object using HTML's `ID` attribute. For example, a paragraph of text can be instantiated through the `ID` attribute of the following HTML line:

```
<P ID=ParagraphX STYLE="font=weight: normal">
```

Using the `ID` attribute value, a function within a `<SCRIPT></SCRIPT>` tag pair can then instantly change the page as a result of an event, such as the following:

```
ParagraphX.style.fontstyle = "Italic"
```


Any `ID` attribute value can be instantiated to receive messages from other script-based functions on the HTML Web page.

Available Events in Dynamic HTML

Client-based interactive Web pages are made possible by the events incorporated into Dynamic HTML. Table 22.1 presents the available events provided by the Internet Explorer 4.0 implementation at press time. Follow the Reference link on Microsoft's Dynamic HTML Web page for new developments.

NOTE

At press time, the URL for Dynamic HTML information at Microsoft was `http://www.microsoft.com/workshop/prog/ie4`. Any changes to the Dynamic HTML specification since this chapter was written should be documented at Microsoft's Web site.

Table 22.1. Dynamic HTML events.

onabort	onfocus	onmouseup
onafterupdate	onhelp	onreadystatechange
onbeforeupdate	onkeydown	onreset
onblur	onkeypress	onrowenter
onbounce	onkeyup	onrowexit
onchange	onload	onscroll
onclick	onmousedown	onselect
ondblclick	onmousemove	onstart
onerror	onmouseout	onsubmit
onfinish	onmouseover	onunload

The user's mouse and keyboard actions, as well as the current state of the active Web page, generate events. You can embed event handling directly into an HTML tag, as in the following code:

```
<P ID=ParagraphX STYLE="font=weight: normal"
onmouseover="Italicize();"
onmouseout="Normalize();"
>
```

Passing the mouse over and off the paragraph text that follows this `<P>` tag in an HTML Web page invokes the `Italicize()` and `Normalize()` functions. Both functions are written in the same `.htm` file as the `<P>` tag using a scripting language within the `<SCRIPT></SCRIPT>` tag pair.

With the new object and event model in hand, developers can provide the following features in a single HTML Web page:

- Dynamically changing styles
- Dynamically changing content
- Dynamic positioning of graphics
- Dynamic data-awareness

I will review each feature in this chapter by way of an example created for the Internet Explorer 4.0 pre-release version provided by Microsoft for download through its Web site. I have coordinated the examples to reflect four pages of the same Web site.

DYNAMIC HTML: MICROSOFT, NETSCAPE, AND THE W3C

To be honest, I am using Microsoft's IE4 for this chapter's examples because it appears to be a bit more mature than Netscape's implementation at the time of this book's publication. The examples definitely give you a flavor for the kind of things that will be available from Dynamic HTML even if the name is changed to reflect HTML 4 or some other buzzword.

Both Netscape and Microsoft are admitting allegiance to the W3C as the final arbiter of any official specifications that come out as a result of these Dynamic HTML features. I definitely suggest that you check out Netscape's implementation if you really like authoring for the Navigator browser—especially if you are already using the LiveConnect API architecture. You should check out Microsoft's implementation if you really like authoring for Internet Explorer and you've invested substantial time in learning ActiveX and Visual Basic. But regardless of your situation, you should always read the technical papers at the W3C Web site because they define the reasons behind the decisions to make changes for the mutual progression of the Web. Check out the current Cougar project for some very important papers on HTML's development. For now, if your Web site exists for general consumption, be sure to test your Dynamic HTML files with both browsers.

Because the examples from this chapter lean so heavily toward Microsoft's vision of Dynamic HTML, I include the following list so that you can ponder Netscape's vision. Within Netscape's implementation are the following sample features:

- Netscape defines a <LAYER> tag that you can use to set up independent regions of a Web page and position them independently.
- Netscape defines a standard, vectored font specification that can be downloaded with a page to create a font that is not resident on the reader's machine.
- Netscape defines another scripting language, called JavaScript, that can be used to handle dynamic events.
- Netscape is considering changes to its plug-in architecture to take advantage of new Dynamic HTML features.

Both Microsoft and Netscape are committed to developing Dynamic HTML features that behave well in older browsers that can't use all of the Dynamic HTML functionality. Significant work must be done in order to keep that promise, but I suspect features that work in one browser should not detrimentally affect the other. If they do, the public at large should give that browser developer a hard time.

The examples are interactive Web pages designed using Dynamic HTML for the fictitious Web Baseball League (WBL), which consists of eight baseball teams playing a 112-game season of virtual baseball over the Web. The site has been designed for fans of the league to keep abreast of league standings and interact with the pages from their computers.

The following list summarizes the features of the sample WBL pages by each listing number that you can reference in this chapter:

- Point and click team names to see expanding and collapsing current information sections on each team. (See Listing 22.1.)
- Drag and drop team logos into an interactive form to participate in a contest of predicting future team standings. (See Listing 22.2.)
- Click buttons to enter each team's electronic locker room with data-awareness enabled for local data manipulation. (See Listing 22.3.)
- Use dynamic table interaction to sort current league standings by team name, wins, losses, batting averages, and earned run averages. (See Listing 22.4.)

Now let's investigate the code behind the examples. Consider that all code is integrated with HTML tags in a single `.htm` file. No additional plug-ins, helper applications, or virtual machines are necessary to create the dynamic effects of each page. The Web server is contacted only once to present the information to the Web browser client. With Dynamic HTML, Internet packet traffic is reduced significantly and a considerable burden is taken off the Web server.

Creating Dynamic Text

Dynamic HTML sets up event handling in the Web page itself. You can handle events that dynamically change the text on the Web page as the user moves, clicks, or double-clicks the mouse, or presses a key on the keyboard. The effect can be dramatic because you can change the HTML tags that format a paragraph as well as the text itself on the fly.

The example in Listing 22.1 builds a Web page that league fans can visit daily to get up-to-date information on each team. Perhaps you could have similar pages for injury reports, upcoming weekly schedules, or insider's gossip.

The page is set up to track the user's mouse clicks. As the user clicks a team name, the team name expands to present detailed information for that team. The user can click again to collapse the detail. Such page behavior presents an uncluttered Web page, avoids unwanted

detail, and reduces the need to scroll to find information at the site. Dynamic text is a powerful feature in combating information overload for the casual Web surfer, but also can be used to anticipate the needs of the audience. Figure 22.1 shows the team page before a user has clicked a team. Figure 22.2 shows the page after a user has clicked the Spiders team name. The page returns to appear as in Figure 22.1 when the user clicks anywhere on the Spiders detail text.

Listing 22.1. Example of dynamic text.

```
<HTML>
<BODY TOPMARGIN=0 LEFTMARGIN=40 BGCOLOR= "#FFFFFF" LINK= "#000066"
 VLINK= "#666666" TEXT= "#000000">
<HEAD>
<FONT FACE="verdana,arial,helvetica" SIZE=3>
<TITLE>The WBL Daily Update</TITLE>
</HEAD>
<BODY>
<STYLE>
.redText {color:Red}
.blueText {color:Blue}
</STYLE>
<H3>The Web Baseball League Daily</H3>
<IMG SRC="Bears.jpg" ALIGN=LEFT HEIGHT=32><P CLASS="blueText">
 Bears
<IMG SRC="Tigers.jpg" ALIGN=LEFT HEIGHT=32><P CLASS="blueText">
Tigers
<IMG SRC="Robins.jpg" ALIGN=LEFT HEIGHT=32><P CLASS="blueText">
Robins
<IMG SRC="Lions.jpg" ALIGN=LEFT HEIGHT=32><P CLASS="blueText">
Lions
<IMG SRC="Rhinos.jpg" ALIGN=LEFT HEIGHT=32><P CLASS="blueText">
Rhinos
<IMG SRC="Frogs.jpg" ALIGN=LEFT HEIGHT=32><P CLASS="blueText">
Frogs
<IMG SRC="Sharks.jpg" ALIGN=LEFT HEIGHT=32><P CLASS="blueText">
Sharks
<IMG SRC="Spiders.jpg" ALIGN=LEFT HEIGHT=32>
<P id=spider CLASS="blueText" onclick="expandP();">
Spiders</P>

<SCRIPT LANGUAGE=VBScript>
function expandP()
   dim r
   set r = document.rangeFromElement(spider)
   if (spider.className="blueText") then
      r.pasteHTML("<B><I>Spiders</I></B><BR>Wins: 4<BR>Losses: 14<BR>
        ➥Batting Average: .212<BR>Earned Run Average: 3.86<BR>")
      spider.className="redText"
   else
      r.pasteHTML("Spiders")
      spider.className="blueText"
   end if
end function
</SCRIPT>
</FONT>
</BODY>
</HTML>
```


FIGURE 22.1.
Before clicking `Spiders`.

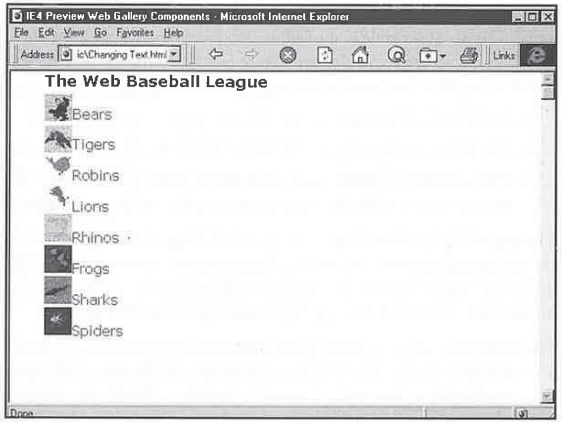
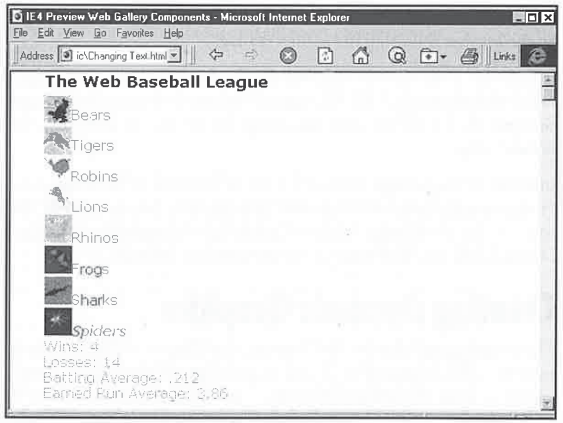


FIGURE 22.2.
After clicking `Spiders`.



The `<STYLE></STYLE>` tag pair defines two styles to be used in the page according to the specification for cascading style sheets presented in Chapter 19, "Introducing Cascading Style Sheets." The style `redText` colors text red. The style `blueText` colors text blue.

A paragraph is created for each team, including a thumbnail of the team logo and the team name. The `blueText` style is applied to each paragraph using the `CLASS` attribute of the `<P>` tag. In this example, the dynamic text feature is enabled for the Spiders team paragraph only. To enable dynamic text, the following steps are executed:

1. The `ID=spider` attribute of the `<P>` tag instantiates the paragraph as a dynamic object.
2. The `onclick="expandP();"` attribute of the `<P>` tag establishes the click event.
3. The `expandP()` function is created and inserted in a `<SCRIPT></SCRIPT>` pair tag using an appropriate scripting language such as JavaScript or, in this case, VBScript.

The `expandP()` function obtains a handle to a range of HTML text in the Web page by calling the `rangeFromElement()` method of the `document` and passing it an object that specifies a range in the Web page. In this case, the `spider` object defines a paragraph of text. Every Web page is instantiated as a `document` object upon loading in the Web browser.

The `expandP()` function interrogates the `CLASS` attribute value of the `spider` paragraph. If the `bluetext` style is active for the `spider` paragraph, the function calls the `pasteHTML()` method of the range handle it obtained and stored in variable `r`. The singular argument of the `pasteHTML()` method is a string of valid HTML tags and text. The detailed HTML tags and text string are pasted into the `spider` paragraph and dynamically presented on the Web page by the Web browser. The HTML syntax replaces the HTML tags and text that had been initially loaded by the browser following the `spider <P>` tag.

In contrast, if the `redtext` style is active for the `spider` paragraph, the `pasteHTML()` method is called with the original HTML tags and text to reset the `spider` paragraph back to its original contents. As the HTML contents change for the `spider` paragraph, the style toggles between red and blue.

Inherent in this example are two features of Dynamic HTML that you can use independently. First, you can change styles dynamically. Second, you can change the HTML tags and text contents for a text range (such as a paragraph or heading) on the fly, based on the available Dynamic HTML Web page events presented in Table 22.1.

Creating Dynamic Graphics

The cascading style sheet model introduced in Chapter 19, includes details for defining a `ZINDEX` value for a Web page graphic. Z-indexing enables a Web author to identify which graphic should appear in front of another if they share the same (x,y) coordinates. The object with a lower `ZINDEX` value is presented in front of objects with higher ones. This example builds on other dynamic graphic placement features added by Dynamic HTML.

The HTML Web page detailed in Listing 22.2 contains eight `.jpg` graphic files—one for each team logo in the Web Baseball League. The logos are presented on the page shown in Figure 22.3 with the expectation that a user will drag and drop the logos on the page to place them in the order he believes the teams will finish after the upcoming season. Perhaps the contest could be held weekly or monthly instead.

Listing 22.2. Example of drag-and-drop graphics.

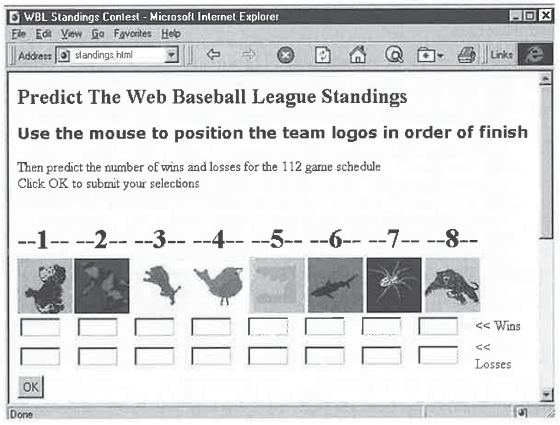
```
<HTML>
<HEAD><TITLE>WBL Standings Contest</TITLE></HEAD>
<BODY BGCOLOR="#FFFFFF">
<H2>Predict The Web Baseball League Standings</H2>
<FONT FACE="Verdana, Arial, Helvetica" SIZE="4">
<B>Use the mouse to position the team logos
 in order of finish</B><BR><BR></FONT>
Then predict the number of wins and losses for the 112 game schedule<BR>
Click OK to submit your selections<BR>

<SCRIPT LANGUAGE="VBScript">
function document_onmousemove(button, shift, x,y)
dim newleft, newtop, srcElement
  if (button = 1) then
    set srcElement = window.event.srcElement
    ' if mouse is dragging and IMG, move it.
    if srcElement.tagname="IMG" then
        ' move team logo
        newleft=x-document.all.OuterDiv.docLeft-(srcElement.docWidth/2)
        if newleft<0 then newleft=0
        srcElement.style.pixelLeft= newleft
        newtop=y-document.all.OuterDiv.docTop-(srcElement.docHeight/2)
        if newtop<0 then newtop=0
        srcElement.style.pixelTop= newtop
        window.event.returnValue = false
        window.event.cancelBubble = true
    end if
  end if
end function
</SCRIPT>

<DIV ID=OuterDiv style="position:relative;width:100%;height:600px">
    <IMG ID="bear" STYLE="container:positioned;position:absolute;TOP:58pt;
      LEFT:0px;WIDTH:64px;HEIGHT:64px;ZINDEX:-1;" SRC="Bears.jpg">
    <IMG ID="frog" STYLE="container:positioned;position:absolute;TOP:58pt;
      LEFT:66px;WIDTH:64px;HEIGHT:64px;ZINDEX:-2;" SRC="Frogs.jpg">
    <IMG ID="lion" STYLE="container:positioned;position:absolute;TOP:58pt;
      LEFT:134px;WIDTH:64px;HEIGHT:64px;ZINDEX:-3;" SRC="Lions.jpg">
    <IMG ID="robin" STYLE="container:positioned;position:absolute;TOP:58pt;
      LEFT:202px;WIDTH:64px;HEIGHT:64px;ZINDEX:-4;" SRC="Robins.jpg">
    <IMG ID="rhino" STYLE="container:positioned;position:absolute;TOP:58pt;
      LEFT:270px;WIDTH:64px;HEIGHT:64px;ZINDEX:-5;" SRC="Rhinos.jpg">
    <IMG ID="shark" STYLE="container:positioned;position:absolute;TOP:58pt;
      LEFT:338px;WIDTH:64px;HEIGHT:64px;ZINDEX:-6;" SRC="Sharks.jpg">
    <IMG ID="spider" STYLE="container:positioned;position:absolute;TOP:58pt;
      LEFT:406px;WIDTH:64px;HEIGHT:64px;ZINDEX:-7;" SRC="Spiders.jpg">
    <IMG ID="tiger" STYLE="container:positioned;position:absolute;TOP:58pt;
      LEFT:474px;WIDTH:64px;HEIGHT:64px;ZINDEX:-8;" SRC="Tigers.jpg">

<H1> --1-- --2-- --3-- --4-- --5-- --6-- --7-- --8--</H1>
<PRE>
```

continues

Listing 22.2. continued

```
</PRE>
<TABLE>
<TR>
<TD WIDTH=64><INPUT TYPE=text NAME="wins1" SIZE=5 VALUE="" MAXLENGTH=3></TD>
<TD WIDTH=64><INPUT TYPE=text NAME="wins2" SIZE=5 VALUE="" MAXLENGTH=3></TD>
<TD WIDTH=64><INPUT TYPE=text NAME="wins3" SIZE=5 VALUE="" MAXLENGTH=3></TD>
<TD WIDTH=64><INPUT TYPE=text NAME="wins4" SIZE=5 VALUE="" MAXLENGTH=3></TD>
<TD WIDTH=64><INPUT TYPE=text NAME="wins5" SIZE=5 VALUE="" MAXLENGTH=3></TD>
<TD WIDTH=64><INPUT TYPE=text NAME="wins6" SIZE=5 VALUE="" MAXLENGTH=3></TD>
<TD WIDTH=64><INPUT TYPE=text NAME="wins7" SIZE=5 VALUE="" MAXLENGTH=3></TD>
<TD WIDTH=64><INPUT TYPE=text NAME="wins8" SIZE=5 VALUE="" MAXLENGTH=3></TD>
<TD WIDTH=64><< Wins</TD>
</TR>
<TR>
<TD WIDTH=64><INPUT TYPE=text NAME="loss1" SIZE=5 VALUE="" MAXLENGTH=3></TD>
<TD WIDTH=64><INPUT TYPE=text NAME="loss2" SIZE=5 VALUE="" MAXLENGTH=3></TD>
<TD WIDTH=64><INPUT TYPE=text NAME="loss3" SIZE=5 VALUE="" MAXLENGTH=3></TD>
<TD WIDTH=64><INPUT TYPE=text NAME="loss4" SIZE=5 VALUE="" MAXLENGTH=3></TD>
<TD WIDTH=64><INPUT TYPE=text NAME="loss5" SIZE=5 VALUE="" MAXLENGTH=3></TD>
<TD WIDTH=64><INPUT TYPE=text NAME="loss6" SIZE=5 VALUE="" MAXLENGTH=3></TD>
<TD WIDTH=64><INPUT TYPE=text NAME="loss7" SIZE=5 VALUE="" MAXLENGTH=3></TD>
<TD WIDTH=64><INPUT TYPE=text NAME="loss8" SIZE=5 VALUE="" MAXLENGTH=3></TD>
<TD WIDTH=64><< Losses</TD>
</TR>
</TABLE>
<INPUT TYPE=submit VALUE="OK">
</FONT>
</BODY>
</HTML>
```

FIGURE 22.3.
WBL standings.



The HTML Web page in Listing 22.2 includes a script function that sets up an `onmousemove` event for the whole page. The Web page is instantiated as a `document` object when the Web browser loads the page. The `document_onmousemove()` function contains four arguments: the mouse button the user is currently holding down, the state of the Shift key on the keyboard, the x location of the mouse pointer, and the y location of the mouse pointer. This function is available for any Web page taking advantage of Dynamic HTML.

If the user is dragging an object by the left mouse button (button 1), the function associates that object with an `srcElement` variable declared in the function body. Next, the `srcElement` is interrogated to determine whether the object being dragged was created as the result of an `<IMG>` tag. If so, it is a graphical object and the script continues. If not, the function exits without interest in the mouse movement.

If the function determines that the user is dragging a bitmap with the left mouse button, the bitmap location is changed dynamically on the screen to reflect the current x and y values of the mouse pointer.

I have declared two variables in the function body—`newleft` and `newtop`—that I use to determine where the bitmap should be placed. The movement of the bitmaps is constrained by a region created by the Web author in an `OuterDiv` object identified with the `ID=OutDiv` attribute. The region is the full width of the Web page wide (`100%`) and is 600 pixels (`px`) high at the relative point in the page where it is encountered when parsing the `.htm` file.

The line `window.event.cancelBubble = true` tells the browser window to refrain from bubbling up the event in the case that the document cannot handle the event. The concept of bubbling has to do with the automatic passing of events up to a parent class of an object when an object is not equipped to handle that event. Bubbling helps reduce the amount of function writing that is necessary in cases where some objects on a page can handle an event by themselves. Objects that can't handle the event can use the same function and the event is automatically passed up the class hierarchy until it can be handled.

The rest of the HTML Web page after the `</SCRIPT>` tag is standard cascading style sheet and HTML syntax. I create and position the team logo bitmaps initially using the `STYLE` attribute of the `<IMG>` tags, I create a heading to mark the positions of the bitmaps from first to eighth, and I add a table of input text areas to accept the guesses of the user. The user submits his or her choices by clicking the OK button.

Substantial interactivity is added to Web pages by allowing for the dragging and dropping of component objects. Such interactivity is critical if the Web is to be able to compete with successful CD-ROM titles.

Creating Dynamic Data-Aware Pages

Data-awareness is a powerful enhancement that Dynamic HTML provides. With Dynamic HTML, you can access a data source anywhere on the Web with a Web browser, deliver it

locally to the requesting client, and then interactively present it to an audience without any additional network traffic.

To make a data-aware Web page, use the data source control specified within the <OBJECT> </OBJECT> tag pair. The Web browser can provide the data source control as a browser class or user control. Dynamic HTML coordinates the binding of the HTML page elements to the data source. Typical data sources that Dynamic HTML handles include comma-delimited files, SQL query dynasets, and other open database connectivity sources.

Data-aware Web pages using Dynamic HTML have the following features:

- Automatically generating table rows from data records
- Dynamically expanding tables
- Interactively filtering and sorting table contents
- Binding HTML elements to a specific record
- Data-binding form fields

The example in Listing 22.3 covers binding of HTML elements to a specific record and the example in Listing 22.4 reviews automatic generation of table rows and interactive sorting.

Binding HTML Elements to a Specific Data Source Record

Dynamic HTML provides a method of downloading a substantial data source from the Web, maintaining the data in computer memory, and presenting the data in smaller, comprehensible pieces. Such a feature has two main benefits: one-time Web server access and manageability to avoid user information overload. Traditionally, users have accessed multiple pages from a Web site to fulfill their information consumption needs. Web authors have provided these smaller pages to keep the presentation clean and to limit the need for scrolling the text (as well as to present well on WebTV). With Dynamic HTML, the author can still present the information in smaller chunks, but the Web server can deliver it all at once to avoid multiple request servicing.

As an example, Listing 22.3 creates a Web page that enables a new visitor to the Web Baseball League Web site to learn about the teams in the league one team at a time. Instead of requesting eight different pages from the Web server over time, the user is provided buttons to access the rows of a data source one row at a time. The whole data source is delivered to the user as a result of the client making a single server request. Figure 22.4 shows the Web page for the Frogs team row of the data source, which, in this case, is a simple comma-delimited text file named `wblteam.txt`. The `wblteam.txt` file is provided in Listing 22.5 later in this chapter.

Listing 22.3. An example of dynamic record presentation.

```
<HTML>
<HEAD>
<BODY TOPMARGIN=0 LEFTMARGIN=40 BGCOLOR="#FFFFFF" LINK="#000066"
VLINK="#666666" TEXT="#000000">
```



```
<FONT FACE="verdana,arial,helvetica" SIZE=2>
<TITLE>WBL Team Records</TITLE>
</HEAD>
<H2>Web Baseball League Teams</H2>
<BODY BGCOLOR="#FFFFFF">
<HR>
<P>Click the buttons below to investigate the teams of the Web Baseball League.
<P>
<OBJECT ID="teamlist"
        CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
        BORDER="0" WIDTH="0" HEIGHT="0">
  <PARAM NAME="DataURL" VALUE="wblteam.txt">
  <PARAM NAME="UseHeader" VALUE="True">
</OBJECT>

<TABLE>
  <TR>
    <TD ALIGN=RIGHT><INPUT TYPE=BUTTON ID=backward VALUE="   <   "></TD>
    <TD ALIGN=LEFT><INPUT TYPE=BUTTON ID=forward VALUE="   >   "></TD>
  </TR>
</TABLE>
<P>
<IMG ID=Picture SRC="Tigers.jpg" BORDER=4 HEIGHT=128 WIDTH=128 ALIGN=LEFT>

<SCRIPT LANGUAGE=VBScript>
function document_onclick()
    Picture.src = team.value + ".jpg"
end function
</SCRIPT>

<TABLE ALIGN=CENTER CELLSPACING=0 CELLPADDING=0>
<TR>
<TD ALIGN=RIGHT VALIGN=TOP><LABEL FOR=team>Team Name: </LABEL></TD>
<TD ALIGN=LEFT VALIGN=TOP WIDTH="10"></TD>
<TD ALIGN=LEFT VALIGN=TOP>
<INPUT ID=team TYPE=text DATASRC=#teamlist DATAFLD="Team"></TD>
</TR>
<TR>
<TD ALIGN=RIGHT VALIGN=TOP><LABEL FOR=wins>Current Number of Wins: </LABEL></TD>
<TD ALIGN=LEFT VALIGN=TOP WIDTH="10"></TD>
<TD ALIGN=LEFT VALIGN=TOP>
<INPUT ID=wins TYPE=text DATASRC=#teamlist DATAFLD="Wins"></TD>
</TR>

<TR>
<TD ALIGN=RIGHT VALIGN=TOP>
<LABEL FOR=losses>Current Number of Losses: </LABEL></TD>
<TD ALIGN=LEFT VALIGN=TOP WIDTH="10"></TD>
<TD ALIGN=LEFT VALIGN=TOP>
<INPUT ID=losses TYPE=text DATASRC=#teamlist DATAFLD="Losses"></TD>
</TR>

<TR>
<TD ALIGN=RIGHT VALIGN=TOP>
<LABEL FOR=BA>Current Team Batting Average: </LABEL></TD>
<TD ALIGN=LEFT VALIGN=TOP WIDTH="10"></TD>
```

continues

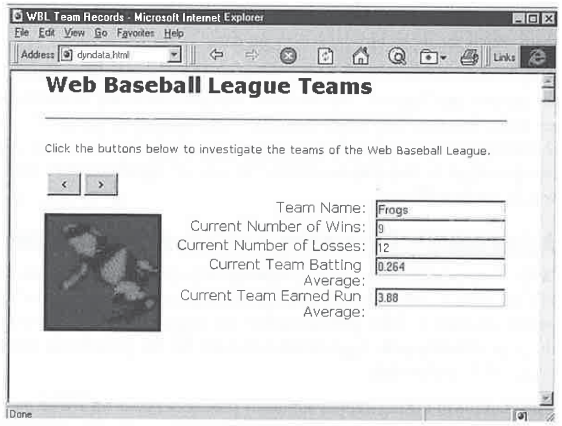
Listing 22.3. continued

```
<TD ALIGN=LEFT VALIGN=TOP>
<INPUT ID=BA TYPE=text DATASRC=#teamlist DATAFLD="BA"></TD>
</TR>

<TR>
<TD ALIGN=RIGHT VALIGN=TOP>
<LABEL FOR=ERA>Current Team Earned Run Average: </LABEL></TD>
<TD ALIGN=LEFT VALIGN=TOP WIDTH="10"></TD>
<TD ALIGN=LEFT VALIGN=TOP>
<INPUT ID=ERA TYPE=text DATASRC=#teamlist DATAFLD="ERA"></TD>
</TR>
</TABLE>

<SCRIPT LANGUAGE=VBSCRIPT>
function backward_onclick()
  if teamlist.recordset.AbsolutePosition > 1 then
    teamlist.recordset.MovePrevious
  else
    msgbox "At First Team"
  end if
end function

function forward_onclick()
  if teamlist.recordset.AbsolutePosition <> teamlist.recordset.RecordCount then
    teamlist.recordset.MoveNext
  else
    msgbox "At Last Team"
  end if
end function
</SCRIPT>
</FONT>
</BODY>
</HTML>
```

The data control object in this example is an ActiveX control that a user can download once from the Internet to extend the capabilities of his or her Web browser. I use the same control in Listing 22.4. The `<OBJECT></OBJECT>` tag pair includes the data control object in the HTML page. The `<OBJECT></OBJECT>` pair is the HTML specification's attempt to standardize a tag that encompasses browser-specific tags of the past, such as the `<EMBED></EMBED>` tag pair that Netscape uses to enable the plug-in architecture in HTML Web pages. Here are the ActiveX control identification lines from Listing 22.3:

```
<OBJECT ID="teamlist"
        CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
        BORDER="0" WIDTH="0" HEIGHT="0">
  <PARAM NAME="DataURL" VALUE="wblteam.txt">
  <PARAM NAME="UseHeader" VALUE="True">
</OBJECT>
```


FIGURE 22.4.
Dynamic record presentation.



The `ID` attribute value instantiates `teamlist` as the data source object. The `CLASS_ID` attribute value of the `<OBJECT>` tag is a sophisticated code with embedded security used to select the appropriate ActiveX control for data set management. The control requires two parameters identified in `<PARAM>` tags with `NAME` and `VALUE` attributes. The `DataURL` parameter supplies to the Web browser the URL of the data source. The `UseHeader` parameter identifies whether or not to use the header row of the data set as useful information.

The Web page includes a table with two buttons to provide the user with controls to move from team to team. I instantiate both buttons as objects with the `ID` attribute. I instantiate one button as `backward` and the other as `forward`.

I place the bitmap image for the Tigers team logo on the page in an `<IMG>` tag as an object that the `ID=Picture` attribute instantiates. The Tigers team information is the first row of the data set `wblteam.txt`. The `document_onclick()` function is available to any Web page loaded in a Web browser supporting Dynamic HTML. In this case, I create it to keep the team logo being displayed in agreement with the current data row shown on the Web page. The single line `Picture.src = team.value + ".jpg"` handles every mouse click on the page for every team. It changes the `SRC` attribute for the `Picture` object based on the `team` field of the current data row. The `team` field is instantiated with the `ID=team` attribute of its `<INPUT>` tag.

The tags for each data set field are similar. For the `team` field, the connection with the data source is made in the `<INPUT ID=team TYPE=text DATASRC=#teamlist DATAFLD="Team">` tag. The `TYPE` attribute identifies the presentation type for the field. The `DATASRC` attribute

identifies the source of the data. The DATAFLD attribute identifies the field in the data source from which the input box expects to get data.

After I create the four other fields from the data set in a similar fashion, I create two functions for the two user buttons. The backward_onclick() function moves the user back up the data set to the first row. The forward_onclick function moves the user down the data set rows to the last row. The fact that the functions start with the same string as existing instantiated objects is critical. Unless you use exactly the same strings of backward and forward, the function would not be valid.

Automatically Generating and Interactively Sorting Table Rows

With Dynamic HTML, you can automatically generate tables from data sets. As Dynamic HTML creates and places the table on the Web page, the rest of the Web page continues to present its content. Table generation does not stop a user from continuing to read further down the page. After the table is generated, Dynamic HTML permits the user to sort the table based on any field in the table.

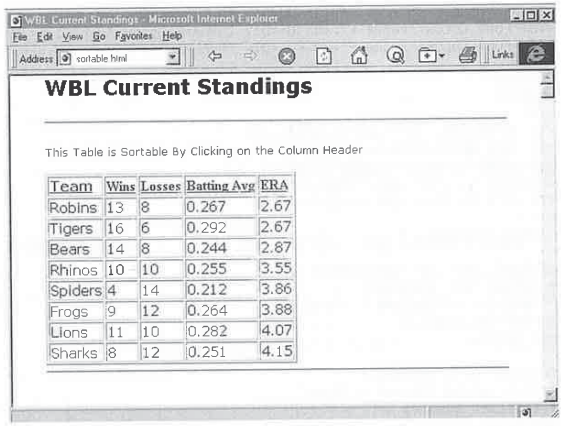
The example in Listing 22.4 dynamically produces a table on a Web page based on the current standing data source that a URL on the Web accesses. Users can access Web Baseball League standings on a daily basis in this manner and can sort the standings based on team name, wins, losses, batting averages, and earned run averages (ERA). In the future, the WBL intends to develop the page further to present individual player statistics and daily box scores. Figure 22.5 shows the Web page after table generation.

Listing 22.4. An example of dynamic sorting.

```
<HTML>
<BODY TOPMARGIN=0 LEFTMARGIN=40 BGCOLOR= "#FFFFFF" LINK= "#000066"
VLINK= "#666666" TEXT= "#000000">
<FONT FACE="verdana,arial,helvetica" SIZE=2>
<HEAD><TITLE>WBL Current Standings</TITLE></HEAD>
<H2>WBL Current Standings</H2>
<HR>
<P>
<OBJECT ID="teamlist"
        CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
        ALIGN="baseline" BORDER="0" WIDTH="0" HEIGHT="0">
      <PARAM NAME="DataURL" VALUE="wblteam.txt">
      <PARAM NAME="UseHeader" VALUE="True">
</OBJECT>

This Table is Sortable By Clicking on the Column Header<P>
<TABLE BORDER="1" ID="elemtbl" DATASRC="#teamlist">
<THEAD>
<TR>
<TD><FONT COLOR="#0000FF"><B><U><DIV ID=team>Team</DIV></U></B></FONT></TD>
<TD><FONT COLOR="#0000FF"><B><U><DIV ID=wins>Wins</DIV></U></B></FONT></TD>
<TD><FONT COLOR="#0000FF"><B><U><DIV ID=losses>Losses</DIV></U></B></FONT></TD>
<TD><FONT COLOR="#0000FF"><B><U><DIV ID=BA>Batting Avg</DIV></U></B></FONT></TD>
<TD><FONT COLOR="#0000FF"><B><U><DIV ID=ERA>ERA</DIV></U></B></FONT></TD>
</TR>
</THEAD>
<TBODY>
<TR>
<TD><SPAN DATAFLD="Team"></SPAN></TD>
<TD><DIV DATAFLD="Wins"></DIV></TD>
<TD><SPAN DATAFLD="Losses"></SPAN></TD>
<TD><DIV DATAFLD="BA"></DIV></TD>
<TD><DIV DATAFLD="ERA"></DIV></TD>
</TR>
</TBODY></TABLE>

<SCRIPT LANGUAGE="VBSCRIPT">
function team_onclick()
  teamlist.SortColumn = "Team"
  teamlist.Reset()
end function

function wins_onclick()
  teamlist.SortColumn = "Wins"
  teamlist.Reset()
end function

function losses_onclick()
  teamlist.SortColumn = "Losses"
  teamlist.Reset()
end function

function ba_onclick()
  teamlist.SortColumn = "BA"
  teamlist.Reset()
end function

function era_onclick()
  teamlist.SortColumn = "ERA"
  teamlist.Reset()
end function
</SCRIPT>
<HR>
</FONT>
</BODY>
</HTML>
```


Again, an <OBJECT></OBJECT> tag pair identifies the data control as it was previously reviewed in Listing 22.3. The table is set up to autogenerate through the DATASRC="#teamlist" attribute of the <TABLE> tag and the use of the DATAFLD attribute for each <DIV> or <SPAN> tag in the second row of the table definition. Through the DATAFLD attribute, each cell in the second row of the table identifies its respective data set field.

FIGURE 22.5.
Dynamic standings sorting.



When specifying the first row of the table, the `ID` attribute of the `<DIV>` tag instantiates each cell as an object. The value for each `ID` attribute is used as the value of the `SortColumn` attribute of the `teamlist` data source object for its respective function. Within each `onclick` function, the `Reset()` method is called for the `teamlist` object, which performs the sort based on its current `SortColumn` attribute value.

Sorting simple league standings is a straightforward use of the technology. The four examples presented in Listings 22.1, 22.2, 22.3, and 22.4 present features that you can mix and match on the same Web page to produce some elaborate, yet manageable, results. Listing 22.5 presents the simple text file used as the data set source of Listings 22.3 and 22.4.

Listing 22.5. The contents of `wblteam.txt`.

```
Team,Wins:INT,Losses:INT,BA:FLOAT,ERA:FLOAT
Tigers,16,6,.292,2.67
Bears,14,8,.244,2.87
Robins,13,8,.267,2.67
Lions,11,10,.282,4.07
Rhinos,10,10,.255,3.55
Frogs,9,12,.264,3.88
Sharks,8,12,.251,4.15
Spiders,4,14,.212,3.86
```


The Future of Dynamic HTML

Dynamic HTML intends for the features provided in this chapter to simplify compound Web-based document composition into a single file format. By providing a reasonable object model and common event handling in the Web page itself, HTML can stand on its own to deliver powerful information content.

The presentation in Chapter 31, "VRML Primer," reflects a phenomenon in which other technologies have been developed for integration with VRML to add functionality not covered by the standard. VRML supporters continue to present enhancements to incorporate the best inventions into the standard itself. Now, with Dynamic HTML, the HTML standard might be following a similar course. The functionality of plug-in applications has added tremendous value to the presentation of Web-accessed information in a Web browser. HTML supporters should continue to find ways to integrate those ideas into the HTML standard itself. Because things move so rapidly on the Web, it makes sense that both HTML and VRML should vigorously continue to entertain new enhancements. Dynamic HTML is a big step forward in providing a vision for HTML-based interactivity, just as the VRML 2 standard is a big step forward in providing a vision for VRML-based interactivity. By the time bandwidth is no longer an issue, the standards should be there to support the development of technologies imagined by optimistic visionaries.

Summary

Dynamic HTML is an enhancement to HTML that adds interactivity and dynamic changes to the Web page. Using the Dynamic HTML syntax, Web authors can compete with CD-ROM titles and traditional business applications to provide value on the Web that the home and office markets demand. Dynamic HTML instantiates Web page components through the `ID` attribute, which is used with many HTML tags. After a component is instantiated, user events can trigger changes to the component such as font, style sheet, size, and `(x,y,z)` location changes.

Dynamic HTML manages data awareness that enables Web pages to filter and sort a data set accessed over the Web. This chapter showed two powerful examples of data awareness in Listings 22.4 and 22.5.

Finally, Dynamic HTML allows for wholesale addition and deletion of HTML tags and text through a `document` object that is always instantiated for the active Web page. With a substantial number of events from which to choose and the power to swap tags in and out, it might take a while until the best uses of Dynamic HTML emerge. Perhaps this is just the start of a new growth for HTML syntax. An interested reader can keep up to date by finding helpful references on Web sites at Microsoft (`http://www.microsoft.com`), the World Wide Web Consortium (`http://www.w3.org`), and Netscape (`www.netscape.com`).

IV

PART

Effective Web Page Design Using HTML

IN THIS PART

CHAPTER 23

Interface Design

by Michael A. Larson

IN THIS CHAPTER

Knowing the HTML tags and their attributes fills only part of the toolbox that you need to build Web pages. Your next step will be to learn how to utilize HTML and related tools to build usable, good-looking, and fast-loading Web pages that effectively fulfill the purpose of your Web site. In many ways, designing Web pages is akin to home construction. To build a house you would use many similar tools and materials: wood, concrete, sheet rock, nails, paint, and so on. Yet there is an almost infinite variety in floor plans and architecture.

In the four chapters of Part IV, "Effective Web Page Design Using HTML," you will learn the techniques for producing clear, informative, easy-to-navigate, and entertaining Web pages.

This chapter deals with the interface of your Web page, so let's assume that someone has decided to visit your Web site. This person might be coming in through the front door—that is, your home page (`index.html`, `default.htm`, `welcome.html`, and so on)—or she might be coming in through a side door, probably from one of the Internet search engines such as Alta Vista, Lycos, or HotBot. It's likely this person also has a mission in mind—to find information or be entertained—or she could simply be curious.

No matter how people get to your Web site or why they come, they must immediately understand what they are looking at when your page loads, or even during the process of loading. They have to know where they are, what that means in relation to their mission, and where they need to go if they can't accomplish their mission on this first page. If you have a good interface design, your visitors will immediately find their information or know where to get it in as few steps as possible. If your visitors can't understand what to do next to accomplish what they came for, they might leave your site frustrated or disappointed—and you have failed in your mission as a Web site designer, whether that is selling a product or distributing important information. Success on a Web site means getting people to visit, making their visit productive, and having them return. To accomplish this, you must pay close attention to a critical factor: interface design.

In this chapter, you will see what elements you need to think about as you brainstorm your Web site theme. You'll also learn about some of the major components of interface design, including controls, color, and style. By combining all these elements, you will be able to build an attractive, utilitarian, and consistent interface into your Web site that will never leave your visitors feeling confused or lost.

What Is Interface Design?

Interfaces are everywhere, incorporated into items you use every day. The dashboard, pedals, gear shift, and steering wheel on your car comprise the interface between you and the essential mechanisms in your vehicle that must work together to get you places every day. The remote control is the interface between you and your evening of entertainment at home after work. Books have an interface (words, ink, and paper) for optimal communication of written material. Even your washer and dryer have an interface. Many of the items you use every day have interfaces. If you think about it, you don't even notice many of these interfaces, because they

are so simple and intuitive to use—like a light switch. When you design an interface for your Web site, you want it to be intuitive and natural. A good interface is

- Easy to use. A visitor can tell at a glance the function of each section of a Web page and intuitively know what to do next.
- Always available without being intrusive. Important items are placed within easy reach.
- Immediately understandable. Everything is logically arranged and well organized in relation to the purpose of your site.

A well-designed interface also can be

- Entertaining or interesting. For instance, having an unusual image in the middle of a business site can coax visitors to see parts of your company that they normally would pass up.
- Colorful or beautiful and nonirritating. Images enhance text and vice versa. Animated GIFs can be irritating to many people trying to read text on a page.

An example of an attractive and useful interface is found on the Microsoft SiteBuilder site (`http://www.microsoft.com/sitebuilder`) as shown in Figure 23.1.

FIGURE 23.1.
The Microsoft SiteBuilder home page. When the mouse passes over a menu option, the color of the menu text changes.

Underneath the SiteBuilder logo at the top of the page is a series of six text choices with small arrows beneath them. When the mouse passes over one of the words, the text color changes from black to light blue, indicating that the choice is available for selection if the mouse is clicked.

This is an excellent example of a successful interface. Visitors have nothing new to learn and the interface is always there—at least as long as you stay in the SiteBuilder section of the Microsoft site. The text defining each choice is clear; there are no cryptic icons to decipher. The color change is also attractive and provides some minor entertainment. This enhanced functionality is added with Visual Basic Scripting. Similar functionality (that uses completely different technology) can also be added with Macromedia Flash or Director Shockwave plug-ins.

The first thing you have to keep in mind when designing your interface is the medium of your method: Your Web site will be read off a screen, most likely a PC monitor, although it could be anything from a television screen to a simple PDA (personal digital assistant) LCD (liquid crystal display) screen. You might also want to keep in mind that the contents of any given Web page may also be printed, but this should be of secondary concern in most cases. Here are some characteristics of screen technology that you need to keep in mind:

- Most monitors and TV screens have color capabilities, but to varying degrees. Many PC systems are limited to 256 colors, whereas TV signals are 24-bit or true color (I'll talk more about color models later). Many PCs are also capable of 24-bit color display, but you don't know what a visitor to your site may be using. If you think some of your visitors will come in through the PDA avenue, consider how your site will look on a PDA.
- The sharpness of different screens varies significantly. Most PC monitors have a much higher resolution than TV, but TV gains a higher level of apparent resolution from its fast frame rate. The graphic you just designed may look great on the high-resolution, small-dot-pitch monitor you are using in True color mode, but how much information will be lost from your graphic for people using poorer-quality monitors or TVs, or fewer than 16 million colors (24-bit color)?
- Screen sizes and resolutions also vary significantly. TV screens come in sizes from 2-inch portable LCD displays to 56-inch or greater monster rear-projection units. PC monitors and graphics cards also have a wide range of resolutions. The baseline resolution for many systems is 640×480 pixels, but higher resolutions are also common, including 800×600, 1024×768, and upward. Monitor sizes also vary from 14 inches for PCs and 12 inches for the Macintosh, through 21 inches. (Larger monitors are also in use, but they are much less common due to their expense.) Additionally, you have to think about laptop screens. Some are strictly grayscale (or only black-and-white) LCD screens, some have color, some are dual-scan, and some have the brighter active-matrix screens.
- Most monitors and TVs have brightness, contrast, and color controls that may or may not be properly adjusted. Will all elements of your Web pages still be visible and usable under less-than-ideal conditions?

As a Web designer, you have to write to a medium that you cannot completely control or define. People creating print documents have exquisite control over their paper and ink, so they can control every part of their information. TV producers also program to the constant standard of color TV resolution (with a nice decay to the black-and-white standard) and have a high level of control over what their content will look like. The WWW presents a unique challenge for the Web designer: access from multiple devices with different characteristics. What this means is that your Web page interface has to be readable/understandable under a variety of conditions. You must have a rugged, simple design in order to reach the widest possible audience.

So with your medium in mind, how do you design an effective interface? First of all, don't reinvent the wheel. Much interface design research and development has already been done by software manufacturers, and you have seen their results in Windows, Macintosh, OS/2, and UNIX X-Windows. They use a variety of interface controls to allow the user to interact with software and hardware on the computer. Some useful lessons can be learned from these common operating systems.

Brainstorming Your Site Theme

The most important component of designing your interface controls is laying a solid foundation for your Web site. This involves setting a Web site theme. A theme is a combination of colors, imagery, and styles that sets the tone of your Web site. A theme is often noticed on both a conscious and subconscious level. Setting a Web theme is a multistep process:

1. Set the goals of your Web site.
2. Define the audience you want to address with the content of your site.
3. Outline the content you want to place on your Web site.
4. Define the look and feel of your Web site.

Setting Your Site Goals

Your site goals can have goals as simple as "I want to tell the world about me" or as complex as "I want a commercial site for selling products, distributing a wide variety of company information, interacting with customers and suppliers, and incorporating support for multiple databases." The key is to clearly define your goals. Get as much input from as many people as possible at this stage. The clearer your goals, the easier it will be to build your Web site.

Here are some questions that you might want to ask to determine your goals:

- Why am I creating this Web site?
- Do I want to concentrate on a single goal or multiple goals? Why? If I decide on multiple goals, what are they, and how can I make them distinct from one another?
- Will these goals always be the same, or will they change over time? How will this influence the Web site, its initial design, and its future maintenance?

You might want to write down as many ideas as possible and then rewrite them clearly and concisely, discarding and narrowing choices as you go along. You will also want to do your first reality check: Are adequate resources available to accomplish the goals set forth? Obviously, you will need to balance your goals with your resources. It is also a good idea at this point to set up a tentative time frame for getting everything done, and see how that reflects on the original set of goals. You might want to design a complex, multitiered site but start out by erecting only a simple site and building on it over time.

An excellent example of using a site to meet multiple goals at a corporation is the Novartis site, which can be found at `http://www.novartis.com`. Their corporate goals are listed on the page entitled Who We Are (at `http://www.novartis.com/weare/index.html`).

If you go back to the company's graphical home page (at `http://www.novartis.com/index2.cgi?4`), you will notice that all its corporate goals have their own link. The company's corporate identity is accurately reflected by its Web site.

Defining Your Audience

After defining the "why" of your Web site, you need to decide which audience to build it for. Your goals will make this step fairly easy. If you're setting up a commercial site for selling products, you should design your site to appeal to the profile of your typical customers. If you are putting up a home page, you might want to decide on a more broad cross-section of the Internet audience. Here are some audience characteristics to keep in mind:

- **Age.** Do you want to appeal to a narrow age group (children, teenagers, the elderly, and so on) with your Web site content or have an "ageless" format that appeals to everyone?
- **Language.** Do you need a multilingual site—with an English version and a Japanese version, for instance—or will one language reach most of your desired audience?
- **Culture or subculture.** People from different cultures will see things differently from how you see them as the Web page author. (This is especially true of icons.) One way to address the widest audience is to avoid slang.
- **Affluence.** If your business is going after sales to a particular income group via the Web, your Web page designs should be crafted to appeal only to this group.
- **Educational sophistication.** If your Web site is scientific or technical in nature, you might want to make sure textual content is very high and graphical content is low to emphasize your theme.
- **Estimated attention span.** For a broad audience, this could be just about anything, and the best thing you can do is to make sure your Web pages load fast. For a narrower audience, try to design your pages to fit what you think the audience's average attention span might be.

Any or all of these audience characteristics may be relevant. Knowing your audience will help you as you write your content, build your interface controls, and design the look and feel of your site. Keeping a particular person in mind when you are actually building site components will help you build a consistent site. (How would so-and-so see this? Would so-and-so understand this? Is this clear enough for so-and-so?) Getting feedback from people who fit your audience type at this stage in the process can be invaluable for deciding on content, writing style, and appealing imagery, and it can have a significant bearing on the ultimate success of your site.

For instance, the Novartis site is primarily aimed at a broad audience of adults around the world. It is geared to be informational, rather than entertaining, in nature. The interface (found at `http://www.novartis.com/agri/index.html`) reflects this in terms of the simple graphics chosen and the wording of the text.

Also note that Novartis attempts to appeal to the widest browser-type audience by allowing the user to choose a text-only or graphical version of the Web site from its home page (at `http://www.novartis.com`).

Outlining Your Content

Now that you know the purpose of your site and your audience, you need to organize and define the content of your site. It is very tempting to dive right in and start experimenting with graphics and styles and fonts after you've gone through the first two planning steps. Try to hold off for just a bit longer and write the outline for your content.

Write down each major topic and subtopic. Decide whether you want to make each major topic a separate Web page (which is recommended) and how many subtopics deserve their own page. You might want to start diagramming your entire Web site as a way to organize your thoughts. This gives you some idea of how much effort will be needed to flesh out the content of the site. Start considering whether some content can best be addressed by hyperlinking to another site, and also what level of detail you want for the content of each section.

Designing the Look and Feel of Your Web Site

This is the stage at which you pour your own personality or your company image into the Web site. If someone visits your Web site and then visits your company in the real world, he or she should immediately feel familiar with the surroundings. You want to communicate not only the hard-wired side of your company or self (logo, product, name, store locations, and so on), but also the qualities that make you unique, and any qualities or accomplishments of which you are particularly proud. Your Web site is the place to strut your stuff. Let the whole world know you are proud of your unique identity by using it to influence the look and feel of your Web site.

You can use all the tools that HTML has available to project your image onto a Web site. This means selecting appropriate fonts, colors, and graphics. For a company, these might revolve around existing objects that already are used in print or other media. For an individual, it might be your favorite colors or the colors of something that means a lot to you, such as your favorite football team or flower. Also, when you think about color, don't forget black and white. You can create some very dramatic and compelling content with these colors.

Now that you have outlined your site, its content, and its image, you can design your interface controls as the finishing pieces. Decide whether you will be using photos, illustration, text, or some combination of these. If you make extensive use of hyperlinked images or image maps in your interface, decide whether you also want to include complementary text hyperlinks. Because text hyperlinks load so much more quickly than images, these are prized by Web surfers who know exactly where they want to go. Also note that up to 30 percent of people surf the Web with graphics turned off in their browsers, and graphically challenged browsers are still used. Again, know your audience.

If you are using a template file to prepare all your Web pages, you need to prepare your interface graphics or hyperlinked text interface and place them on the template properly before preparing the remaining graphics on the site. You can do this before or after you have fleshed out the text content of the site. I don't recommend preparing the remainder of the graphics for your site until all or most of the text content has been written and finalized. This will save you from having to redo any graphics if a new and compelling idea is suggested by the text.

Consider the Novartis site again. What look and feel do you think Novartis is using for its site and why? In my opinion (because I haven't talked with the Novartis Webmaster), the site uses a paper-brochure paradigm. White space is heavily used, and the pages are very simple and uncluttered. The idea, I think, is to emphasize a small amount of important information rather than overwhelm the reader with too many facts and figures.

Interface Controls

Interface controls have two elements: hardware elements and software elements (menus, buttons, dialog boxes, and so on).

For hardware elements, most people will be interacting with the controls in any given Web site interface via the keyboard and some type of pointing device, usually a mouse. The mouse is greatly favored by both Web surfers and Web designers as the primary means of interaction on a Web site. As a Web designer, however, you should also make any controls on a page accessible via the keyboard, if possible, for those few users who don't have a pointing device or for those who might have difficulty using one. Usually the Tab key, Spacebar, and arrow keys step a user through a set of controls or menus. Keep in mind that Java applets or other specialty programs (such as the Visual Basic scripting program mentioned earlier) may only be interactive via the mouse.

The software elements that make up interface controls are by far the most important part of interface design. Most operating systems today are based on a GUI (Graphical User Interface) design that contains the following common elements:

- Menus, usually drop-down
- Toolbars
- Standard dialog boxes for messages
- Controls in dialog boxes for data entry or choice selection
- Onscreen buttons that perform one or more actions
- Icons to represent programs or processes
- Resizable, named windows, usually one window per process or program, usually with scrollbars
- Sound or animations associated with system events
- Fonts for displaying text
- Some way to assign colors, fonts, or styles to various GUI elements

These elements have been around for more than a decade in most major computer operating systems. The other predominant interface, the command-line text interface, is still used in operating systems but is of lesser importance to the Web designer because most Web site access these days is through GUI-based browsers sporting the preceding list of controls.

Unfortunately, there is no way to universally implement many of the interface controls found in GUI operating systems from a Web site. Menus, dialog boxes, associating sounds or animations with events, and other common GUI elements can be added to a Web site via Java, JavaScript, or ActiveX. Not all browsers, however, support part or all of these elements (Internet Explorer comes closest at the present time), so you end up having to build two sites: one that supports the GUI controls, and a second that uses standard HTML elements.

Toolbars and icons can be difficult to implement on a Web site because of a lack of context. Many toolbars are icons only, and without some previous experience, a visitor could be unsure of their meaning. If you're sure that the icons will be understood, you can use image maps to duplicate the toolbar function.

Dialog boxes with data entry fields can be duplicated, without the free-floating characteristic, with HTML forms. This is also true of radio buttons, check boxes, and drop-down list boxes.

For other GUI elements, to some degree, you can duplicate multiple open windows of various sizes by spawning multiple copies of a browser or by using frames. To get around the problem of whether a particular font is present on a user's machine, you can use graphical text, although this quickly becomes bandwidth intensive if overused. As cascading style sheets become more common (and if a Web fonting standard is ever established), this problem will disappear.

For instance, Figure 23.2 shows the use of the GUI menu paradigm on the main page of the Fractal Design Corporation Web site (at `http://www.fractal.com`).

FIGURE 23.2.
The main menu on the Fractal Design Corporation home page.

If you elect to go to the Products page, the menu changes; the color of the text, Products, changes from white to yellow, which appears as a different shade of gray in Figure 23.3.

FIGURE 23.3.
The Fractal Design Corporation Products page menu.

Using standard HTML, Fractal Design has made a GUI-style menu and has been able to demonstrate a menu selection by changing the color of the graphical text. Using common GUI paradigms means that the visitor does not have to spend time figuring out controls and can pay more attention to your content.

Note that the Fractal Design Corporation menu also displays several other characteristics of a good interface control: First, the menu selections are clearly labeled; there is no ambiguity about where you will go when you choose a selection. The menu itself is easy to find in the upper-right corner of the page. The menu design also fits the look and feel of the rest of the page. Also note that the menu has enough contrast to be easily identified against the rest of the page. This is especially essential for laptops, PDAs, or other computers with limited colors or resolution.

NOTE

If you want GUI controls that can't be crafted in HTML, you can build or obtain custom-built controls using Java Applets or ActiveX controls and get the exact control you want with much greater functionality. The disadvantage to this approach is that Java is not supported by many older browsers and ActiveX is currently supported only by Microsoft browsers. Using either can also lead to much larger Web page sizes, slowing page loading time for your visitors. Some security and certification issues are also left to be ironed out for ActiveX controls. As faster access to the Internet becomes available and fixes are made for security problems, using Java and ActiveX to customize your site will become a much more attractive option.

Keep the following concepts in mind when you are thinking about which interface control elements to place on your Web site:

- Unlike an operating system GUI, most of the action on a Web site involves moving the surfer from Web page to Web page or from one part of a page to another. Most controls in your Web site interface will be heavily navigational. Most of the controls in an operating system GUI revolve around modifying some element or initiating some action within an open program, rather than taking a user from program to program.
- Make your controls' functions obvious. Although icon-style graphics on buttons may look attractive, icons can have vastly different meanings to different people. (Remember, the Internet is a global, cross-cultural phenomenon.) Thus, your controls' functions will be clearer if you use text alone or text plus an image to label them.
- You can place your controls just about anywhere on a page. Your choice of location depends on how available you want your controls to be and how the location fits in with the rest of the page. Many people place their prime controls at the top of the page, as you've seen in earlier figures from the Microsoft SiteBuilder site and the Fractal Design Corporation site. It is also common to place a master menu on the left side, both with and without frames.

- The size of your controls in relation to the rest of your page can also vary greatly. If you make your controls too large, they might overshadow other important content on your page. Making them too small means that they might be difficult to find or read. Size your controls for clarity and usefulness.
- Your decision on whether to have multiple sets of controls or menus on the same page depends on the complexity of your site and the number of interactive elements in it. For instance, a very large site, such as the Microsoft home page, shown in Figure 23.4, has a consistent set of controls across the top of every page, each pointing to a different, major function (such as searching) or section. The pages within a section are further indexed by a menu off to the left. No matter what system of controls you decide on, keep them consistent throughout your Web site for optimal clarity.

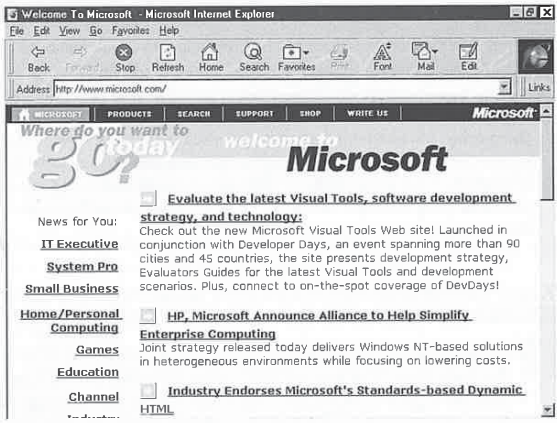


FIGURE 23.4.
The Microsoft home page demonstrating the placement and use of multiple menu systems necessary for a large, complex site.

The controls you use on your site should probably take up only a small part of any given Web page and should complement the colors and the style or mood of your Web site.

Using Color in Your Interface

Color is both the complement and the contrast you need for your interface controls. Effective use of color will give people the visual cues they need to quickly be aware of your interface controls, as well as blend them attractively with the rest of your Web site. Although having good instincts about color is helpful, you should also know how color is defined and used on the WWW.

The first thing you have to learn about is color models. There are two major models—one for the print world, CMYK (cyan-magenta-yellow-black), and one for displays, RGB (red-green-blue). There are also several other important models, but for the purposes of designing Web pages, the RGB model is the one that merits examination. In the RGB model, each color is expressed as a combination of red, green, and blue in either decimal or hexadecimal notation. Whenever color is used in an HTML page, such as designating the color of the font, it is in hexadecimal. For instance, pure red is `FF0000`; each pair of digits represents the red, green, and blue components of each color, respectively. White is `FFFFFF`, and black is `000000` (the absence of any color). Some paint and graphics programs designate each RGB component using decimal notation, so you will need a conversion calculator if you want to apply a particular color from an image to a font or to the HTML page background. You can find online hex-to-decimal (and vice versa) conversions at `http://sicg.calstate.edu/htmlcase/`. You can also find a nice table listing all 216 colors of the Web palette and their hex and decimal equivalents at `http://babeard.simplenet.com/decimal.htm`.

When you are working with a video card displaying 24-bit color, all RGB combinations are available for use. In the Web world, however, many browsers use a 216-color palette called the "Web" or "Netscape" palette. Whenever possible, you should limit your GIF images to these 216 colors (you cannot change the palette in JPG files; they are 24-bit color by definition) for optimal display. Pixels present in an image that are not one of the 216 colors in the palette are dithered. You want optimal control over the appearance of your image rather than depending on a dithering algorithm, so stick with the 216-color palette. Most paint or graphics programs have the Netscape palette built into them.

You can do an entire study on how colors influence people's perception of an image. For instance, red and orange are warm colors that immediately draw people's attention. Blues are cooler, more distant colors. Colors also have a cultural element. In the United States, for instance, brown is associated with soil, nature, and wholesome goodness. Mostly, however, you will want to use color to enhance the appearance of your Web site. As with menu elements, be consistent with the use of color throughout your site. If you choose to use colored text, make sure it contrasts well with its background for easy readability.

Now that you know what colors you have available, how do you choose which colors to apply to your interface controls? Because the interface controls are closely tied to the overall style of your site, you usually let your site style be your guide. You want your interface controls to be recognizable as controls and yet not detract from the layout or appearance of your site by being too utilitarian. For instance, if your site has a desert theme, you could have an image map control that imitates a red rock canyon color to stay within the same theme and provide contrast. You want enough contrast to your controls that people know what they are, but you don't want them to be "other-worldly"—that is, outside of your site's theme. Also, be consistent with the color themes you choose for your controls throughout the site. If you want to tell people what page they are on by "graying out" or changing the color of the menu choice, make sure

the color is different enough that it stands out from the remainder of the menu choices. Try not to use subtle shades in this situation; they might be missed by visitors who have poorer resolution displays or fewer colors.

Incorporating an Image Style

Another part of your interface is the selection of an image style. An image style is merely a definition of how you want to use images to enhance the overall theme or content of your site. If you have a site about astronomy, your images will probably be of various heavenly bodies and spacecraft. The style of the images that you use for controls can vary greatly, depending on your site's tone and intended audience. You can use illustrations of heavenly bodies for the controls, even though much of the content of your site will be actual photos of space phenomena.

How you choose the style of the images and text you use for your controls depends on the content of your site and how you want to communicate that content. Consider these questions:

- Do you want to be formal or informal, even playful, in your approach? If you are being more formal, as in the Novartis site discussed earlier in this chapter, you should choose images with the same tone. On the other hand, if you are authoring an e-zine (electronic magazine) for teenagers, you can use much more outrageous material.
- What audience are you aiming for? Everyone? A particular age group? Use images attractive to your intended audiences.
- Do you want to use photographic images or clip art? Which is more readily available and easier for you to use? Which works better with your site theme? If you're putting together a site about mountain climbing, having a picture gallery of the highest mountains in the world makes more sense than using clip art. If you are putting up an instructional Web page, you might be able to spice it up with some clip art to add variety, especially if your topic is a bit dry. Remember to respect the copyrights on any images you use on your Web site.
- How does the placement of your interface controls on the page influence whether you can use text only, images, or both? Usually placing your controls won't interfere with the rest of your Web page content. The main consideration is that you don't want your controls to be hard to find because of too much clutter on the page.

After you've decided on an image style, you will usually find it easier to settle on a particular file format for generating your interface controls. If you will be using fewer colors, such as illustrations, the GIF format will give you the smallest image size with the highest quality. This is especially helpful if you will be using graphical text (text that is built in a graphics program and exported as a bitmap graphic), because this will give you crisper text. Using JPG files in this situation will blur your text. On the other hand, if you will be using photographs or photo-realistic images, you will usually get smaller file sizes with JPG compression and the blurring effect of this file format will not be as apparent.

A new file format, PNG (Portable Network Graphics), has been approved for use on the WWW, and support for it should start with version 4 of both the Netscape and Microsoft browsers. This file format will address many of the shortcomings found in the GIF format. JPG will still probably be the format of choice for photographs. PNG will support up to 48-bit color (GIF supports only 8-bit); a good, lossless compression algorithm; an alpha channel (for transparency and other effects); and a gamma correction feature to compensate for gamma differences between platforms for improved image appearance. PNG can also be used freely without a license, unlike the GIF format that is now owned by Unisys. The use of PNG probably will not become widespread on the WWW until more of the older browsers go out of circulation. This format will probably eventually supplant the GIF format on the WWW with its superior features and free licensing. You can obtain more information on PNG at `http://www.w3.org/Graphics/PNG/Overview.html`.

TIP

If you currently use a graphics editor that supports PNG (such as Adobe Photoshop) to edit or produce the graphics for your Web site, you can get ahead of the game by saving a copy of your graphics files as PNG before you reduce colors for GIF conversion or increase compression for JPGs. Then when PNG becomes more common, you need to just upload the PNG files, change the file extensions in your HTML files, and you're immediately up to date.

Summary

Defining your site's interface involves deciding what controls you need and how best to make them easily understood and readily available to all visitors to your Web site. The design of your interface controls should be tightly integrated with the style of your site. Because the majority of interface controls on a Web site are navigational, make sure it is clear where they lead. Make sure your interface is both consistent and attractive, even if you just use hyperlinked text interface controls. Spend as much time as possible thinking about control placement and how the controls relate to the content of your site. Build your interface controls based on the GUI paradigm for maximum familiarity. A lost or confused surfer is not a happy surfer, no matter how beautiful your controls are.

CHAPTER 24

Layout Design

by Michael A. Larson

IN THIS CHAPTER

In the print and desktop publishing world, page layout is an art form. You can literally craft documents with sub-millimeter precision to obtain that perfect integration of text and graphics. You have access to myriad fonts and tools for touching them up, or even creating new ones. Control over text is unparalleled. You can control the placement of each letter, the distance between each letter (kerning) and line (leading), and easily add drop-caps and callouts. Then there's the color and picture elements. You can control colors almost exactly (with the help of a good print shop) and nudge and size graphics to place them precisely on the page for maximum impact. On top of all of this, you have exact control over the type of ink and paper. A wide array of tools is available to perform these many tasks, all with many powerful and time-saving features.

Layout for Web pages is, unfortunately, not in such an advanced state of development as printed page layout. Mainly, this is because of the nature of the medium of Web pages (computer screens rather than paper) and the current status of HTML. Also, many users have only 256 colors available on their monitor. Additionally, most monitors are not color calibrated against a color standard. A Web designer simply cannot expect to achieve an exact and reproducible effect on everyone's monitor. HTML was also never designed to be a precise page layout language, but it is certainly evolving in that direction. As such, it is necessary to use several clever workarounds to achieve maximum page layout control. If you're used to exacting page layout control, you will need to lower your expectations a bit or wait until enough browsers support cascading style sheets (CSS) on the WWW. (See Chapter 19, "Introducing Cascading Style Sheets," or Chapter 20, "Cascading Style Sheet Usage.")

You need to remember three things when laying out pages in HTML:

- HTML is relatively new and continually developing. Paper has been around since the Egyptians laid papyrus reeds next to each other for scrolls. HTML has been around for less than 10 years.
- HTML is universal. The language was designed for the lowest common denominator (see next point) so that it could be used by the greatest number of people. Although this is great for communication (and actually fueled the Internet explosion), it is bad news for the sophisticated desktop publisher or artist.
- Computers are not universal. The world is in a sad state of affairs. We have mainframes, minicomputers, UNIX workstations, Macintoshes, PCs (DOS and Windows), and many assorted "miscellaneous" computers of all kinds of different flavors, ages, and capabilities. Some still use monochromatic monitors, and some still don't even have hard drives or wheeze along on a 20MB monster. But they all share one commonality: the capability to run a browser.

This is the "paper" you will be printing to. It is inconsistent and is ever-changing. Screen sizes, screen resolutions, and color support can vary dramatically—and that is just for the one company's computers. The art of page layout in HTML is trying to take some of the design principles from the print world and use them to maximize communication in this chaotic situation.

This chapter reviews the typical elements that you will lay out on an HTML page, including text, fonts, graphics, and other multimedia. You'll see what you can do with HTML page structure without using tables, and then you'll see why tables are a godsend for page layout and how to maximize their use, including nested tables. You'll also look at crafting all of the content on an HTML page as an image in a separate paint or illustration program, and then piecing it together using HTML tables. The chapter ends with a discussion of the way CSS will make many of these page-layout workarounds unnecessary as it ushers in a new age of Web page design.

Layout Elements

Just as in the print world, you are usually faced with a blank page when you open your favorite HTML editor. You need to fill this page with some or all of the elements available to an HTML author. Here's a list of many of the elements that are available for building a Web page:

- Text
- Fonts
- Style sheets
- Static graphics
- Animated or moving images
- Page background
- Tables
- Sound
- Programming and form elements
- Hyperlinks
- Page dimensions
- Specialty elements available only via a browser plug-in

Most of the preceding elements are covered in depth in other chapters in this book, so in this chapter, I simply review how each element influences page layout and some of the things you need to keep in mind as you're laying out a coherent and attractive Web page.

Text Elements

The core of most HTML pages is the textual content. Most people will probably be visiting your Web site because they want to read what you or your company has to say about certain items of interest to them. This is not a book on writing copy, so I can't help you write text. I assume you already know how to write concise copy, it will be provided to you by someone else, or you will obtain it from other documents. In any case, you need to decide how much text to place on a page, how the text relates to the graphical elements, and how you want the text to look. You have some basic control over the appearance of your text, such as a few sizes, bolding, italicizing, and color. Use typical writing rules to decide which control element to use

where. You will not have precise control over kerning or leading. You cannot use textual drop caps or small caps, although you can use graphical representations. In many respects, you have to go back to a typewriter model of producing text for an HTML page. You have a few different heads for your typewriter, and you can change the color of your ribbon.

Fonts in Your Page Layout

Fortunately, starting with version 3 of Navigator and Internet Explorer, you could start to specify font faces by name. If a font is specified in the HTML code and the machine with the browser doesn't have that font, the text will "decay" back to the default fonts specified on that machine. You will not lose information, but the appearance of your page may suffer dramatically, depending on what those default fonts are. The up side of this scenario is that if your target machine does have the fonts installed, they will be used to display the HTML page. Because the vast majority of users will be accessing your Web page from a GUI operating system (OS) with a certain number of basic fonts installed, you can specify several different OS-specific font names each time you use fonts, and the chances of one of them being present is very good. Later browser versions will even let you make fonts available to browsers that don't have them. Be aware that font names aren't standardized. A typical sans serif font might be called Helvetica, MS Sans Serif, Swiss, or Arial depending on the platform. Unless you're targeting a specific type of OS with your Web or intranet site, specifying text fonts is risky.

Cascading Style Sheets

The next page layout element, cascading style sheets, finally gives you the page-layout tool you need to precisely control margins and text elements. At this time, unfortunately, most browsers (Internet Explorer 3 being the main exception) have no support for style sheets. New versions of Internet Explorer and Netscape Communicator (both version 4) will both have built-in support for CSS. Until these and later browsers that support CSS become more widespread, using style sheets exclusively won't be a practical alternative for designing Web sites. If you are working with a narrower audience, such as in an intranet, however, you can start using style sheets right away.

If you have a choice between using style sheets or using many of the page-layout techniques described later in this chapter, you will usually find style sheets to be the better alternative. One significant advantage of style sheets is that the same style sheet can be referenced by multiple HTML pages. This makes it easy to specify the look and feel for an entire Web site from a single style sheet, saving countless hours of work on the page layout side.

While you are waiting for browsers that support CSS to become widely used, an interim solution might be to combine style sheets with your existing Web page layouts or to run two separate Web sites—one with style sheets and one without (similar to running a frames version and a no-frames version of the same site). Using style sheets is a win-win situation for both Web authors and Web surfers: Authors have more tools for design, and surfers see a considerable improvement in Web-page appearance. The sooner Web designers start using CSS, the more quickly surfers will see the benefits and start discarding older browsers in favor of ones that support CSS.

The new HTML 4.0 specification for style sheets adds even more page control for fonts and page display. You can allow the viewer to switch between small or larger fonts depending on the resolution of their current monitors. You can also use style sheets to define which medium is covered by them and have a different style sheet for Web TV, for low-resolution or monochrome monitors, for PDAs or laptops, and so on. With these capabilities, you can finally optimize your site for viewing from most common devices hooked to the WWW. Instead of rewriting many HTML pages—one set for each medium—you need just one set of style sheets to cover them all.

Static Graphics

Static graphics can include images that convey content, ornamental pictures (buttons, bars, and so on), images that can be tiled for the page background, and image maps. The only current standard is two bitmap formats, GIF and JPG. Generally, you will want to keep your image file sizes as small as possible and use images sparingly to improve page load time. When you've decided which graphics to use for a particular page, how large you want them to be, and how many to use, your next question will be where to place them in relation to the text or to other graphics. You will usually employ the same principles for placing graphics as you do text: Keep it in context, make sure it has a use, and make sure it enhances and fits into the current page. Remember, you cannot let text overwrite graphics (except in the case of the page background image), nor can you have text follow a complex path determined by a graphical element (fitting text to paths). You also cannot overlap images (again, except for placing a graphic over the background image), unless you are designing pages that use JavaScript layers and your audience has browsers that support this feature.

Animated or Moving Images

The most common moving pictures used in Web pages are animated GIFs. Moving video, such as AVI or QuickTime movies, are usually not directly incorporated into Web pages because of their huge file sizes. Streaming video technology combined with push technology means that you may eventually be able to incorporate a news feed from CNN into your Web page. For now, animated GIFs can add some action to your Web page. Most animated GIFs need to be quite small in image size because the presence of multiple frames in the file quickly swells the file size. Remember to use animation to emphasize a point or attract attention. There is a fine line between emphasizing something and distracting the user from the remainder of your page. Many people are very irritated by movement on a page. Try looping your animation so that it plays only a certain number or times, or build in a long cycle time between loops. Use the same logic for placing an animated GIF on your page that you use with any other graphic. It also operates under the same restraints.

Page Background

A page background color or image is probably the single easiest-to-implement element of your Web page that can have maximum impact for setting the mood or style of your page. You can use a small graphic to add paperlike textures and colors, add an interesting border to the left side of the screen, or add a simple repeating logo. Any GIF (except an animated one) or JPG can be used. Here are two rules to keep in mind:

- Keep your background image or color in line with the theme for your site.
- Make sure that the text color or font face you are using is easily read against the background.

It's very tempting to use the coolest textures for page backgrounds, but think about whether you want to be remembered as the Web designer who gave someone a headache or eye strain. There are enough problems in everyone's lives without adding one more. Also, some browsers do not support background images. For these browsers, also set a background color that is similar to the color in your image so that visitors will be able to read your text.

Tables

Tables are great for displaying rows and columns of data, textual or graphical. When you're adding a table of data to your Web page, your primary layout concern is whether you want your table to have a relative width (the cells will change size depending on the width of the browser) or an absolute width in pixels, in which case your visitors will need to use the horizontal scrollbar to view all of the table contents if their browser window is narrower than the width of the table. This decision depends on how critical the layout of the table is to the readability of your data, and how much using a relative table will distort other elements on your page if your visitor has a fairly narrow window. Tables are also used as a complete HTML page layout tool; I'll discuss this later in the section "Page Layout with Tables."

Again, the new HTML 4.0 specification adds more features to tables, including the following:

- You now can align columns of data on a decimal point, a colon, or any specified character. This will be useful for people who deal with financial or technical data in HTML tables.
- More options will also be available for defining table rules, which are the horizontal and vertical lines surrounding the cells.
- You won't have to wait for a large table to be completely downloaded into your browser before seeing the data. The data will be displayed as it is received by the browser.
- You can have a fixed header row and scroll the table data below it. No more jumping back and forth to the header row to see the column title. This is a terrific feature for large tables of data.

- You can specify default alignments for entire columns in your table, rather than having to do it on a row-by-row basis. This is a much more logical and useful way to build a table.

Sound

Because sound does not effect any of the display elements on your page, the only layout-related question to ask is when you want the sound to load in relation to the other page elements. If you place the tag earlier in the page, it will play while the rest of the page loads; if you place it later, it will load after other elements. The choice depends on how important the sound is to your Web page.

Programming and HTML Form Elements

Programming script is usually placed into the HTML code before any of the controls it is attached to so that it can be properly initialized before a visitor has a chance to interact with the controls. Tables provide a convenient means of organizing and aligning HTML form controls and their text descriptions. Wherever you decide to place programmed controls or HTML forms, make sure any associated descriptive text will always stick with the control or that the purpose of the control is obvious.

Hyperlinks

You will almost certainly be using text hyperlinks, so you need to decide how you want them to look and how this will influence your page layout. Do you want to stay with the default blue underlined text or use another color? I prefer red underlined text, because it stands out more and encourages clicking. Make sure your choice of hyperlink color is easily visible against your background color or image. Your visited hyperlink color can tend to blend into your background (you probably don't want to make it invisible by blending it in completely) for a grayed-out effect.

Page Dimensions

Unlike the world of paper, where human handling limitations and product use determine the page size, HTML pages have no defined page size and can scroll on indefinitely (you've probably seen some of these mega-pages). Because most people have grown up under the paper book/page paradigm, you will probably want to limit the length of your screen to no more than three to five page-down commands. It is also a good idea to design your page width to not exceed 550 to 600 pixels, because this allows the many people using 640×480 resolution monitors to view your entire page without horizontal scrolling. Another practical limit on page size is to limit the sum of the file sizes on a page to 50KB or less. In other words, add the size of the HTML file with the text elements to the file size (not pixel size) of all the graphics on a given page and any other elements (such as sound files), and that total should not exceed 50KB. This will ensure a snappy response for all the pages on your Web site for the surfer attached to a

modem. It takes a lot of planning and strict graphics guidelines to meet this requirement, but that's part of being an effective page designer.

Specialty Elements Only Available Via a Plug-in

Plug-ins can add some very interesting content and pizzazz to a Web page. If viewing a Web page requires a plug-in, make sure you have a download link to wherever the plug-in is located. ActiveX controls will automatically download and install (with user permission, of course) into browsers that support this standard. Be sure to follow the manufacturer's instructions closely for placing the appropriate code or tags needed to support a plug-in into your HTML file. Otherwise, it is just another page element (albeit one that might help separate your site from the rest of the pack).

Now that you know how each of the elements discussed in the preceding sections might influence your page design or layout, you're ready for a look at what tools are available for page layout without resorting to tables as a layout tool.

Page Layout Without Tables

Prior to the introduction of tables, content developers relied on other means to control alignment and spacing. It's still a good technique to use if your page content is predominantly text or if you don't want to spend a lot of time tweaking page structure. You can still obtain some very acceptable and readable designs using these techniques.

First, here is a brief list of the page organization elements you have access to with this technique:

- Indents
- Lists (ordered and unordered)
- Paragraph breaks
- Line breaks
- Sentence justifications
- Horizontal rules
- Some control of text wrapping around graphics
- Tables used for data elements
- Some special layout control with a single-pixel transparent GIF, a spacer image

Most of these elements have been described in other chapters in this book (except the spacer image), so in this chapter I'll only review what each of these elements does with text and/or graphics in HTML pages. Then I'll show you how to use the spacer images for improved appearance.

Figure 24.1 shows some basic uses of indents, lists, paragraph breaks, and justifications.

FIGURE 24.1.
Some simple text ordering techniques using HTML.

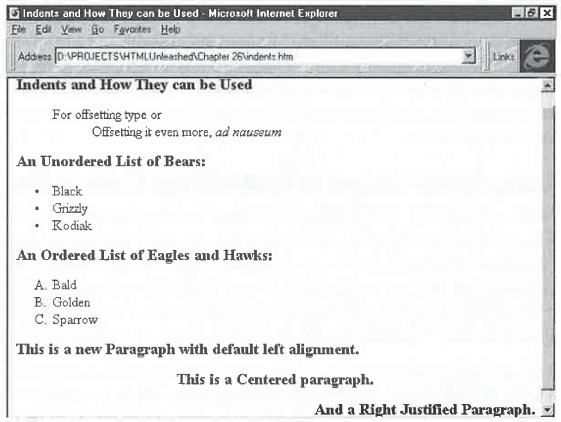
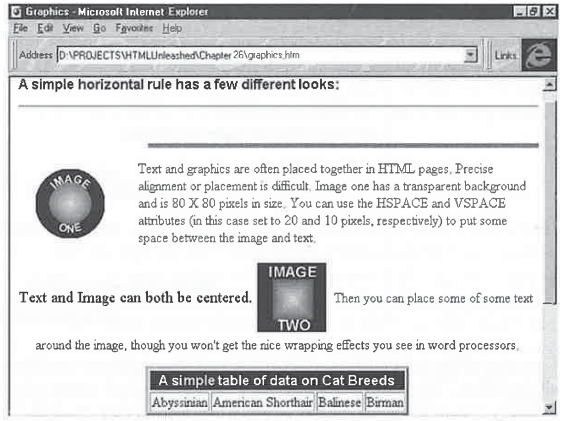


Figure 24.2 shows some of the possible uses of horizontal rules, images and their alignment, text wrapping around images, and a simple table of data.

FIGURE 24.2.
Additional basic HTML page layout elements.



For the greatest speed in getting information up on your Web site, these simpler page organization elements will often suffice. Your page will probably not win any design awards, but there are many times when only a simple page layout is needed, such as for late-breaking news or on pages with heavy content, such as technical information. But if you do want or need more page layout capability than these simple elements afford, spacer images are the next easiest item to use for page layout.

Using Spacer Images to Control Page Element Placement

With standard HTML techniques, you can get orderly pages and have some control over graphics placement and text/graphics interaction. You can gain more precise alignment and control over placement by using spacer images. A spacer image is a single-pixel transparent GIF image that is loaded just like any image but that isn't visible in the browser. The other elements in the HTML page, however, will respond to the presence of this graphic and shift accordingly. This gives you many "invisible hands" for pushing text and graphics around the page.

Let's go back to the text-only HTML page from Figure 24.1 and redo it using spacer images. Figure 24.3 shows the revised page and some of the effects you can achieve with spacer images.

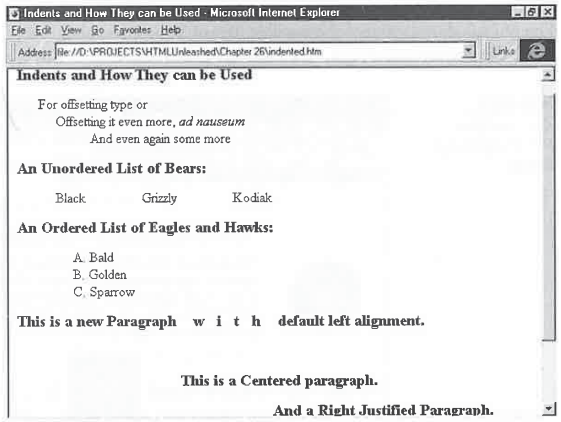


FIGURE 24.3.
Using spacer images for aligning text and other special effects.

I removed all the positional tags (indent, list, and paragraph) and used the spacer image to achieve all the effects. The spacer image is a one-pixel-by-one-pixel transparent GIF. You can achieve different sizes by changing the `HEIGHT` and `WIDTH` attributes in the `<IMG SRC>` tag. To achieve the stepladder effect for the first three sentences about offsetting, I placed the spacer image in front of each sentence using standard image-insertion techniques, and then edited the `WIDTH` to `20`, `40`, and `80` pixels, respectively, for each of the three sentences. This pushed each sentence out by that predetermined distance. This level of control isn't possible using the HTML indent. I used a `<BR>` after each sentence to keep the text close together. I changed the list of bears from a vertical list to a horizontal one by using a 40-pixel `WIDTH` spacer image before the first word and a 60-pixel `WIDTH` spacer image between each of the bear species. This type of horizontal listing can also be done using a table with `BORDER=0`, although sometimes it's easier to use the spacer image. I used the spacer image to offset the list of eagles. I had to type in the list letters (A, B, C) because the spacer image will offset the list letter from the list item rather than offsetting them both from the left margin. The sentence about left alignment shows the use of the spacer for changing the space between letters for the word *with*. I used a spacer image with a `WIDTH` of five pixels before and after each letter. You can also use the spacer image for vertical space control, as you can see from the next sentence. This "centered paragraph" was vertically offset from the paragraph above it by changing the `HEIGHT` in the spacer image to 50 pixels. The last right-justified paragraph has a spacer image with a `WIDTH` of 50 pixels after the period to shift the paragraph away from the right margin by that amount.

As you can see, the spacer image is a handy way to push text around. But how about other elements? First of all, spacer images will not work in concert with horizontal rules for horizontal offset, although you can use them to control vertical offset. You can also use a spacer image to add both horizontal and vertical offset between two images; this works much as the spacing you saw between the bear species text.

One major use of spacer images might be to add custom spacing between text sections on an HTML page. For instance, if you have a heading, some text, and then another heading, rather than adding two paragraph tags between the last of the text and the second heading, you could add one paragraph tag and a spacer image with a `HEIGHT` of your choosing for finer control over the spacing between your sections. This will improve the overall attractiveness of your page. The spacer image is also a very useful and powerful tool when used in concert with a page layout table.

Page Layout with Tables

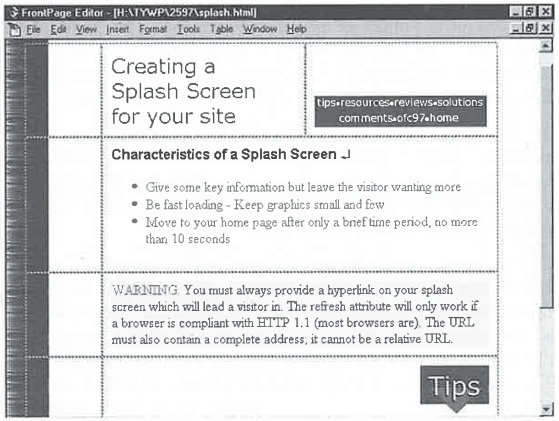
A very powerful tool that should occupy a place of importance in the toolbox of all HTML authors is using the HTML table for page layout. This tool will eventually be replaced by style sheets, but for now it represents the easiest and most flexible manner for placing and aligning all the elements on a Web page. This is made even easier by the fact that almost every HTML editor in the marketplace has easy-to-use features for building and modifying tables. With little more than a click or two, you can merge cells, delete cells, add or subtract entire columns or rows, change element alignment (both vertical and horizontal) within a cell, change background colors in groups of cells, and even create new tables within existing tables (that is, nested tables). By using tables, you will finally feel like you have some serious control over the appearance of an HTML page.

I assume that you know basic HTML table structure and use. If you are unfamiliar with tables, read Chapter 10, "Creating Tables for Data and Page Layout," before proceeding.

General Page Layout with HTML Tables

So what is page layout with HTML tables? It involves placing some portion of a Web page or the entire Web page completely inside one or more tables. The tables will have their borders set to zero, so they are invisible when the page is loaded into a browser. Figure 24.4 shows an example of using a table for page layout inside FrontPage 97 so that you can see the individual cells.

FIGURE 24.4.
Using a table for basic page layout in FrontPage 97.



Here are some general guidelines for using page layout tables effectively:

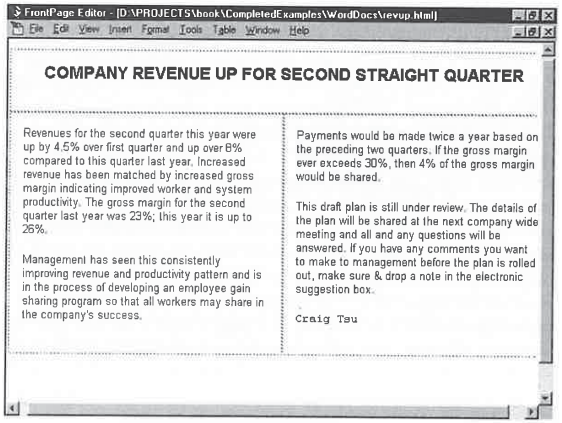
- Use one image per table cell. You can then optimally align the image in the cell, both vertically or horizontally, without influencing any other elements.
- Do not mix text and images in the same cell. Cells should contain an image only or text only. It's easy to create extra cells. Again, this gives you maximum flexibility in aligning elements.
- Use cells for white space. Use a spacer image inside a cell to exactly control the height and/or width of the cell. Leave entire rows or columns empty to separate key sections.
- Fix the width of your layout table to an absolute pixel value; don't set it at 100 percent relative. When you use a fixed width, the structure of your table will not distort if your page is being viewed through a narrow browser window. Your table width should not exceed 600 pixels (I use 550 pixels to be safe). This width is based on the fact that many people still have their screen resolutions set at 640×480 pixels. One note: If personal digital assistants (PDAs) become commonly used to surf the Web, what will the newest lowest common denominator in screen width become—200 pixels?
- Use the cell background color to emphasize text or to add a simple colored division line between sections on your page. This is an easy way to add inverse text for emphasis or interest without having to create a separate image in a bitmap editing program.

Building on the white-space idea, play your table structure off of your page background image for added impact, as shown in Figure 24.4. You will see this idea all over the WWW. By using a background image that tiles to create an interesting element to the left of the page, you can overlay this with a table. By placing a spacer image that is slightly wider than the left element created by your page background image in the far-left cells of the table, and fixing the width of your table with an absolute width rather than a relative width, you can guarantee that any text or elements on your page won't run over your left element. Or, better yet, instead of placing a spacer image in the far-left cells of your table, you can include some image elements in those cells. This will give you access to some interesting blending or offsetting features, sort of a primitive layers capability.

Adding Newspaper-Style Columns for Text Using Tables

Another capability that tables can add to a Web page is the capability to give you pseudo-newspaper columns. This is shown in Figure 24.5.

FIGURE 24.5.
Adding newspaper-style columns with tables in FrontPage 97.



Of course, text will not flow from cell to cell as you edit it. You will need to do everything manually. After you have completely written and edited your text, you will need to do a word count, place the first half of the words in the left column, and place the last half in the right column (or some variation thereof if you use more than two columns).

Nested Tables

Even finer page layout control can be gained through nested tables. Nested tables are created by placing an entirely new table within the confines of a single cell in an existing HTML table. Nested tables are useful for special layouts in the following situations:

- Subsectioning portions of a Web page already laid out with a table. If you want to lay out the sections of a Web page by topic, for instance, you could control the layout of each subsection with a separate nested table.
- Adding tables of data (with or without borders) in a Web page already laid out with a table. This would be a convenient way to associate an athlete's statistics with a picture of the athlete.

Don't forget that the background color of the cell into which you place a nested table becomes the default background color for the nested table. Also remember that some older browsers (now fairly rare) do not support nested tables and that any information in a nested table will not be seen by that browser.

A new capability of tables in the HTML 4.0 specification will be useful in page layout—the use of the relative width characteristics to specify the relative width of both the columns and the font sizes so that the data and table columns will both cleanly scale up or down to match the browser window size. Currently, if you use relative width rather than absolute width to define a table and the browser window is narrower than the table width, the table columns will shrink but the text won't, often resulting in a messy, unaligned display. Under HTML 4.0, both the columns and the text size would shrink or expand together to fill the screen and maintain an orderly display. If you have a text-only portion on your page that you are laying out with an HTML table, you no longer need to make the table width absolute, which could force many viewers to scroll more. Unfortunately, this scaling does not apply to images placed into table cells. Therefore, the text might scale with the columns, but the graphics will not. Theoretically, you might be able to use style sheets to change the `WIDTH` and `HEIGHT` attributes of images to accommodate differences in browser window sizes. If you used the new relative table width for laying out your Web pages and combined this with the new HTML 4.0 style sheet capability, in theory, your entire Web page would scale with every browser window size—assuming, of course, that the browser supports these new HTML 4.0 features.

After you've decided on a page layout using tables, it's very easy to set up a template and fill in the cells with content. Tables give you good alignment and placement control for all of the elements in a Web page. But for even more control, you need to be more of an artist than an HTML author.

Page Layout Using Graphics Only

If you must use an uncommon font to achieve a specific effect, or you have a variety of graphics that must be aligned down to the last pixel, you may be a candidate for graphics-only page layout. What this means is that you create your Web page in another program using all the layout tools therein and then export your completed page, text and graphics, as a single graphic or several graphics. These are then assembled into an HTML shell page (which may or may not use tables for layout). The browser will display the graphic as it would any other graphical element, but this graphic will contain most of the richness of the page you created in your original graphic editing program.

This approach has several advantages:

- You can leverage all of your skills with desktop publishing programs into the world of Web publishing.
- You have as much control as your creation program, such as Quark XPress, gives you. You are not bound by the limitations of the HTML language.

There are two major disadvantages: You still have only 216 colors to work with, and you will have file size limitations. Because you most likely will be mixing text and images on your page, the final image should be in GIF format, because JPG will blur your text if you use any compression at all. Making a full-page graphic can lead to huge file sizes very quickly, especially if you want to stay under 50KB for your total HTML page size. These two disadvantages mean that some of the richness of your page will be lost, but you will gain access to more layout tools. You have to balance which is most important to you.

Programs Needed for Building Graphics-Only Web Pages

If you are going to try to build a Web page using graphics only, you must have, at a minimum, the following:

- A program that is capable of exporting an entire page in a bitmap format.
- A paint or utility program for reducing the colors in your graphic to the 216-color Netscape palette and for translating files to GIF, if necessary.
- An HTML editor that easily creates image maps. This will be the means for adding hyperlinks to your pages.

Making Web pages from desktop publishing programs is not just a matter of exporting pages to a bitmap format. Don't forget the following:

- The text in your exported file must be viewable under several different screen resolutions; there is no way for browsers to zoom in on text that is too small. With this in mind, you will probably want to use a larger font size than you would for print media and place less text information on a page. You will probably have to experiment to find your optimal font size.
- You want to export your page at the size, in pixels, that you will be using on your Web page. You want to do minimal resizing of the final graphic to maintain maximum readability of your text. Your desktop publishing program must give you this control during the exporting step.
- The background color of the HTML page that you place your graphic into should be the same as the graphic's background color. Some browsers offset their pages differently than others. Keeping the HTML page background color and the graphic background color the same will minimize any artifacts (lines, border effects, and so on).
- Make sure the type of image map you select will work on your Web server. Most browsers support client-side image maps that do not require any interaction with the Web server to read the coordinates of the hotspots. If you want or need to use a server-side image map, make sure your HTML editor or image-map-making program supports that coordinate system.

By using the graphics-only approach with care, you can build your entire Web site this way. You may also want to build a text-only version for text-only browsers or for people who surf with graphics turned off. An alternative might be to add text hyperlinks to your pages that reproduce any image map hyperlinks.

Be on the lookout for programs that can significantly simplify the process of creating graphics-only Web pages. For instance, in CorelDRAW! 7 and Corel PhotoPaint, you can assign URLs to any object. If you then publish that image to the Web, any objects with URLs become hotspots on an image map. These programs do all the coordinate drawing and also save an HTML file with the map coordinates in addition to the graphic as a GIF image.

Summary

As you can see, page layout under HTML can be approached from several different directions. It can be as simple as adding some text and graphics and using some basic justifications, paragraphs, and horizontal rules. Or it can be as complex as doing all your page design in a desktop publishing or drawing program and exporting it to a Web-compatible bitmap format as a large image map. Tables are a good in-between choice, because they have excellent alignment and organization capabilities and are easy to use. Tables also give you access to some special effects,

such as crude layering with patterns in the page background image. The spacer image is also a useful tool for moving text around a page or for filling in a table cell to prevent it from moving. Cascading style sheets provide the most control and the cleanest appearance with the least effort and will probably become the predominant means of page layout in the very near future.

Try to keep page layout in perspective with your needs. If it is critical to get information out to an audience quickly, you may want to use simpler page layout techniques and forget using tables or CSS. If a more complex layout is critical to communicating the information on your Web page, pull out your whole arsenal of tools. Good use of page layout tools and design is essential if you want eye-catching pages that will attract more users and keep them on your site. A well-organized page also makes it easier for your users to find the information they want. Remember: Page layout should add to the enjoyment of your Web pages and complement your content.

CHAPTER 25

User Navigation

by Michael A. Larson

IN THIS CHAPTER

No matter how beautiful your graphics, how powerful your prose, or how cool your Web site, if visitors can't find their way around or to an important part of your site, all your efforts have been for naught. If you have a large Web site and someone has followed one train of thought through your site and then needs to get to an entirely different part of your site but can't see an easy way to get there, again you have failed in the navigation design of your Web site. Most visitors do not want to play Dungeons and Dragons through your Web site (unless you are sponsoring a treasure hunt or other game), finding information by solving the maze that is your navigation interface design. Visitors who become frustrated because they can't get around easily can negate all of the time, effort, and expense that you put into crafting your site.

Each Web site is like a little town. Towns have many obvious and intuitive similarities. They all have streets, houses, businesses, and vehicles. Most people can figure out the basics of how to get around a new town because they know the common elements of all towns. But people might have trouble finding the details, because on the Web each "town" or Web site can have a very different purpose.

As the Chamber of Commerce for your town, you must make sure to give your visitors an understandable and quick (with as few clicks as possible) means of navigating through your particular street layout, pointing out attractions as well as preventing your visitors from ending up in dead-end alleys. Most visitors will know the basics of Web site navigation, hyperlinks, buttons, and image maps, just as they know the basics of getting around a town using street signs, roadmaps, billboards, and phone books. Do not assume that your visitor understands your town as well as you do. Be courteous, and think about the best way to guide your visitors.

This chapter explains the underlying principles of optimal site navigation design. You will look at ways to incorporate navigation systems into your Web site, including using text hyperlink menus, groups of images and/or graphical text with hyperlinks, image maps, or some combination of these items. You'll also see examples of these systems and how they can be applied to an entire site, whether you're building a small Web site or a megalith. Finally, you'll see how having your own search engine can enhance the ease of navigation at your site with a minimum of HTML programming.

Fundamentals of Navigation Design

A Web site navigation system consists simply of a set of hyperlinks that enables a visitor to view and select all the different content sections of a Web site. The design of the navigation system varies with every Web site. The process of building Web site navigation systems involves both laying the roadway and clearly marking it. The navigation system must also be integrated with the other components of your site. Site navigation should be a clearly visible, integral part of your Web town. Don't forget that your Web site navigation system can be both attractive and useful; you do not have to sacrifice beauty for utility.

The first step in designing a navigation system is to create an outline for your Web site. The content and design of your Web site will often suggest the best means to navigate it.

During the process of outlining your site, write down the structure and relationships between parts of your site. You can do this with words, symbols, or a combination of words and symbols. (Flowchart programs are very helpful here.) I find it easiest to approach outlining by topic. I usually list all of the major topics, and then place the subtopics under the major topics. After that, I prioritize the topics by how interesting I think they will be to my visitors and how important each topic is to me. You might also end up with more branches under your subtopics. The number of branches often suggests whether you can get by with a simple navigation system or whether you need multiple navigation systems to help visitors get around your site.

After you have your site outlined, consider the following questions to define your navigational needs:

- Do you see your Web site structure (not content) changing with time? That is, do you see yourself erecting whole new sections and tearing down others, or will you mostly just be changing or updating the content of existing sections?
- How will most users travel around your site? Can you see any obvious traffic patterns? Do you have a major theme that will attract the majority of users?

Answers to these questions will help you determine what navigation system (or systems) to build. For instance, if you have just a few sections, one navigation menu will easily lead the user around your site. If you have a larger site with more sections and subsections, you may need two separate navigation systems—one for major topics, and one for subtopics. If your site structure is fairly fluid, you will want to use a navigation system that is easy to change. If you want to direct visitors to an important feature of your site, you may want to emphasize it in your navigation menu by using a brighter color or bolder text.

TIP

For a small Web site with just a few topics, you might need only a simple set of text hyperlinks or a small image map (or both) placed across the top or bottom (or both) of each page. When your needs grow past one menu, you need to think more about placement, how to relate your navigation menus to each other, and what visual cues to give your readers as to their location at any given time. For instance, you could build a primary navigation menu composed of text and icons. Then for each page under this main topic, you could place the icon on the same place on every page. This icon could then be hyperlinked back to the beginning page for this section, no matter how deep your visitor goes. This uses the visual cue idea, and it provides a convenient navigation means for your visitors. Whatever system you finally choose, remember to keep your navigation system as simple and easy to use as possible.

Two other items will influence your choice of navigation systems: placement and clarity.

How and where you place your navigation system will very much depend on your content. For instance, if you have a mostly textual or a linear site (the content of one page leads to the next and so on to the last page of the site), placing the navigation system at the bottom of each page may be the easiest for your user. If you have major sections and/or subsections, placing the navigation system to the side (usually the left) or across the top of a page may work better. For a complex site, you may need to have multiple points on the page (top, side, and bottom) available for navigation selection, as well as a page dedicated to mapping your site or searching it using keywords. Your choice of navigation systems will be suggested by traffic patterns, your content, and how complex or large your Web site is.

Another key element in choosing your navigation system design is to define the level of clarity and understanding you want from your navigation system. By clarity, I mean, "How easy is it for anyone to understand your navigation system?" You need to balance clarity with the design or theme of your Web site. Remember, the WWW is global in nature, and many of the visual shapes and colors common in your culture will not mean anything (or worse, will mean something very different) to visitors from other countries. If you have a site that depends heavily on graphics, using a graphical-style navigation system will enhance your site's appearance, but you may sacrifice some clarity. You can usually obtain the best clarity with text hyperlinks. If your text is well written, most people will immediately understand where a hyperlink leads within the context of your site. Images are not usually as obvious as text to a visitor, although they maximize the design and beauty of your site.

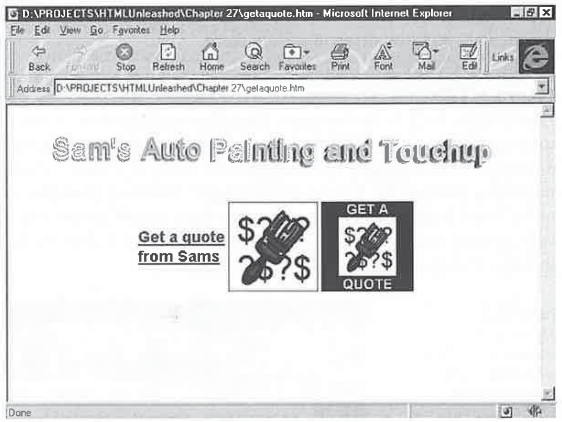
A good compromise is often a combination—an image with text incorporated into the graphic. The images can then be color-keyed or be simple shapes that represent each area of your Web site as an additional visual hint. You can see an example of this in Figure 25.1. Say you own an automobile painting business, and you want people to be able to request quotes from a page on your Web site. Which of the three navigational choices shown in Figure 25.1 best conveys the content of this quotes page to the visitor?

NOTE

Don't forget that navigation applies to moving through a single page as well as between Web pages. If your content requires you to build long pages with multiple sections, use the HTML bookmark to build a navigation menu of the sections at the top of the page. Some people also place a "Back to Top" hyperlink after each section and at the bottom of the page.

You've seen some of the factors influencing the placement and clarity of your navigation system. The next step is to look at some of the navigation systems commonly used on a Web site.

FIGURE 25.1.
Three ways to steer a visitor to the "Request for a Quote" page.



Navigation Systems

Navigation systems must fulfill a variety of demanding roles. First-time visitors to your site must be able to easily find everything. Repeat visitors should have an easy route to get to their favorite part of the site without going through 20 hyperlinks to get there. The navigation system should be easy to use and complete, but shouldn't contain too many selections. Putting together an effective navigation system is one of the many balancing acts that a Web author has to go through. There are four universal systems that you can mix and match for a navigation system:

- Text hyperlinks, separate or grouped together
- Hyperlinked, individual images
- Image maps
- The Submit button on an HTML form

The Submit button is usually not thought of as a navigation aid, although after clicking the Submit button as part of the transaction with the Web server, you are usually served a new HTML page; that is, you have essentially navigated somewhere. HTML forms can be used for restricting access to some parts of your site. (Registration may be required, for instance.) Don't forget to include these restricted routes when you're mapping out the navigation for your site.

There are also two, nonuniversal, special containers for navigation systems to consider:

- Frames
- Specialty programs such as Java, JavaScript, ActiveX controls, or other specialty programs supported by a plug-in

Some combination of these six items will likely comprise your navigation system. Let's look at the strengths and weaknesses of each in detail.

Using Hyperlinked Text for Navigation

Hyperlinked text is by far the easiest kind of navigation system to put together. You just type some descriptive text and add the hyperlink. The default style of the hyperlink (usually a blue, underlined font) is automatically applied to the text. You're done. It's easy to go back and tweak the text for more clarity. You can sprinkle hyperlinked text throughout your Web pages. For a more formal menu appearance, you can combine a set of common destinations in a list, in a table, or in a nested table. Figure 25.2 shows a navigation table of text hyperlinks at the bottom of a Web page.

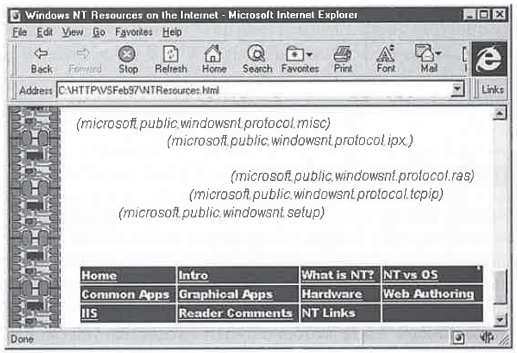


FIGURE 25.2.
Hyperlinked text selections grouped in a table create a basic navigation system.

Text also loads very quickly, leaving you more room to use graphics for other things on your Web page. Text hyperlinks, of course, can be viewed and used by every browser on the market. The only disadvantage of hyperlinked text is that you have limited control over its appearance under HTML.

NOTE

When you have more than about 10 text hyperlinks in your navigation menu, you need to start thinking about breaking up the menu by topic. This will speed users to their destination on your site. If your text hyperlink list becomes too long, you might tire your visitors by making them read every item.

Using Hyperlinked Images for Navigation

The next most commonly used navigation elements are hyperlinked images. These act in much the same way as icons do: Clicking on them produces one action. You can sprinkle these throughout your pages or put a hyperlinked image in the same place on every page to point, for example, to the site home page. You can string these together in a table to perform a function similar to that of an image map. Images can be graphical text (text saved as a bitmap image from a graphics editing program), an elaborate, irregularly shaped image, or some combination of the two. The main advantage of hyperlinked images is their excellent combination of design and usefulness. Their main disadvantages are that they take up more bandwidth than text, it is very difficult to make an icon (without text) that everyone understands, and you need a bitmap editing program if you ever need to tweak them. Don't forget to use the `ALT=` attribute in the `<IMG SRC>` tag for adding descriptive text so that browsers that don't support graphics or have graphics turned off know where the hyperlink leads.

Using Image Maps for Navigation

Image maps are arguably a Web designer's most useful and beautiful navigation aids. (See Chapter 14, "Creating Image Maps," if you need more information about image maps.) When visitors see an image with many easily identifiable parts, they know immediately that this is probably a navigation menu. There is also no restriction on the shape of the hot zone you can map out for each part of the image, so you have complete freedom in designing the image. An image map is fairly easy to build using any modern HTML editor. Some paint programs let you associate layers or image objects with URLs and create the image map code for you. Often, the primary disadvantage of an image map is its file size. Because there are usually multiple and differing parts to the image map, it is difficult to reduce it in size or color and maintain enough detail to make it usable. It takes careful planning to avoid this pitfall.

TIP

Don't forget to start using cascading style sheets to supplement your navigation menus as soon as enough browsers support them. With CSS, you can layer text over your images. This means you no longer have to decide whether to use GIFs for optimal text clarity or JPEGs for optimal photographic quality in your image maps. Now you can have both. The use of PNGs will also eliminate the need to choose between GIFs and JPEGs for image maps.

One disadvantage to using image maps is that you can use only one `ALT=` attribute in the `<IMG SRC>` tag for the entire image. People surfing with graphics turned off won't be able to use an image map to navigate, because they won't be able to see the image sections or hot spots based on the `ALT=` attribute. You need to have backup text hyperlinks or a separate text-only version of your Web site.

Using Frames for Navigation

Hyperlinked text and images or image maps can all be used in concert with frames (see Chapter 18, "Creating Sophisticated Layouts with Frames and Layers," for more information) to build a complete navigation system. Frames are a natural for this. You can place menus across the top, across one or both sides, and/or across the bottom of the frameset page. These menus remain static while the results of any menu selection can be displayed in a central window. Figure 25.3 shows one way to use frames to display navigation menus and content in one browser window.

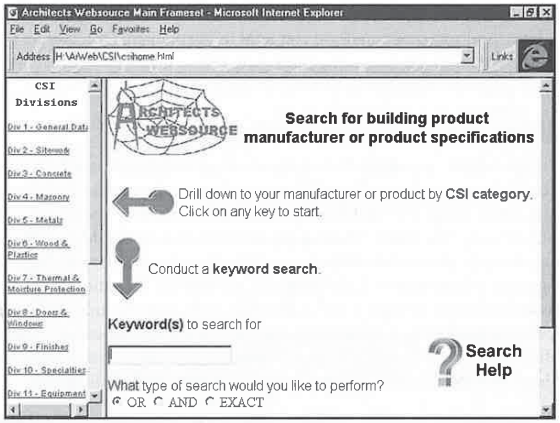


FIGURE 25.3.
Using frames to display both the navigation menu and the contents of any selection.

With frames, you don't need to include a copy of your navigation menu(s) on every page of your Web site. Also, changing or expanding your navigation menu(s) involves changing only one page and one set of hyperlinks. This helps reduce the onerous task of Web site maintenance.

Although frames are very useful from the Web designer's perspective, they meet with mixed results in reality. You have to be careful in programming the target frame for each of your hyperlinks, or you can end up loading multiple copies of the frameset page and get the dreaded "hall of mirrors" effect. Also, a fair number of browsers don't support frames, which means that you need to produce both a frames version and a no-frames version of your Web site. Frames also seem to be a sore point with some users; the number of user complaints about this feature in newsgroups is exceeded only by complaints about animated GIFs. Given these drawbacks, many Web authors have removed frames from their pages, returning to the use of nested tables or Java applets to achieve similar effects. However, many sites still successfully use frames.

In the HTML 4.0 specification, a new element called `iframe` is added to frames. An `iframe` is an independent HTML document window that appears within the body of another HTML document rather than as part of a frameset. Using an `iframe` brings up interesting navigation possibilities. You could have a small `iframe` in every Web page, which could contain navigation menus. This `iframe` could contain a main navigation menu in the `iframe` and have secondary navigation menus in the HTML page containing the `iframe`. Or the navigation menu in the `iframe` could be different for every page on your Web site, which means that you could essentially have a rolling, customized navigation menu for every page on your site if you choose. An `iframe` would also make an excellent container for specialty navigation systems built using Java, JavaScript, or ActiveX controls.

Using Java, JavaScript, and ActiveX Controls for User Navigation

The last type of navigation systems container uses Java programs, JavaScript, CGI scripts, ActiveX controls, or plug-ins. Some extremely useful and unique effects can be achieved using these controls. There are ActiveX controls that act like Windows Explorer: When you click on the top menu choice, a sublist appears, allowing you to place and organize many choices into a relatively small area on your page. Macromedia Director and Flash, via their Shockwave technology, enable you to design multimedia components that respond with a color change or movement when you pass the mouse cursor over them.

All these components allow a greater degree of interaction and entertainment than is possible with text hyperlinks, image hyperlinks, or image maps. Unfortunately, they are not universally supported. Java probably comes the closest to being universal, but you must be a trained programmer or be able to purchase the program to get that interactivity. ActiveX is currently only supported by Microsoft browsers. Plug-ins are not something that you can expect your visitors to have available with their browsers. In addition, many plug-ins may not be supported across every platform. If you choose to use any of these options on your site, make sure that you make available alternative pages that don't contain these options, or warn your visitors that they need a special component to view the page. It is also often possible to detect browser types (and to a lesser degree, plug-ins) using JavaScript or VBScript and send the browsers to the appropriate page.

Although it is extra work to support all of these specialty programs, the effects are often unmatched. Using these components is one way to build a site that stands out.

Adding Visual Cues to Your Navigation Menus

The means by which you show visitors which page or section of the Web site they are currently on is another important factor. With a text-based hyperlink system, you can remove the hyperlink from the text selection for the page you are on. For instance, in Figure 25.2, the visitor knows that he is on the page with NT Links because it is the only selection without a hyperlink and it is also a different color.

Instead of removing the hyperlink, you could change the color of the text to a color more closely approximating the background color, giving it a grayed-out effect.

This system can be applied to hyperlinked images. If your images are colored, you might have a grayscale version indicating the current page. Or if the image has text in it, you might want to blur it. Another option would be to apply another color. Whatever indicator system you use, make sure it is the same for all pages in your site.

Image maps are the easiest to modify in this manner, although it is usually not practical to change the image. To indicate which page or section is being viewed, all you have to do is remove the hotspot coordinates for that section from your image map. I don't recommend changing the image used for the image map unless you have a very small graphic. If you do have a small image file size, it certainly gives your visitors more information if you can change the color or style of the image map to represent the section they are currently in.

Navigating Through an Entire Web Site

This section discusses tools that help users get around a huge Web site. You can use standard navigation menus for small or medium-sized sites. It is also often possible to make all of the Web site pages or sections available from one or two menus. But when your site swells to several hundred pages or more, trying to include links to all sections becomes cumbersome; you need to offer other options for getting around your site. The two commonly used options are

- A site map
- Your own search engine

Using and Building Site Maps

A site map consists of one or more pages whose primary function is to give a user a quick, at-a-glance view of your site. The site map, also called a *site atlas*, is usually based on some type of hierarchical structure. It can be an image map, a strictly text-based outline of your site, or both. The idea is to let users see your entire site outline and be able to go directly to the areas that interest them most.

If you decide on a text-only site map, you can build it with a variety of techniques. You can use the listing capabilities in HTML coupled with indenting (`<BLOCKQUOTE>`) and font sizes to quickly build a useful site map, as demonstrated in Figure 25.4.

You can also use tables to organize your text-based site maps and add some small images to generate interest or draw attention to the major section headers, as shown in Figure 25.5.

FIGURE 25.4.
A text-based site map using lists, indents, and font sizes for organization.

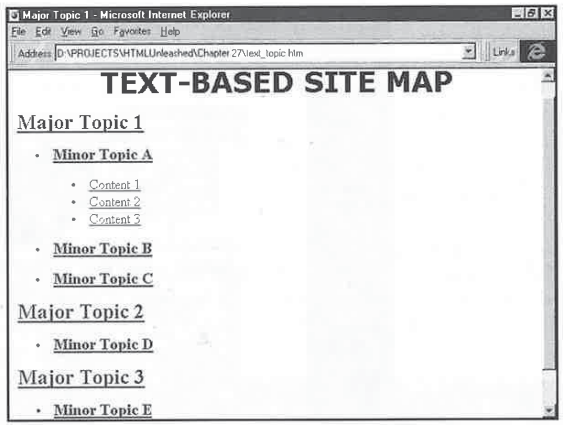
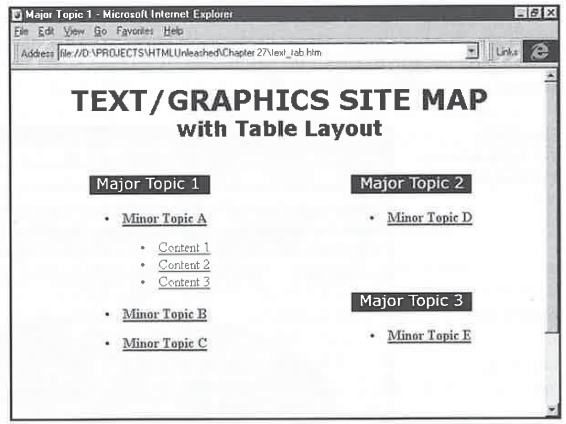


FIGURE 25.5.
Using a table to organize a combination of text and small graphics for your site map.



The more commonly implemented site maps are image maps with hot spots leading to the various sections of the site. Your image map can mimic a text-based hierarchy by using a flowchart metaphor. This is illustrated by the Lycos (`http://www.lycos.com`) search engine's site map, shown in Figure 25.6.

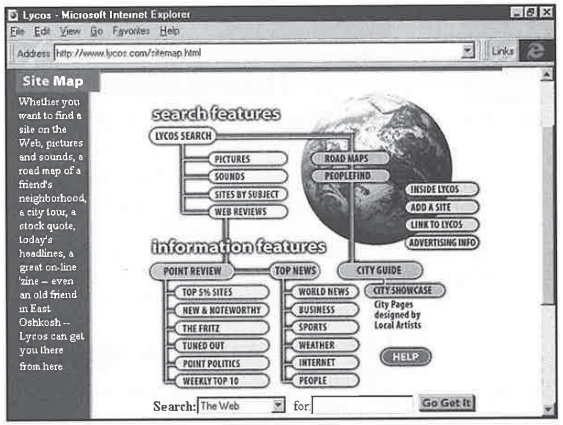


FIGURE 25.6.
Lycos uses a flow-chart metaphor for its site map.

Just because site maps that use image maps need to be clear doesn't mean that they can't be interesting. The Indian government used one of the country's national treasures as the background for a site map that leads the visitor through a budget report (`http://www.m-web.com/sitemap.html`), as shown in Figure 25.7.

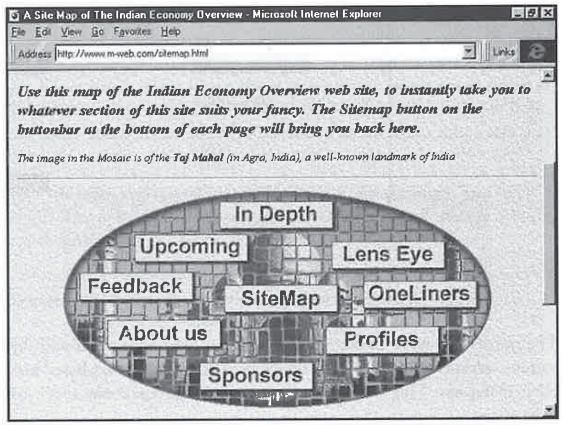


FIGURE 25.7.
The Taj Mahal acts as a backdrop for a site map.

A good site map should have the following characteristics:

- It should be easy to follow.
- It should have enough detail to be informative without being either too verbose or too succinct.
- It should have a help page available, if necessary.
- If the site map is multipage, the navigation system to move around the index should be distinct from the index itself or from any other navigation menus you use on your site.

Color-coding the graphics or text for each section of your site map will make it easier to follow. Also, decide whether you want to arrange your site map content alphabetically or topically. Typically, a topical arrangement will work better for a Web site. Your help page should include an overview of how your content is arranged and why it is arranged that way, as well as any hints for moving through it.

CAUTION

One fact to keep in mind as you decide whether to have a site map is that a site map will increase your Web site maintenance requirements. Every time you add or delete a page, you will have to update the site map. For a site with rapidly changing content, this may prove more burdensome than any convenience or usefulness the site map offers visitors. This factor should also influence how you design your site map. You may want to separate the site map hyperlinks leading to rapidly changing parts of your Web site from the more static portions of your site map. There is no requirement that all of your site map has to be on one page.

Adding a Search Engine to Aid Navigation

When your Web site becomes too large or too active to adequately describe with a site map, you might consider using a search engine. The same technology used by Alta Vista (`http://www.altavista.digital.com`), Lycos, Excite (`http://www.excite.com`), or any of the other major Internet search engines can be used to index all of the text content on your Web site. This index would then be searchable by anyone on your site from an HTML form using keywords. A list of "hits" from your site is returned by your local search engine. The hits are active hyperlinks to other pages on your site. Using a search engine has several advantages:

- Visitors can use their own words for searching.
- You probably won't need a hierarchical site map.
- Search engines are usually faster to use than site maps.
- Search engines require little maintenance or attention.
- Search engines automatically update themselves.

Almost all Web servers support at least one search engine. If your Web site is hosted by an ISP, the ISP will usually have a search engine that you can use with instructions on how to add it to a Web page. It takes only a few lines of HTML code to add a search box to your HTML page. For instance, Microsoft uses Index Server as the search engine in support of their Internet Information Server (IIS) Web server. To add a keyword search box to an HTML page, you need only the following few lines of code:

```
<FORM ACTION="/scripts/samples/search/query.idq" METHOD="GET">
    Enter your query below:
    <TABLE>
        <TR>
            <TD><INPUT TYPE="TEXT" NAME="CiRestriction" SIZE="60"
            ➥ MAXLENGTH="100" VALUE=""></TD>
            <TD><INPUT TYPE="SUBMIT" VALUE="Execute Query"></TD>
            <TD><INPUT TYPE="RESET" VALUE="Clear"></TD>
        </TR>
        <TR>
            <TD ALIGN=right><A HREF="/samples/search/tipshelp.htm">
            ➥Tips for searching</A></TD>
        </TR>
        <INPUT TYPE="HIDDEN" NAME="CiMaxRecordsPerPage" VALUE="10">
        <INPUT TYPE="HIDDEN" NAME="CiScope" VALUE="/">
        <INPUT TYPE="HIDDEN" NAME="TemplateName" VALUE="query">
        <INPUT TYPE="HIDDEN" NAME="CiSort" VALUE="rank[d]">
        <INPUT TYPE="HIDDEN" NAME="HTMLQueryForm"
        ➥ VALUE="/samples/search/query.htm">
    </TABLE>
</FORM>
```

A table is used to organize the elements in the preceding HTML form. The form from this code as it looks from a browser is shown in Figure 25.8.

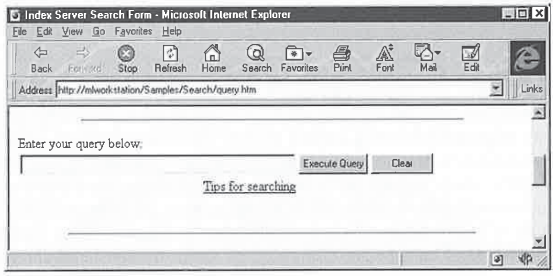


FIGURE 25.8.
Adding a keyword search text box to an HTML form.

Notice from the HTML code that you can customize a number of options for the output of the form, including the following:

- `CiRestriction` enables you to exclude certain parts of your Web site from being searched or to exclude certain users from searching.

- `CiMaxRecordsPerPage` controls how many hits per page are displayed. The default is `10`.
- `CiScope` defines the starting point for the search, typically the root of your Web site.
- `CiSort` indicates how to sort the returned information, by rank or filename.
- `HTMLQueryForm` is the path to the template.

You can allow a visitor to specify some of these parameters by adding other controls to your HTML search form.

Most of the other search engines have similar parameters, such as the following:

- Whether to search by a single word, several words, or a phrase
- Options to match your search keywords along a scale from exactly to very loosely
- Whether to return hits with brief or verbose descriptions
- The capability to search the entire Web site, parts of it, or specific elements of it, such as images
- The capability to filter pages by date
- The capability to have the results displayed in a new browser window

Be sure to include a link to a help page for tips on using your search engine.

Most indexes automatically update themselves and do not add much to your site-maintenance duties. As such, they are often much easier to use on a Web site than a complete site map. I have seen numerous Web sites that provide both search capability and a site map. This gives visitors the greatest flexibility and the most options in finding something on a site. Within reason, you cannot have too many navigational aids.

Summary

Thinking about navigation systems is not very exciting, but your Web site will quickly become useless without one. Currently there is no single standard for designing and building a navigation system; Web authors must devise their own. There are, however, some essential guidelines to follow. If you incorporate good design techniques with the utilitarianism of the navigation system, the navigation system becomes unobtrusive, yet easy to find and use. Remember that a navigation system should never lose any of its clarity or ease of use during the creation process.

Though graphics or graphical text give you the most attractive navigation systems, don't forget to use text in your basic navigation systems or site maps. Many Web surfers know exactly where they want to go and don't want to wait for graphics. Text hyperlinks speed these users to the key information, and they will (hopefully) remember your site because of it. Have you ever visited a site where you just didn't have time for some large image map to download, so you went to the next site instead? Your navigation system should never be a reason from someone to leave your site. Give people the text navigation choices, at least until more people have faster connections to the WWW.

CHAPTER 26

Putting It All Together: HTML Design

by Michael A. Larson

IN THIS CHAPTER

Knowing about design theory, page layout techniques, color models, and Web site navigation systems is all fine and good. Effectively applying these principles, however, is where the tread meets the road. If you know how to combine these principles with your site content, you will be building an excellent Web site in no time.

So, what makes one Web site excellent and another not so excellent? Discerning this takes a practiced eye, and the best way for you to practice is to dive into a couple of well-designed Web sites and take them apart to see what makes them tick. I have found two superb Web sites just waiting for your review: Travelution (`http://www.rosenbluth.com/`) and CyberMad (`http://www.CyberMad.com`). Travelution is a one-stop shop on the Internet, providing extensive travel tips, package tours, and reservations under one roof. This large, fairly complex business site put together by a third-party Web development company for Rosenbluth, International and currently under the care of webmaster Kate Urzillo is representative of a well-done corporate site. CyberMad is an award-winning electronic magazine (e-zine) by Christopher Parr. This Web site is moderately large and extremely diverse in its content and presentation styles; it is more representative of the entertainment genre on the WWW. The entire contents of both Web sites are on the CD-ROM accompanying this book. They are completely browsable from the CD-ROM (using Netscape or Internet Explorer, both Version 3.0 or later, is strongly recommended) with hotlinks to their current versions on the WWW.

In this chapter, you play the role of a Web site reviewer. Using the principles from the earlier three chapters as guidelines, I take you through some of the pages from each Web site, examining many aspects of both of these sites to see how true they are to the guidelines. The main lesson for you to learn here is how to determine why these are excellent Web sites. After you can recognize what goes into a successful Web site, you can build your own top-notch site.

I strongly encourage you to browse the Web sites on the CD-ROM. The grayscale screenshots in this chapter do not do justice to the Web author's skill in using color within each page to communicate or entertain. Before you get to the CD-ROM Web sites, you will run across an introductory page. This page contains hyperlinks to the Web pages I used for each of the screenshots in this chapter. You might want to view the actual Web pages before you read this chapter or read this chapter with your computer nearby. As you browse the sites on the CD-ROM, any time something catches your eye, make sure to view the HTML source. In fact, if you want to forget about reading the rest of this chapter, and just dive into the CD-ROM sites, I won't be offended. Both sites are so interesting, you might need to view them twice, once to read the content and a second time to review the design. If, at the end of your tour, you know what makes each site superb, you've learned all I've wanted to show you. If you'd rather undertake a more structured review of each site, then just read on.

Examining a Corporate Internet Site

The Internet was once the domain of universities, government, and the military, but it is now the next market in the business world and businesses are flocking to it in unprecedented numbers. Some businesses are on the WWW because it's cool. Some see it as another means of advertising, like TV or radio. Some see it as a library, a place to deposit information about themselves that anyone can read. Some businesses have more vision and see it as a unique means of tying together themselves, their vendors, and their clients for improved competitiveness and service. And, of course, for many, their business is the Internet.

For all of the tens or even hundreds of thousands of businesses on the Internet, I had a very difficult time finding business-oriented Web sites that made good use of all of the design principles presented in the last three chapters—namely, an attractive and useful interface, good use of page layout, and easy to use navigation systems. I am very puzzled by this because most businesses are very savvy about their other media advertising. They will spend top dollar on materials and talent to put out ads, brochures, posters, and billboards displaying their products or services. Yet when these same companies enter the Internet world, the company logo, a picture of the company president, and a few pages of text are somehow supposed to be enough. Obviously, many businesses have not yet adapted to this medium.

I did finally find a company that understood the Internet audience, how to use Web technology, and how to adapt its business to it. This company, Rosenbluth International, built the Travelution Web site, `http://www.rosenbluth.com`. Travelution is basically your very own online travel agent. All of the resources and information that you need to plan a personal or business trip, whether domestic or international, are available from this site. And, the company used a variety of approaches to optimize this information for use over the Internet and add some services unique to the Internet. Rather than just trying to reproduce its paper, radio, or TV ads, the company has used the design principles discussed in the earlier chapters to good effect.

Before I discuss how these principles were used, I'll first give you some background from Susan Steinbrink, director of Consumer and Strategic Marketing at Rosenbluth International, on the factors that influenced the building of Travelution:

> Rosenbluth was a pioneer in the travel industry as it specifically relates to electronic commerce. In 1988, Rosenbluth Vacations established its presence on Prodigy. In May, 1995, Rosenbluth's site was launched on the Internet. This past November, the site was enhanced by Rosenbluth International. The core competency behind the site was twofold—increase brand awareness and seek a new audience (from corporate travel to the general traveling consumer). Through the Internet, we hope for Rosenbluth International to become the trusted advocate of travel information for Internet users, as we have for millions of corporate travelers worldwide. Rosenbluth International's Web site offers the general traveling consumer a fresh alternative to value-added travel data and services, including the ability to book air, car, hotel, cruise, and tour reservations, and browse specific destination information.

Building Travelution was a team effort that included not only the Web site builders, but also many key personnel at Rosenbluth International:

> Our Web site was developed by a third party Web development company who worked closely with our IS and marketing group to ensure that the design met our expectations. Even Hal Rosenbluth, the president and CEO of our company, was involved in how the site looked. He had final approval on all design aspects prior to the launch.

Even though Rosenbluth was an early pioneer in electronic business, the company still took seriously the updating of its Internet site:

> We spent a good deal of time ensuring that we had a good handle on exactly what we wanted to end up with. In order to accomplish this, we developed detailed Web site specifications and detailed flowcharts, offering up every page that we wanted to create. Then, the work began.

Even though Internet technology is constantly changing, the team kept things in perspective:

> Our goals for our Web site are constantly being refined. We, of course, took into consideration new technologies, versions of browsers, and the like. It was important for us to be cognizant of the lowest common denominator with all of our decisions.
>
> When designing our site, great measures were taken to create a Web site that would not be voluminous, but rather informative and fun. Easy navigation was a priority during the design, as well as the moderate use of graphics to ensure speed. The text on the site is light, as well, to insure that the information is not overwhelming to visitors. We know that travel is stressful enough; we wanted our Web site to help travelers "de-stress" their trip by providing only the information that would be the most useful and easy to use.

According to Steinbrink, here are some of the future directions Travelution will be exploring:

> Recently, we integrated Rosenbluth International's patented E-Res Electronic Reservation System, which allows Rosenbluth International corporate clients the opportunity to utilize the Internet as a means to book their travel plans online by incorporating negotiated fares and travel policy. Currently, there is a fear in the business travel industry of travelers booking online, if they use a system that does not incorporate travel policy.
>
> We will also explore push marketing through the Web site in the future. For example, if a traveler is in the cruise and tour section and requests specific information about a supplier, we will be able to push information to them via the telephone lines.

Interface Design at Travelution

Before proceeding, note that it is best to view Travelution with Netscape Navigator, version 3 or better. Although the site displays quite nicely in Internet Explorer, some special effects added using JavaScript will not work with IE, version 3 or earlier.

A good interface

- Is consistent
- Is easy to use
- Is colorful or beautiful and nonirritating
- Can be seen on a variety of monitors with different resolutions and color depths
- Has well placed and appropriately sized controls

Travelution receives an A+ for consistency. Its home page, shown in Figure 26.1, is a good example of the consistency used through the whole site.

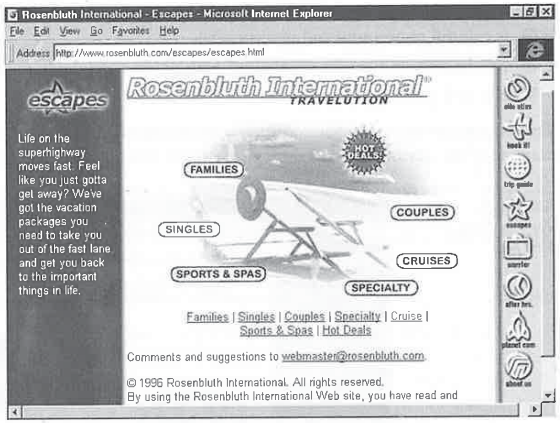
FIGURE 26.1. *The Travelution home page.*

A background image is used to set the color tone throughout the majority of the site. The left portion of the page is a textured blue; the larger, central portion is white; and the right side is a neutral yellow. Both the blue and yellow portions make good use of shadows to add some attractive and simple 3D effects. The colors visually organize the information for you. The right side is for placement of the main navigation menu. The middle contains the bulk of the information you will read or respond to. The left side is for secondary navigation menus or miscellaneous information. This design is used on almost every page of the site, even forms.

Most readers will understand this simple and easy-to-use interface completely after traveling through only two or three screens. The strength of this design is in its simplicity and good use of color. Notice how any elements placed over any of the three color regions have good contrast with that region, which means good visibility with a variety of monitor settings and color depths. As if this layout isn't enough, further graphical hints as to your location are given

throughout the site. For instance, if you click on Escapes from the home page, you will see the main screen for the Escapes section as shown in Figure 26.2.

FIGURE 26.2.
The main page for the Escapes section of Travelution.



An attractive graphic, using the word "escapes," is placed in the upper-left blue portion of every page in this section. The star graphic that represents the Escapes menu item in the main navigation menu to the right is included in the graphic as an additional visual cue. Additionally, if you explore the Escapes section further, the Escapes graphic becomes a hyperlink back to the main Escapes page. More location hints are also incorporated into the company logo. You can see how this is effectively used by going back to the home page, clicking Book It, and traveling to the Flights screen shown in Figure 26.3.

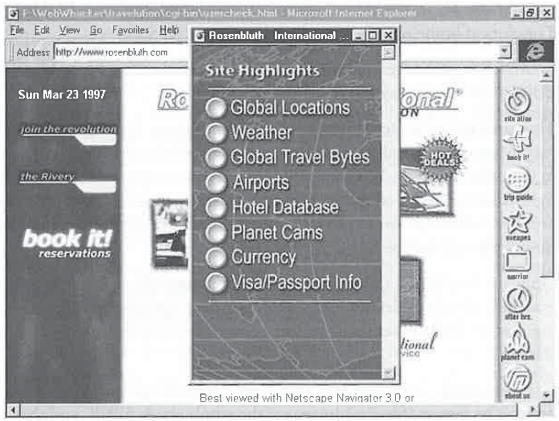
Note the word "Flights" has now been added to the company logo at the top of the page. This has the added advantage of keeping the user's visual attention on your company logo. Because the logo is present on every page, it might tend to fade out of the visual attention of most visitors after awhile. Because it's an active part of the site with these location hints, however, visitors will automatically check the company logo on every page. This is an excellent use of advertising technique in the Internet medium.

As an aside, also notice how the page layout shown in Figure 26.3 no longer has the blue-white-yellow background pattern nor the navigation menus seen on the rest of the site. This is another visual cue to visitors that they are about to leave the site and enter the reservation computers.

FIGURE 26.3.
Adding site location hints to the company logo.

One interesting use of the browser interface is spawning off a new browser window (via JavaScript) and resizing it to display additional navigational information not directly available from the page. For instance, from the Travelution home page, if you click Site Highlights, a new browser window is opened and filled with an HTML page as shown in Figure 26.4.

FIGURE 26.4.
Spawning a new, customized browser window.



Notice how the new browser window has no controls or address bar and has a specific window size. As taken from the Travelution home page, the JavaScript code for spawning the new window is

```
function WinOpen() {
 /* Turn status, toolbar, directories, etc. off by replacing
    yes with no. Modify the HEIGHT and WIDTH calls to suit
➥your needs for this window. */
  var msg = window.open("../global.html","DisplayWindow","HEIGHT=360,WIDTH=230,
➥status=no,toolbar=no,scrollbars=yes,directories=no,menubar=no,location=no");
  if (navigator.appName == "Netscape" )    {
  msg.creator = self;
  }
```

The hyperlink code for initiating the spawn is

```
<A HREF="usercheck.html" onClick="WinOpen();"><IMG SRC="../images/main_03b.gif"
➥WIDTH="250" HEIGHT="96" BORDER="0"></A>
```

This new window disappears after a hyperlink within it is selected.

The vast majority of controls used at Travelution are related to navigation and are discussed in a later section of this chapter, "User Navigation at Travelution." The only other interface controls are typical HTML form buttons for submitting or clearing forms.

Page Layout Techniques Used at Travelution

As you might have guessed by now, tables are used at Travelution for most page layout. The home page uses a container table three columns wide to lay out the contents of the left and right portions of the page. A nested table is placed into the center column and contains the content shown in the white section of the page. This nested table effect is shown on the Travelution home page in more detail in Figure 26.5.

In this figure, I changed the table border width for the container table to two and for the nested table to five for display purposes.

This same layout is used throughout much of the site, making maintenance and changes quite easy. Most changes will be made to the content of the nested table. Any content can be added to the nested table without disturbing the layout or appearance of the content of the left and right columns in the container table. Many times the content of the central column does not need the additional organization of a nested table, and the content of the central column is simply placed into the container table.

Only one simple font face, Arial, is used throughout the site to improve page appearance. This gracefully decays to default fonts if the user does not have Arial on his or her machine, ensuring that no information will be lost to any user and that the basic flow of text will not be affected.

Graphics are for the most part small and fast-loading, leading to a quick response for most pages. The largest (file size) image on the whole site is 59KB, and that is a weather map over which Travelution has no control. The Web site builders followed the general rule of thumb of keeping most pages under 50KB in size. Graphics are used well to inform or entice the

visitor. The enticement part is especially important for a travel-oriented site. It is much easier to fantasize about running down a white sand beach if you can see a picture of it. In fact, it would be quite easy to try to overload Travelution with too many images or images that are too large. The builders struck a good compromise between the need to display pictures of travel locations and the limited Internet bandwidth available to many visitors.

FIGURE 26.5.
Using nested tables to lay out Travelution.

Interestingly, only one animated GIF and no video is used on the entire site. The animated GIF is the centerpiece of the opening page; it does not compete with any text. Again, in a business situation, having a potential customer leave the site because he or she is irritated by animated GIFs is unacceptable. In my opinion, Travelution does not suffer any ill effects in appearance or usefulness by not having more animated GIFs or video. There is one short audio clip with a warm welcome from the Rosenbluth CEO. You can find this on the About Rosenbluth part of the site.

Neither style sheets nor frames are used at Travelution. This ensures compatibility with the maximum number of browser types. This compatibility is essential for a business site because any browser not served means potential lost business and revenue. Travelution also does not use spacer images for pushing text or graphics around a page. This keeps page building simple, and means more time can be spent placing content onto the site.

There is only one small layout flaw at Travelution, and I only point it out because it can become a major problem in certain situations. You will need to view the file `tip_int.html` in the tips folder on your screen to actually find this flaw; it will not show up in a screenshot. If you look on the left blue side, below the navigation text, you will see a text hyperlink about a Big

Mac. The problem here is that the blue hyperlink color blends in with the blue page background color. On many monitors, you might not even notice this hyperlink. In this case the hyperlink is more entertainment-oriented and does not detract from the information on Travelution, so little is lost. But what if the hyperlink were more important? Whenever you think about the colors to use for hyperlinks or text, think about what might happen if your text shifts around in a browser. Will that text or hyperlinked text end up over a background element that will camouflage it? If you think your text or hyperlinked text will end up camouflaged, you might consider replacing them with graphics (for Travelution, a piece of hamburger clip art and some graphical text would do nicely) or changing the default text or hyperlink colors. Another possibility is using nested tables to ensure that a particular background color is always associated with certain text or text hyperlinks.

User Navigation at Travelution

I don't know any other way to say it except that the navigation system at Travelution is simply one of the best I've seen on the WWW. Every single navigation system you read about in Chapter 25, "User Navigation," is attractively represented, consistent, and well placed on most of the pages of the Travelution site.

Take another look at the main navigation menu that travels down the yellow, right side of most Travelution pages. (Refer back to Figure 26.1.) First, all choices on this menu fit comfortably in a 640×480 screen. Next, the navigational choices are represented by both text and interesting icons. The `ALT` field in `<IMG SRC>` is also filled in appropriately in case a surfer has images turned off or is using a text-based browser. The menu choices are arranged in a descending order, from most likely to be used to least likely. As if that isn't enough for a navigation menu, a special effect using JavaScript is built in for users of Netscape 3 or later. On the `mouseover()` event, the button appears to shrink as if it has been pushed. This improves the interactivity of this menu while providing some simple entertainment.

Can't find what you want from the main navigation menu? Go straight to the first choice in the navigation menu, the site atlas, which is shown in Figure 26.6.

This site map makes excellent use of graphics and text to outline the site. A user is able to take everything in quickly and clearly and click to a destination in short order. The icons and most of the main headers match those found in the main navigation menu along the left side. Other nice touches include a "You are here" note and inclusion of important non-Travelution sites, including After Hours and the Currency converter under Trip Guide.

Clicking on one of the major sections, in this instance Escapes, takes you to the Escapes opening page, shown earlier in Figure 26.2.

Here, an attractive image map is backed up by text hyperlinks that will take you to any of the listed escapes. The Escapes graphic is in the upper left as a further indicator of your location, as noted earlier. The main navigation menu is always to your right if you change your mind. Clicking on one of the Escapes, such as Cruises, takes you to the main menu for cruises shown in Figure 26.7.

FIGURE 26.6.
The site atlas at Travelution.

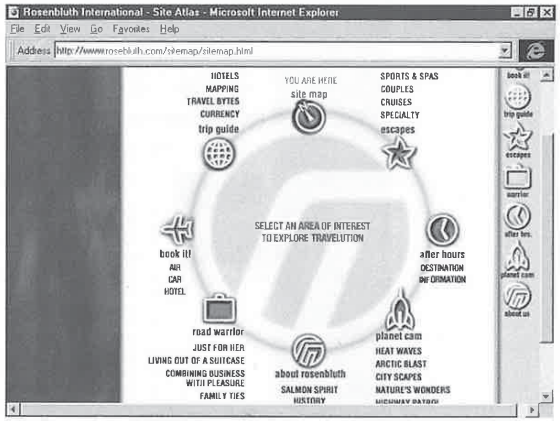
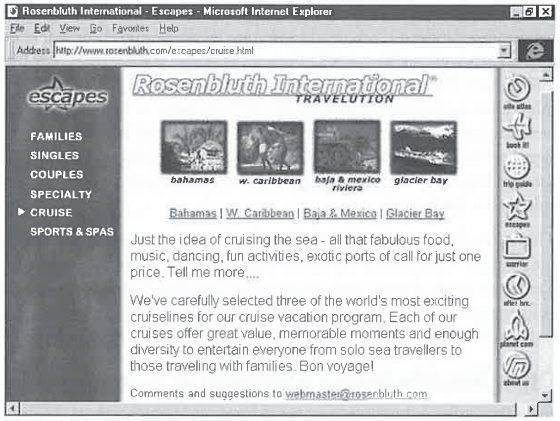


FIGURE 26.7.
Choosing a cruise in the Escapes menu at Travelution.



Notice how the navigation menu to the left has changed. Now you have graphical text, with the text highlighted (in this case, yellow) for the Escapes subsection you are in. Each of the choices on the left represents one part of the image map from the Escapes page. This is included below the Escapes graphic as further emphasis that you are in a subsection of the Escapes section of Travelution. In the middle of the page you can choose from any of the four cruise packages. If you click on the Glacier Bay cruise, you will see the screen shown in Figure 26.8.

FIGURE 26.8.
One of the cruises offered at Travelution.

Notice here that none of the navigation menus changed even though you've gone another level deeper into the site. Of course, the question is, where would you put another navigation menu, or is it even appropriate here? Do you now change the left side navigation menu to reflect the four cruises, or leave it as is to provide a means of visiting other Escapes? Because there are no more levels to drill down through (an HTML form to request more price information is the next and last level from this page), an additional menu is optional here. There is no right answer here in my opinion. This is a judgment call and should be based on how you think most users will want to navigate through this section (feedback or user testing would provide invaluable information here).

This style of navigation is used through most of Travelution and makes it almost impossible for even the most novice surfer to become lost. The vast amount of travel information on Travelution is well organized using these many navigation menus, emphasizing the point that your next trip to Glacier Bay will be as well planned and organized by Rosenbluth International as your trip around the Travelution Web site has been.

Other Notable Design Highlights at Travelution

In addition to good page layout, navigation, and interface design, I was drawn to Travelution because the designers sprinkled it all with a bit of well-placed fun. If you go to the Road Warrior section, for example, you will see a picture of a cow with the phrase underneath "Feeling stressed?" (See Figure 26.9.)

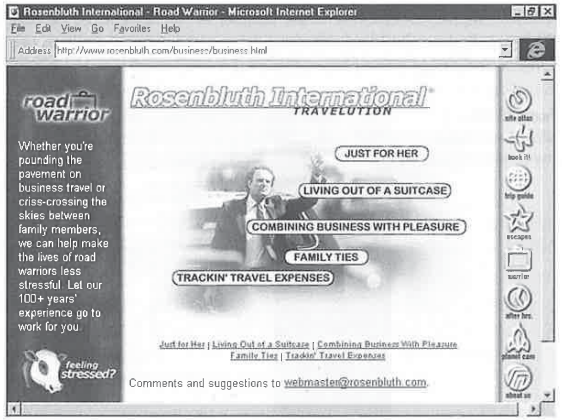


FIGURE 26.9.
A cow on a travel site?

You don't really expect to see a cow on a travel site, so it is immediately eye catching. Clicking on the cow brings up a list (again, using the spawned browser window technique mentioned earlier) of 10 reasons to visit a resort ranch. Any road warrior will empathize with at least one of the reasons and be encouraged to investigate this particular travel package by clicking a button at the bottom of the list. Although this is simply good advertising technique, it is executed well within a Web site and adds to the enjoyment of Travelution.

Another excellent blending of Web technology with the travel theme of Travelution is an entire section called Planetcam. Here you can find lists of hyperlinks to various Webcams around the Internet that are pointing at items of interest to the traveler. One of these lists, Heat Waves, is shown in Figure 26.10.

These hyperlinks will load the latest picture for wherever the Web camera is pointed to give you immediate feedback on what's happening. Going to Maui tomorrow? Don't rely solely on a weather map or atlas. See for yourself by checking out the Maui Web cameras. Where else can you get this type of immediate feedback, day or night, from the comfort of your computer chair? Travelution did not skimp on its selection of locations to choose from, either, with a total of 60 cameras from around the world. This is an excellent example of putting the unique features of the Internet (Web cams everywhere) to work for your business. It also shows the Internet savvy at Rosenbluth International.

Rosenbluth International has effectively demonstrated that a Web site can be attractive, well designed, and even a little fun without sacrificing any business usefulness. Building a site like this takes a commitment in time and resources, but if a business is serious about its presence on

the WWW, why should it settle for a mediocre site? A quality business should have a quality Web site. You can build an effective business presence on the Web using mostly HTML and some sound design principles, as you've seen well demonstrated at the Travelution site.

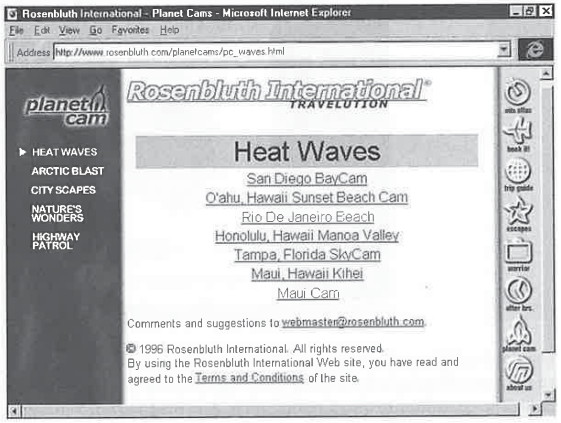


FIGURE 26.10.
Web camera sites of interest to travelers.

Examining an E-zine

Before I review the CyberMad Web site, you need to remember that this is an entertainment-oriented Web site. Although content clarity and ease of navigation are still important, there is much more room for a broader interpretation of the guidelines. On a business site, a word that is not understood or a user who becomes lost can cost the business many dollars—and that is completely unacceptable for a business site. For an e-zine, missing a word or sacrificing some clarity for art might actually contribute to the entertainment value of a particular page. Remember, your design guidelines must be tightly integrated with the flow of your Web site content. Design techniques used on a corporate site can devalue the content (that is, lead to terminal boredom) on an entertainment site, and entertainment techniques can lead to misunderstandings on a business site (that is, lost dollars and upset clients). A particular approach to designing any Web site is successful because the content and the design work together in synergy. Either one alone is insufficient to achieve greatness for a Web site.

Because Web site design and content are so tightly interwoven, in order to effectively study CyberMad, we first need to hear from Christopher Parr, Creator and Executive Producer, as to why he created the site:

> CyberMad evolved due to the lack of quality sites. Either they had no content or the graphics were pulled straight out of the clip art books. Being a writer with experience in theater and TV, I constructed the CyberMad project as a similar medium—kind of like "Saturday Night Live" meets *Alice in Wonderland* online. Since then, the style and influence has been repeated in what has been dubbed as an *e-zine*. I never once considered it as your standard static Web site. CyberMad morphed into a dynamically changing and inviting presence, showcasing something unique and strange without a corporate presence.

The majority of CyberMad was written solely by Christopher Parr with some CGI scripts from a UNIX guru, and later, inclusion of the works of various artists. Parr says that after he had decided on his goals, his Web site started this way:

> CyberMad is a constantly evolving presence. At the beginning, the project was fairly thin, but after a few weeks of conceptualization, I constructed a solid skeleton—designed for growth and expansion. Currently, the site consists of many layers—getting deeper and deeper every month.

Concerning the problem of designing Web pages in relation to constantly changing Web technology, Parr says this:

> It is difficult to tolerate old browsers, but you need to think of the audience. If one person with Netscape 1.0 is turned away, that's one person who will never experience your presence. Web technology grows too fast and doesn't coexist with the reality that most people are still connected with 14.4 modems.

As your Web site evolves, Christopher Parr recommends, "Watch the trends, but don't always follow everyone's advice. Stay true to your identity and get online."

Interface Design at CyberMad

With these site goals in mind, let's look at how some of the interface controls at CyberMad are designed and placed to meet these goals. The interface controls that are primarily navigational in flavor are discussed later in this chapter. Our first stop is the Hollywood Geek Machine, the first screen of which is shown in Figure 26.11.

As you can see, there are only two obvious controls, Push and Exit, both part of a generic machine image. Because most people come here out of curiosity, the Push button is the most obvious next choice. When Push is clicked, a page with a JavaScript random number generator is loaded and a random "geek" image is displayed, as shown in Figure 26.12.

In this second screen, the leftmost button now reads Again rather than Push. Clicking on Again takes you back to the first screen and you can loop through these two screens until you've seen all the geeks. Then you can exit back to the Culture Department page.

FIGURE 26.11. *Screen one of the Hollywood Geek Machine.*

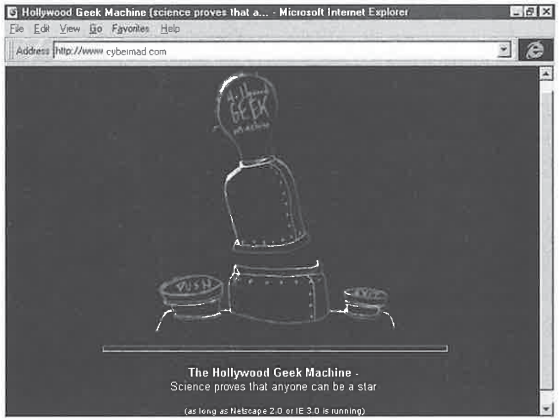
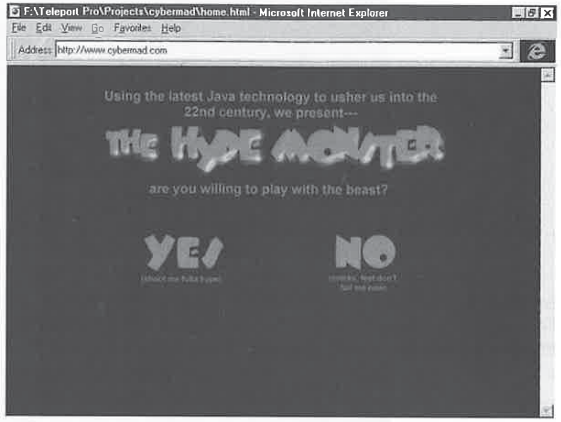


FIGURE 26.12. *Screen two of the Hollywood Geek Machine.*

From an interface design perspective, this is perfectly done. The controls are obvious, relatively easy to read (acceptable for an entertainment site), cleverly incorporated into a machine-style image (maintaining the machine context of this area of the site), and consistent in style and color with the rest of the page. There is no clutter on this page to distract you from the machine, so it will clearly be the focus of attention. The color scheme is simple (red, black, and white), so it should be readable on almost any monitor at any resolution. The entertainment value of this section is enhanced by the excellent use of these controls, making the experience of the Hollywood Geek Machine that much more enjoyable.

Our next stop as we examine interface design at CyberMad is the Hype Monster, screen one of which is shown in Figure 26.13.

FIGURE 26.13. *Screen one of The Hype Monster.*



Again, this mostly black and red interface has only two obvious choices, Yes and No responses to the single question posed on the page. Again, the controls are easy to find, entertaining in themselves, and well integrated with the page theme and style. The small point text under the responses might be a bit hard for some readers to see, but it is not essential to operating the controls. True to form, clicking on the Yes response brings up the next page, shown in Figure 26.14, which contains a Java applet scrolling through a number of hype phrases.

The single interface control on this page is a skillful combination of a color reversed image and some text. Again, even though this control is not quite the same style as the Yes and No responses on screen one (the font used in the text is the same) of The Hype Monster, it is immediately apparent that it is a control because little else is on this page. The control is again well integrated with and adds beauty to the page. When you've seen most of the hypes, you assume you can return by clicking this control. Surprise! This control takes you down another path and you come to the screen shown in Figure 26.15.

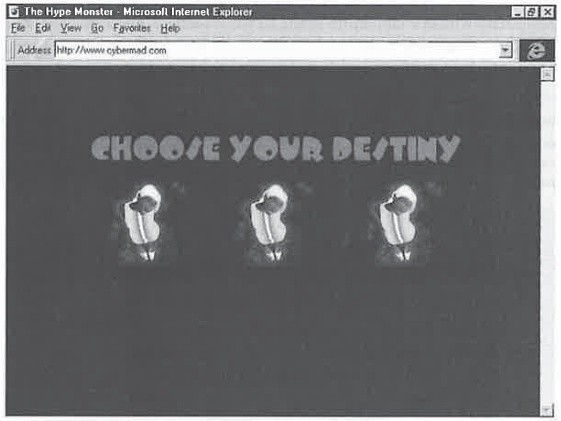
FIGURE 26.14.
Screen two of The Hype Monster.

FIGURE 26.15.
The Hype Monster won't let go.

Where are you now? Obviously, you can use your browser Back button to back out of all of this, but this is one instance where the mystery of The Hype Monster is enhanced by offering no alternative controls or navigation. This heightens your curiosity because you thought you had seen all The Hype Monster had to offer when you were watching the hype phrases go by in the last screen. This is an excellent example of taking an interface no-no, offering only one poorly marked path, and making it work for you. You have little choice but to click the image

and continue forward even though you don't know where you are or where you're going. The final satire for this piece of CyberMad becomes evident from the last screen of The Hype Monster, shown in Figure 26.16.

FIGURE 26.16.
The final message of The Hype Monster.



If your browser is set to display the URL in the status bar, you will notice that you can go to Microsoft, Netscape, or back to the CyberMad Culture page. Again, the interface is consistent with that found in the rest of The Hype Monster. It is interesting to note that on this page, the same image is used to represent all three destinations with no text explanation. Yet the rest of the controls in The Hype Monster were very clearly marked. Why do you think Christopher Parr left the text out? In my opinion, he is again using mystery to emphasize the lessons The Hype Monster has just taught you. Sometimes as much of a message is communicated by the lack of clear interface controls as is communicated using clearly marked controls.

There are many other excellent examples of well-integrated, beautiful, and yet very useful interface controls throughout the CyberMad site. I recommend visiting and examining Movie Central and Insult-a-rama for further examples in the excellent use of interface controls and design. As you surf through the site, notice how each section has its own unique flavor and how the interface is crafted as an integral part of the flavor for that section.

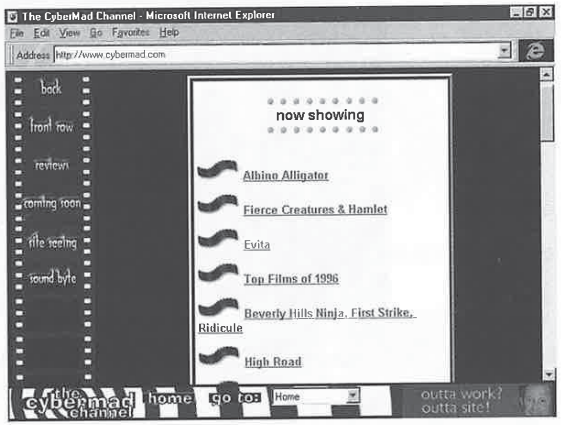
Page Layout Techniques Used at CyberMad

Page layout techniques vary tremendously at CyberMad, as you might expect given the wide range of topics. In this review of layout techniques, I provide the HTML filename under discussion so that you can examine the HTML source on the CD-ROM for more detail on how a particular page was built.

The first screen of the Hollywood Geek Machine (`geek.html`), which you saw in Figure 26.11, simply centers a series of images and builds the machine without having to resort to a table for layout. The second screen (`geeks.html`) continues with this simple layout scheme even with a JavaScript routine inserting an image.

Simplicity within a table rules in the Reviews section (`reviews.html`) of Movie Central. Figure 26.17 shows a fixed width table (300 pixels) with a white cell background centered against a black page background (the left and bottom menus on the page are navigation menus in frames).

FIGURE 26.17.
`reviews.html` uses a single cell table for page layout.



This table is unique in that it is composed of only a single cell and all of the text and graphics are included in this single cell. Here is the HTML code for much of the table:

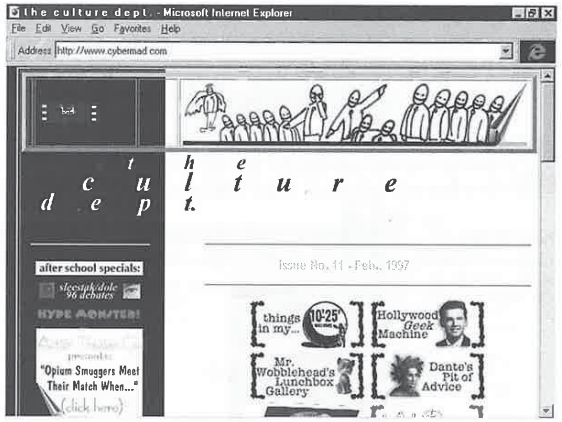
```
<CENTER><TABLE CELLPADDING=3 CELLSPACING=3 BORDER=3 WIDTH=300>
<TR><TD COLSPAN=2 ROWSPAN=2 ALIGN=left VALIGN=MIDDLE BGCOLOR=#FFFFFF>
➥<font face=arial size=3>
<CENTER><br><img border=0  src="flick.gif"  width=150 height=13><br><b>
➥now showing</B><br><img border=0  src="flick.gif"   width=150
➥height=13></CENTER></FONT><P>
<BR><b>
<font size=2 face=arial><b>
<a href="gator.html"><img src="film-1.gif" border=0>Albino Alligator</a><P>
<a href="fierce.html"><img src="film-1.gif" border=0>Fierce Creatures
➥& Hamlet</a><P>
 ---- SNIP ----   THE REMAINING REVIEWS   ---- SNIP ----
</TD></TR>
</TABLE>
```

Notice how the film strip image opposite each review is inserted in the hyperlink anchor with the text, allowing one anchor set to work for both the text and the graphic. This saves a lot of

coding time. Even though it is common practice to include graphics in their own table cell, there are instances, such as this, when good organization can be achieved without the hassle of working with many table cells.

CyberMad also makes good use of nested tables for page layout in The Culture Dept. (`culture.html`). First, the page background (`backcine.gif`) is a simple black and white (4 pixels high by 640 pixels wide) line. The width of the black (182 pixels) and white (458 pixels) sections acts as a simple grid for placing the graphics and text elements. The container table, within which two nested tables are placed, is set at 600 pixels wide to constrain the page size for easy viewing by people at 640×480 monitor resolution. The container table is again composed of a single cell. The first nested table is composed of four cells and contains the back button and the graphic at the top of the page. (See Figure 26.18.)

FIGURE 26.18.
The first nested table in `culture.html`.

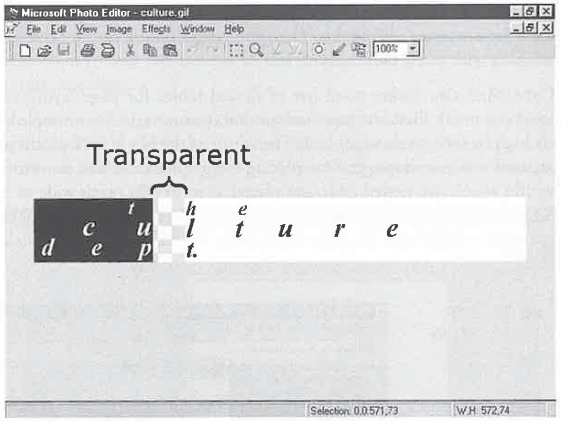


Again I turned on the table borders for improved viewing. The table cell widths and cell justifications are used together to determine whether an element is placed against the black portion or white portion of the page background image.

Below the first nested table, a graphic (`culture.gif`) is simply placed into the container table. If you load this image into a bitmap editor that displays transparency, you will notice that Christopher Parr leaves transparent the portion of the image that crosses the black/white boundary of the page background image. (See Figure 26.19.)

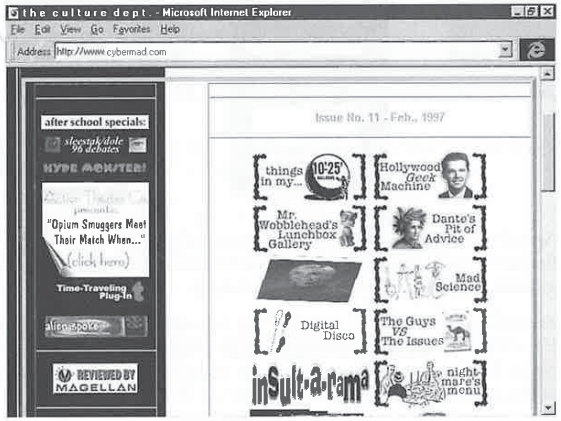
This is a simple way to allow for slightly different browser offsets without worrying about whether the image will stay aligned against the page background.

FIGURE 26.19.
Leaving a transparent hole in `culture.gif`.



The second nested table contains the bulk of the elements seen on The Culture Dept. page. This table is four columns wide but again contains only a single row, as shown in Figure 26.20.

FIGURE 26.20.
Using nested tables in the Culture Dept. at CyberMad.



The first column is used to add an offset (seven pixels) to the left of the table. The second column (`width=128`) contains all of the text and graphics that display against the black portion of

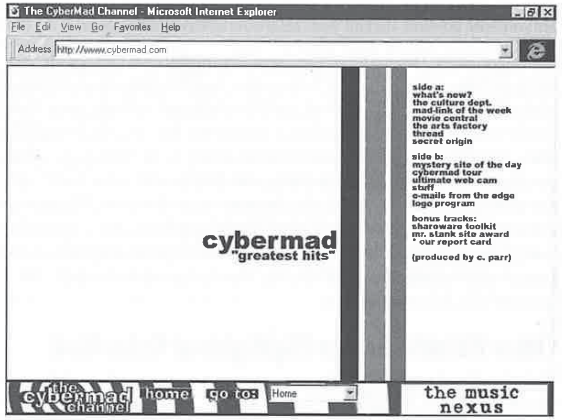
the page background. The third column (`width=70`) spans the black/white transition of the page background. The final column (`width=398`) contains all of the elements that display against the white portion of the background. Again, even though each text and graphics element in `culture.html` is not placed into its own table cell, the page is still well organized.

As Christopher Parr has demonstrated here, you should use the simplest page layout design possible. This will save you time and give you an attractive page. You've also seen how the page layout is tightly integrated with the content of the page and how it can enhance the content. Also notice that no long, run-on pages are found on CyberMad. Even at lower monitor resolutions, you seldom have to page down more than three times to read an entire page. This keeps the reader constantly interacting with the material, which is a primary goal for an entertainment site.

User Navigation of CyberMad

CyberMad uses a drop-down list box in a frame at the bottom of the browser window to make all of the major CyberMad site sections available to visitors at all times. The Table of Contents page, including the bottom frame, for CyberMad is shown in Figure 26.21.

FIGURE 26.21.
The two main navigation menus at CyberMad.



Christopher Parr used JavaScript to build the drop-down list box navigation menu in the bottom frame. This makes the control available to the vast majority of browsers. This also makes excellent use of the GUI paradigm, building on most people's familiarity with a windowing environment. The JavaScript code can be seen in `nav.html`. Note that the original JavaScript included the complete URL for the page to be accessed. I had to change this to include only the filename so that the site could be placed on the CD-ROM. A new page is loaded simply by

selecting an item from the drop-down list. Also notice the "Home" graphic that returns the user to the Table of Contents page (shown in Figure 26.21).

The Table of Contents page emphasizes the navigation menu in the upper frame through good use of white space and the unique placement of the vertical stripes down the page. Anyone using any monitor resolution or color depth will be able to read and find this menu. The menu itself is an image map, and a page layout table is used to align the image map against the background stripes.

Notice how there are no backup text hyperlinks for either menu. Although an informational or business-oriented Web site would have back up text hyperlinks to ensure navigability by all browsers (especially when each browser is a potential client), on an entertainment site, they are again optional. Because so much of the CyberMad content is based on graphics (think back on how the Hollywood Geek Machine would look without graphics), there is no need to design for browsers that can't handle graphics. Even though Christopher earlier stated his commitment to maximizing browser compatibility, the entertainment goals of CyberMad simply cannot be met using text. For instance, the Culture page shown in Figure 26.20 is almost completely dependent on graphics for navigation.

The site author does, however, make good use of the `ALT` attribute for labeling images. Again, let your site content dictate your selection of navigation tools.

Throughout the remainder of the site, Christopher Parr uses a wide variety of text and graphical hyperlinks to keep the user from getting lost. He adds another frame to Movie Central. He uses a slide show paradigm (Back and Next buttons) in many parts of the CyberMad Tour. As you browse the site, also notice how Christopher Parr uses the hyperlinked word "Back" rather than "Home" as his primary means of returning to an index page. This builds on the browser paradigm for navigation because everyone knows where the Back key is on a browser within five minutes of learning how to use a browser for the first time. This makes navigating CyberMad even more intuitive. The hyperlinked "Back" graphic is also not placed in the same location on every page, nor does it look the same. This shows good integration of navigation with the design of each individual page or section. Very few pages in this entire site required the use of the browser's Back button to leave.

Other Notable Design Highlights at CyberMad

CyberMad makes good use of an entrance or cover page to introduce you to the site. This page, shown in Figure 26.22, immediately sets the irreverent tone for the site and invites further exploration.

Animation and video are sparsely but effectively used throughout the site. The Culture Page has a scrolling marquee of the e-zine volume and date for browsers that can view it. Some of the animated GIFs include Mr. Wobblehead (how could you not animate this?), several other unintrusive animations on the Culture Page, and a marquee of lights on many pages in Movie Central. My favorite animation was the rotating nut used as the "o" on the Arts Factory page.

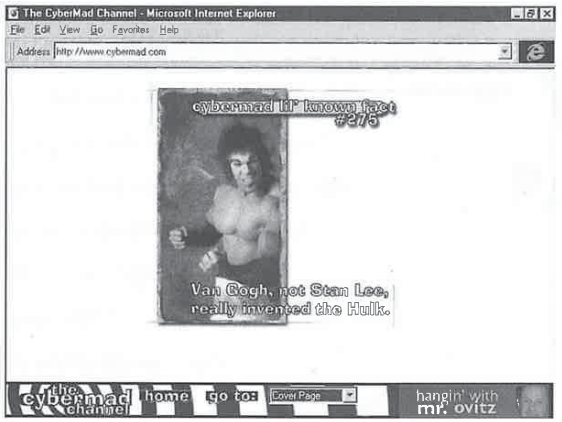


FIGURE 26.22. *The entrance to CyberMad.*

Parr also uses Java applets for The Hype Monster and in `guyindex.html`. There are also a number of sounds (as WAV files) sprinkled through the site. The only video is a very short AVI file connected via the `DYNSRC` attribute in the `<IMG SRC>` tag to a hyperlinked "Back" graphic on many pages. The only overly annoying movement I found on the whole site was the ubiquitous ads in the lower navigation frame.

Even with some of the pages shouting with oversaturated colors, CyberMad is still one of the best examples I have seen of the use of standard HTML to match content with Web page design. The many different page designs and techniques throughout the site were always matched to their content, enhancing the entertainment experience at CyberMad and, ultimately, making a unique contribution to the new media that is the WWW. The medium, of the WWW is not paper, it is not TV or the movies, and it is not radio. The WWW is a unique medium, and Christopher Parr has shown us one way to master it.

Summary

The lessons in this chapter are best summarized by a few more words of wisdom from Christopher Parr:

> Reaction was initially positive, with the site being selected for the standard recognition with "Cool Site of the Day" and "USA's Hot Site." Then, for about five months, nothing happened. Visitors were still coming, but it seemed as if there was no buzz about the site in the press. I actually considered canceling the site. Then, all of a sudden, CyberMad won the 1996 NetBest Award (`http://www.ypn.com/netbest`) and

I was in Boston being introduced by author Douglas Coupland. After that point, everything escalated. I was catching reviews and articles about the site everywhere I turned. *Tenacity* and *dedication* are two words to remember, and don't expect overnight success.

IN THIS PART

V PART

Associated Technologies and Programming Languages

CHAPTER 27

CGI Programming

by Eric Herrmann

IN THIS CHAPTER

HTML Unleashed has everything you need to know about HTML—so what does CGI have to do with HTML? Not only will you learn what CGI has to do with HTML in this chapter, you will also learn how to use CGI to create interactive Web pages and get the detail you need for later reference. In the next few pages you will learn enough to build your own interactive Web pages, and gain an understanding of why CGI is extremely important to anyone building HTML Web pages. It's important, that is, if you want your Web page to sell products, interact with your Web page visitor, or provide any type of dynamic Web experience on your Web site. Finally, this chapter provides the details of CGI protocols of the CGI environment. If your only resource for CGI programming is *HTML Unleashed*, you will be able to refer to this chapter again and again to get the details about HTTP headers, environment variables, and decoding data from the all the interactive forms on your Web site.

What CGI Programs Do on Your Web Site

CGI programs act as a special interface between your Web page and your Web server. CGI stands for Common Gateway Interface, and the name describes exactly what CGI programs are doing on your Web site. A CGI program receives the data from your Web page. Usually, a Web page uses an HTML `<FORM>` tag to initiate the CGI program. The HTML `<FORM>` tag has become the method of choice for sending data across the Net because of the ease of setting up a user interface using the HTML `<FORM>` and `<INPUT>` tags. Your CGI program interprets the data passed to it. It can then manipulate that data and return another Web page, or it can pass that data to another program, acting as an interface between the Web and other programming environments.

Putting databases, catalogs, and company information on the Web has become big business for lots of Web designers in the 1990s. The database and other programs don't typically interact directly with the Web; they use a CGI program to read the incoming data and another CGI program to return the data in HTML format. The terms *gateway* and *interface* come from this common use of CGI programs. The programs act as a gateway by reading incoming data from your HTML pages, and again as a gateway when they send new HTML pages back to your Web client. They also act as an interface when they send incoming data or receive outgoing data from other more complex programs, such as database servers.

HTML and CGI

HTML, the language that gathers the data for CGI programs, is primarily designed for formatting text. So, how does it get involved with your CGI program? The primary method is through the HTML `<FORM>` tags. However, it's not required that your CGI program be called through an HTML `<FORM>` tag; your CGI program can be invoked through a simple hypertext link using the anchor tag, perhaps like so:

```
<a href="A CGI program"> Some text </a>
```

The CGI program in this hypertext reference or link would be called, or activated, in a manner similar to being called from an HTML `<FORM>` tag.

You can even use a link to pass extra data to your CGI program. All you have to do is add more information after the CGI program name. This information is usually referred to as extra path information, but it can be any type of data that might help identify to your CGI program what it needs to do.

The extra path information is provided to your CGI program in a variable called `PATH_INFO`. `PATH_INFO` is any data after the CGI program name and before the first question mark (?) in the `href` string. If you include a question mark after the CGI program name and then include more data after the question mark, the data would go in an environment variable called the `QUERY_STRING`. The environment variables available to your CGI program are covered later in this chapter. So to put this all into an example, if you created a link to your CGI program that looked like

```
<a href=www.practical-inet.com/cgibook/chap1/program.cgi/
extra-path-info?test=test-number-1> A CGI Program </a>
```

then when you selected the link A CGI Program, the CGI program named `program.cgi` would be activated, the environment variable `PATH_INFO` would be set to `extra-path-info`, and the `QUERY_STRING` environment variable would be set to `Test=Test-number-1`.

Normally, this is not considered a very good way to send data to your CGI program. For the programmer, it's harder to modify data hardcoded into an HTML file because it can't be easily done on the fly, and it's easier to modify for the Web page visitor who is a hacker. Your Web page visitors can download the Web page onto their own computers, modify the data your program is expecting, then use the modified file to call your CGI program. Neither of these scenarios seems very pleasant. And lots of other people felt so also, so this is where the HTML `<FORM>` tag steps in.

The HTML `<FORM>` tag is responsible for sending dynamic data to your CGI program. The basics just outlined are still the same. Data gets passed to the server for use by your CGI program, but the way you build your HTML `<FORM>` tag defines how that data will be sent. And your browser does most of the data formatting for you.

However, the most important feature of the HTML `<FORM>` tag is that the data can change based on user input. This is what makes the HTML `<FORM>` tag so powerful. Your Web page client can send you letters, fill out registration forms, use clickable buttons and pull-down menus to select merchandise, or fill out a survey.

To sum up, HTML, and in particular the HTML `<FORM>` tag, is responsible for gathering data and sending it to your CGI program. HTTP headers are covered in more detail later in this chapter.

The next section explains how the data collected by your browser is formatted and sent to the Web server. After you know how the data collected by your Web page is formatted, you can use a CGI program to decode the data when it gets to the server.

What Happens to the Data Entered on Your Web Page

The GET method is the default method for sending data gathered by the <FORM> tag to the server. The URL-encoded string passed to your server is limited by the input buffer size of your server, which can be as small as 256 bytes. This means that the URL-encoded string can get too big and lose data. That's bad.

The data entered on your form is URL-encoded into name/value pairs and appended after any path information to the end of the URL identified in the ACTION field of your opening <FORM> tag.

Name/value pairs are the basis for sending the data entered on your Web page form to your CGI program on the server. These pairs are covered next in detail. The browser takes the following steps to get your data ready for sending to the server:

1. The browser takes the data from each of the TEXT entry fields and separates them into name/value pairs.
2. The browser encodes your data. URL encoding is covered last in this section.
3. After the data is URL encoded, the data is appended to the end of the URL identified in the ACTION field of your FORM statement. A question mark is used to separate the URL and its path information.

The data after the question mark is referred to as the query string.

Whether you use the GET method or not, the URL encoding of the query string is consistent for all data passed across the Internet. QUERY_STRING, by the way, is one of the environment variables that is discussed later in this chapter.

The HTML used to create the form is shown in Listing 27.1, and the form itself is shown in Figure 27.1. Listing 27.2 contains the data from a registration form. You can see the name/value pairs separated by the ampersand (&) and identified as pairs with the equal sign (=).

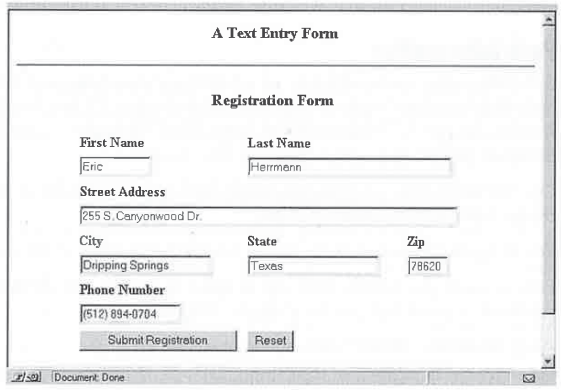
Listing 27.1. The HTML for a registration form.

```
01: <html>
02: <head><title> HTML FORM using Text Entry</title></head>
03: <body>
04: <h1> A FORM using the Get method for text entry </h1>
05:
06: <hr noshade>
07: <center>
09 : <FORM Method=GET Action="/cgi-bin/nph-get_method.cgi">
10: <table border = 0 width=60%>
11: <caption align = top> <H3>Registration Form </H3></caption>
12: <th ALIGN=LEFT> First Name
13: <th ALIGN=LEFT colspan=2 > Last Name <tr>
14:
```



```
15: <td>
16: <input type=text size=10 maxlength=20 name="first" >
17: <td colspan=2>
18: <input type=text size=32 maxlength=40 name="last" > <tr>
19: <th ALIGN=LEFT colspan=3>
20: Street Address <td> <td> <tr>
21:
22: <td colspan=3>
23: <input type=text size=61 maxlength=61 name="street"> <tr>
24: <th ALIGN=LEFT > City
25: <th ALIGN=LEFT > State
26: <th ALIGN=LEFT > Zip <tr>
27: <td> <input type=text size=20 maxlength=30 name="city">
28: <td> <input type=text size=20 maxlength=20 name="state">
29: <td> <input type=text size=5 maxlength=10 name="zip"> <tr>
30:
31: <th ALIGN=LEFT  colspan=3> Phone Number <tr>
32: <td colspan=3> <input type=text size=15 maxlength=15
➥name="phone" value="(999) 999-9999"> <tr>
33: <td width=50%> <input type="submit" name="simple"
➥value=" Submit Registration " >
34: <td width=50%> <input type=reset> <tr>
35: </table>
36: </FORM>
37: </center>
38: <hr noshade>
39: </body>
40: </html>
```


FIGURE 27.1.
The registration form.



Listing 27.2. Registration form data encoded for the server.

```
QUERY_STRING  first=Eric&last=Herrmann&street=255+S.+Canyonwood+Dr.&city=
Dripping+Springs&state=Texas&zip=78620&phone=%28512%29+894-0704&simple=
+Submit+Registration+
```

Name/Value Pairs

All of the data input from a form is sent to the server or your CGI program as name/value pairs. The preceding registration example used only TEXT input, but even the Submit button is sent as a name/value pair. You can see this at the end of the line of Listing 27.2. The Submit button name is simple and the value is Submit Registration. Notice that case is maintained in the value fields.

Name/value pairs are always passed to the server as *name=value* and each new pair is separated by an ampersand (&) like so: *name1=value1&name2=value2*. This arrangement allows you to perform some simple data decoding and have a *variable = value* already built for your Bourne or C-shell script to use. Using Perl, you can filter out name/value pairs with just a little bit of effort.

Notice on line 16, `<input type=text size=10 maxlength=20 name="first" >`, that the name attribute is added to the input type of text. In programming, the name is the formal parameter declaration, and the value, whether given by default or by entering data into the entry field, is the actual parameter definition.

Put in another way, the name is your program's way of always referring to the incoming data. The name field never changes. The data associated with the name field is in the value portion of the name/value pair. The value field changes with every new submittal. In the sample name/value pair first=Eric, the name is first, and the value is Eric.

Just remember that whether you use text entry, radio buttons, checkboxes, or pull-down menus, everything you enter on your Web page form will be sent as name/value pairs.

Path Information

Path information can be added to the action string identifying your CGI program. You can use path information to give variable information to your CGI program. Let's assume you have several forms that call the same CGI program. The CGI program could access several different databases, depending on which form was submitted.

One way to tell your CGI program which database to access is by including the path to the correct database in the form submittal.

You add path information in the ACTION field of the opening HTML <FORM> tag.

First, as usual, you identify your CGI program by putting into the ACTION field the path to your CGI program and then the program name itself. For example:

```
<FORM METHOD=GET ACTION="/cgi-bin/database.cgi/">
```

Next, you add any additional path information you want to give your CGI program. If you wanted to add path information to one of three databases in the preceding URL, it would look like this:

```
<FORM METHOD=GET ACTION="/cgi-bin/database.cgi/database2/">
```


The path information in this example is database2/.

When the Submit button (in line 33 of Listing 27.1) is clicked, the browser appends a question mark (?) to the ACTION URL; the name/value pairs are appended after the question mark.

URL Encoding

By now you have figured out that in order to send your data from the browser to the server, some type of data encoding must have occurred. This is called URL encoding.

The convention of URL encoding Internet data was started to handle sending URLs by electronic mail. Part of the encoding sequence is for special characters like tabs, spaces, and periods. E-mail tools have problems with these and other special characters in the ASCII character set. The URL gets really confused if you used the reserved HTML characters within a URL. So if the URL you're referencing includes restricted characters like spaces, they must be encoded into the hex equivalent.

URL encoding is important to your CGI program for the following two reasons:

- Several reserved characters must be URL encoded if you include them in the URL string in the ACTION field. Spaces, the percent sign (%), and the question mark (?) are examples of special characters. (More about these characters in the next section.)
- All data is URL encoded, and if you want to be able to decode it when it gets to your CGI program, you need to understand it.

Reserved Characters

So what is this set of characters that can't be included in your URL? One of the simple ones is the space character. If you own a Macintosh computer, you know that spaces in filenames are a common and convenient feature of the Apple operating system. However, when shipped on the Internet, they confuse things. So if you had a file called Race Cars, you would need to encode that into Race%20Cars.

The percent sign (%) tells the decoding routine that encoding has begun, and the next two characters are hex numbers that correspond to the ASCII equivalent value of a space.

If you were trying to send HTML tags as part of your data transfer, the < and > symbols would need to be encoded. They encode as %3C for <, and %3E for >.

Hex is simply a numbering system with values ranging from 0 to 15, where the numbers 10–15 are encoded as the letters A–F. So the hex range is 0–F. A code is always made up of a percent sign followed by two hex values. You don't really need to understand hex values any better than that—just read the numbers from the table and encode them as needed.

Table 27.1 lists the ASCII characters that must be encoded in your URL. It lists both the decimal and hex values. The decimal values are only included for your information; you must use the hex values.

Table 27.1. URL characters that must encoded.

Character	*Decimal*	*Hex*
tab	09	09
space	16	20
"	18	22
(	40	28
)	41	29
,	44	2C
.	46	2E
;	59	3B
:	58	3A
<	60	3C
>	62	3E
@	64	40
[	101	5B
\	102	5C
]	103	5D
^	104	5E
`	106	60
{	113	7B
¦	114	7C
}	115	7D
ñ	116	7E

In addition to the reserved characters listed in Table 27.1, there are several other characters that should be encoded if you don't want them to be interpreted by your server or client for their special meaning.

- Encode the question mark (?) as %3F; otherwise, you will begin a query string too early.
- Encode the ampersand (&) as %26; otherwise, you start the separation of a name/value pair when you don't want to.
- Encode the backslash (/) as %2F; otherwise, you will start a new directory path.
- Encode the equal sign (=) as %3D; otherwise, you may bind a name/value pair when you don't want to.

- Encode the number sign (#) as %23. This is used to reference another location in the same document.
- Encode the percent sign (%) as %25; otherwise, you will really confuse everyone. Decoding will start at your unencoded %.

If you want to look at the gory details of MIME/URL encoding, you can get RFC 1552, the MIME message header extensions document, off the Net. It has the encoding format in section three and is available with the other Internet RFC documents at `http://ds.internic.net/ds/dspg1intdoc.html`.

The Encoding Steps

So now you know the basis for encoding all the data. Remember that all data sent on the Internet is URL-encoded. The steps used for getting your data encoded follow. These steps work for both the POST and the GET methods.

1. Data is transferred as name/value pairs.
2. Name/value pairs are separated from other name/value pairs by the ampersand.
3. Name/value pairs are identified with each other by the equal sign. If no data is entered and a default value is defined, the value will be the default value. If no default value is defined, the value will be empty but a name/value pair will be sent.
4. Spaces in value data are a special case. They are converted to the plus sign (+).
5. Reserved characters cannot be used in the URL; they must be encoded.
6. Characters that have special meaning must be encoded before being sent to the browser.
7. Characters are encoded by being converted to their hex values.
8. Encoded characters are identified with a percent sign and two hex digits (%*NN*).

Decode FORM Data

Now that you know the steps for encoding the data that's sent from the browser to your server, the decoding is easy. This section presents the basic steps of decoding the data sent from your HTML form to the Web server. This is where you put your new knowledge of CGI programming and its environment to work.

Your CGI program will be activated based on the ACTION field in the HTML <FORM> tag on your Web page. The data from your form will be available either in the environment variable QUERY_STRING or at STDIN. Thus, step one of decoding the incoming URL-encoded FORM data is figuring out how the form sent your data. To do this, you read the environment variable REQUEST_METHOD. Your CGI programs rely on the environment variable for a lot of information; the environment variables available to your programs are covered in the last sections of this chapter. The REQUEST_METHOD environment variable should equal GET or POST, the two methods for sending data from an HTML form. If the REQUEST_METHOD isn't set, the default method for sending data from your HTML form is the GET method.

Step two in decoding the incoming form data is to read the data into a common variable your code can process. If the data was sent using the GET method, your program needs to copy the environment variable QUERY_STRING into your decoding variable. If the data was sent using the POST method, the data is available at STDIN. The environment variable CONTENT_LENGTH contains the amount of data your program needs to read from STDIN. Your CGI program could be written in Perl 5, C/C++, Java, TCL, Visual Basic, or some other language of your choice. You might use a loop, a simple read, or some other function to move the data from STDIN into your decoding variable. Just remember that the amount of data you need to move from STDIN is available in the CONTENT_LENGTH environment variable.

Now that you have the data from your form, step three starts the actual decoding process. Remember that the first step of sending URL-encoded data was placing the data into name/value pairs and that name/value pairs are separated by an ampersand. If your CGI program is written in Perl 5, use the split function to separate name/value pairs based on the ampersand. Save the split data into an array for further decoding. If you're using Java, the stringTokenizer function performs a similar function, separating the data into tokens based on the ampersand. C has a similar tokenizing function called stoken for tokenizing strings. At the end of this section is part of a program written by Steven E. Brenner, cgi-lib.pl. This program is included on the CD-ROM (and can be found online at www.bio.coma.ac.uk/cgi-lib) and illustrates the steps you're learning about here. I've included this program because it is well commented, so even if you're unfamiliar with Perl, you should be able to understand the program.

The fourth step in decoding the data is converting the plus signs back into spaces. Use a straightforward substitution function for this action. In Perl you could use tr or s. In C you might just search for the plus sign and replace it with a space. Regardless of the method, the plus sign is replaced with a space character, as it was before it was URL encoded.

Step five separates the name/value pair based on the equal sign. In Perl and Java, you can again use the split or stringTokenizer functions, respectively, to separate the data.

Step six involves converting all those hex-encoded special characters shown in Table 27.1 back into their normal state. Remember that all the hex-encoded characters begin with a percent sign. Your code should look for that first percent sign, and you know the next two characters will be hex values. Convert these hex values back to their ASCII equivalent, and your data is now URL decoded.

In Perl and Java, a convenient storage place for name/value pairs is in an associative array. Save the URL-decoded name/value pairs and begin the decoding steps on the next set of URL encoded name/value pairs. When your code has processed all the name/value pairs, you can take the data and do as you please with it.

As promised, part of Steven E. Brenner's code for reading and decoding form data is shown in Listing 27.3.

Listing 27.3. Reading and decoding form data in Perl 5.

```
sub ReadParse {
  local (*in) = shift if @_;    # CGI input
  local (*incfn,                # Client's filename (may not be provided)
     *inct,                  # Client's content-type (may not be provided)
     *insfn) = @_;           # Server's filename (for spooled files)
  local ($len, $type, $meth, $errflag, $cmdflag, $perlwarn, $got);

  # Disable warnings as this code deliberately uses local and environment
  # variables which are preset to undef (i.e., not explicitly initialized)
  $perlwarn = $^W;
  $^W = 0;

  binmode(STDIN);   # we need these for DOS-based systems
  binmode(STDOUT);  # and they shouldn't hurt anything else
  binmode(STDERR);

  # Get several useful env variables
  $type = $ENV{'CONTENT_TYPE'};
  $len  = $ENV{'CONTENT_LENGTH'};
  $meth = $ENV{'REQUEST_METHOD'};

  if ($len > $cgi_lib'maxdata) { #'
      &CgiDie("cgi-lib.pl: Request to receive too much data: $len bytes\n");
  }

  if (!defined $meth || $meth eq '' || $meth eq 'GET' ||
      $type eq 'application/x-www-form-urlencoded') {
    local ($key, $val, $i);

    # Read in text
    if (!defined $meth || $meth eq '') {
      $in = $ENV{'QUERY_STRING'};
      $cmdflag = 1;  # also use command-line options
    } elsif($meth eq 'GET' || $meth eq 'HEAD') {
      $in = $ENV{'QUERY_STRING'};
    } elsif ($meth eq 'POST') {
       if (($got = read(STDIN, $in, $len) != $len))
      {$errflag="Short Read: wanted $len, got $got\n";};
    } else {
      &CgiDie("cgi-lib.pl: Unknown request method: $meth\n");
    }

    @in = split(/[&;]/,$in);
    push(@in, @ARGV) if $cmdflag; # add command-line parameters

    foreach $i (0 .. $#in) {
      # Convert plus to space
      $in[$i] =~ s/\+/ /g;

      # Split into key and value.
      ($key, $val) = split(/=/,$in[$i],2); # splits on the first =.

      # Convert %XX from hex numbers to alphanumeric
      $key =~ s/%([A-Fa-f0-9]{2})/pack("c",hex($1))/ge;
      $val =~ s/%([A-Fa-f0-9]{2})/pack("c",hex($1))/ge;
```

continues

Listing 27.3. continued

```
    # Associate key and value
    $in{$key} .= "\0" if (defined($in{$key})); # \0 is the multiple separator
    $in{$key} .= $val;
  }
```

The HTTP Headers

If HTML is responsible for gathering data to send to your CGI program, how does it get there? The data gathered by the browser gets to your CGI program through the magic of the Hypertext Transport Protocol request headers, or HTTP headers for short. The HTML tags tell the browser what type of HTTP header to use to talk to the server, your CGI program. The basic HTTP headers for beginning communication with your CGI program are `GET` and `POST`.

If the HTML tag calling your program is a hypertext link, such as `<a href=www.domain.com/progam.cgi>, call a CGI program </a>`, then the default HTTP request `METHOD GET` will be used to communicate with your CGI program. If instead of using a hypertext link to your program, you use the HTML `<FORM>` tag, then the `METHOD` attribute of the `<FORM>` tag defines what type of HTTP request header will be used to communicate with your CGI program. If the `METHOD` field is missing or set to `GET`, the HTTP `METHOD` request header type is `GET`. If the `METHOD` attribute is set to `POST`, then a `POST METHOD` request header will be used to communicate with your CGI program.

After the method of sending the data is determined, the data is formatted and sent using one of two means. If the `GET METHOD` is used, the data is sent via the URL field. If the `POST METHOD` is used, the data is sent as a separate message, after all the other HTTP request headers have been sent.

After the browser has determined how it is going to send the data, it creates an HTTP request header identifying where on the server your CGI program is located. The browser sends this HTTP request header to the server. The server receives the HTTP request header and calls your CGI program. Several other request headers can go along with the main request header, providing to the server and your CGI program useful information about the browser and this connection.

Your CGI program performs some useful function, and then it tells the server what type of response it wants to send back to the server.

So where are we so far? The data has been gathered by the browser using the format defined by the HTML tags. The data/URL request has been sent to the server using HTTP request headers. The server used the HTTP request headers to find your CGI program and call it. Now your CGI program has done its thing and is ready to respond to the browser. What happens next? The server and your CGI program collaborate to send HTTP response headers back to the browser.

What about the data, the Web page, your CGI program generated? Well, that is what the HTTP response headers are for. The HTTP response headers describe to the browser what type of data is being returned to the browser.

Your CGI program can generate all of the HTTP response headers required for sending data back to the client/browser by calling itself a nonparsed-header CGI (nph-cgi) program. If your CGI program is an nph-cgi program, then the server will not parse or look at the HTTP response headers generated by your CGI program. The HTTP request headers will be sent directly to the requesting browser, along with data/HTML generated by your CGI program.

However, the more common form of returning HTTP response headers is for your CGI program to generate the minimum required HTTP request headers; usually just a `Content-Type:` HTTP response header is required. The server then parses or looks for the response header your CGI program generated and determines what additional HTTP response headers should be returned to the browser.

The `Content-Type:` HTTP response header identifies to the browser the type of data that will be returned to the browser. The browser uses the `Content-Type:` response header to determine the types of viewers to activate so the client can view things like inline images, movies, and HTML text.

The server adds the additional HTTP response headers it knows are required and then bundles the set of the headers and data up in a nice TCP/IP package and sends it to the browser. The browser receives the HTTP response headers and displays the returned data as described by the HTTP response headers to your customer.

The HTTP Headers

HTTP headers are the language your browser and client use to talk to each other. Think of each of the HTTP headers as a single message. In the client and server sense, first there are a bunch of questions, which are the request headers, and then there are the answers to those questions, the response headers.

Status Codes in Response Headers

Status codes in the response header tell the client how well your request for a URL went. There are five basic types of HTTP status codes. A status code is always included in the HTTP response headers returned from the HTTP request headers. The status codes are shown in Tables 27.2 and 27.3.

The following information is a subset of an Internet draft known as Hypertext Transfer Protocol, and is available at

`http://www.ics.uci.edu/pub/ietf/http/draft-ietf-http-v10-spec-01.html`

The status code element is a three-digit integer result code of the attempt to understand and satisfy an HTTP request header. The reason phrase is intended to give a short textual description of the status code.

The first digit of the status code defines the class of response. The last two digits do not have any categorization role. There are five values for the first digit, as listed in Table 27.2.

Table 27.2. Status codes and their meanings.

Numeric	*English*	*Meaning*
1xx	Informational	Not used, but reserved for future use.
2xx	Success	The action was successfully received, understood, and accepted.
3xx	Redirection	Further action must be taken in order to complete the request.
4xx	Client error	The request contains bad syntax or cannot be fulfilled.
5xx	Server error	The server failed to fulfill an apparently valid request.

The individual values of the numeric status codes defined for HTTP/1.0 are presented in Table 27.3.

Table 27.3. The status codes and the reason field and its meaning.

Code	*Reason field*	*Meaning*
201	Created	The request has been fulfilled and resulted in a new resource being created. The newly created resource can be referenced by the URL(s) returned in the URL header field of the response, with the most specific URL for the resource given by a Location header field.
202	Accepted	The request has been accepted for processing, but the processing has not been completed.

CODE	*Reason field*	*Meaning*
203	Non-authoritative information	The returned meta-information in the entity header is not the definitive set as available from the origin server, but is gathered from a local or a third-party copy.
204	No content	The server has fulfilled the request, but there is no new information to send back.
300	Multiple choices	The requested resource is available at one or more locations and a preferred location could not be determined via content negotiation.
301	Moved permanently	The requested resource has been assigned a new permanent URL and any future references to this resource should be done using one of the returned URLs.
302	Moved temporarily	The requested resource resides temporarily under a different URL.
303	See other	The requested resource resides under a different URL and should be accessed using a GET method on that resource.
304	Not modified	If the client has performed a conditional GET request and access is allowed, but the document has not been modified since the date and time specified in the If modified since field, the server shall respond with this status code and not send an entity body to the client.
400	Bad request	The request could not be understood by the server because of a malformed syntax.

continues

Table 27.3. continued

CODE	Reason field	Meaning
401	Unauthorized	The request requires user authentication. The response must include a WWW authenticate header field containing a challenge applicable to the requested resource.
402	Payment required	This code is not currently supported, but is reserved for future use.
403	Forbidden	The server understood the request but is refusing to perform the request because of an unspecified reason.
404	Not found	The server has not found anything matching the request URL.
405	Method not allowed	The method specified in the request line is not allowed for the resource identified by the request URL.
406	None acceptable	The server has found a resource matching the request URL, but not one that satisfies the conditions identified by the accept and accept encoding request headers.
407	Proxy authentication required	This code is reserved for future use. It is similar to 401 (Unauthorized), but indicates that the client must first authenticate itself with the proxy. HTTP/1.0 does not provide a means for proxy authentication.
408	Request timeout	The client did not produce a request within the time that the server was prepared to wait.
409	Conflict	The request could not be completed because of a conflict with the current state of the resource.

CODE	Reason field	Meaning
410	Gone	The requested resource is no longer available at the server and no forwarding address is known.
411	Authorization refused	The request credentials provided by the client were rejected by the server or insufficient to grant authorization to access the resource.
500	Internal server error	The server encountered an unexpected condition that prevented it from fulfilling the request.
501	Not implemented	The server does not support the functionality required to fulfill the request.
502	Bad gateway	The server received an invalid response from the gateway or upstream server it accessed in attempting to fulfill the request.
503	Service unavailable	The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.
504	Gateway timeout	The server did not receive a timely response from the gateway or upstream server it accessed in attempting to complete the request.

The Method Request Header

Status codes are part of the server's response to an HTTP request header. The client makes the request of the server, and the server builds the response headers. The most common request header is the GET method request header.

The client sends to the server several request headers defining to the server what the client wants, how the client can accept data, how to handle the incoming request, and any data that needs to be sent with the request.

The first request header for every client/server communication is the Method request header. This request header tells the server what other types of request headers to expect and how the server is expected to respond.

The most common request methods are GET, POST, and HEAD. The HTTP specification also allows for the PUT, DELETE, LINK, and UNLINK methods, along with an undefined extension method. You will be dealing primarily with the GET and POST methods, so this chapter focuses on them.

Each of the request headers identifies a URL to the server. The difference between GET and POST is the effect on how data is transferred. The HEAD request method affects how the requested URL is returned to the client.

The next section discusses the FULL request method line. This is the request header that includes the type of access (GET, POST, HEAD, and so on) that the client is requesting. Of all the request headers, this is the one that really makes things work. This is the request header that tells the server which Web page you want returned to the browser. Without this header, no data will be transferred to the calling client.

The Full Method Request Header

The FULL Method request header is the first request header sent with any client request. The FULL request method line is made up of three parts separated by spaces: the method type, the URL requested, and the HTTP version number.

The syntax of the FULL Method request header is as follows:

```
Request_Method URL HTTP_Protocol_Version \n (newline)
```

For example:

```
GET http://www.practical-inet.com/index.html HTTP/1.0
```

The Request_Method can be any of the following method types: GET, POST, HEAD, PUT, DELETE, LINK, and UNLINK.

The URL is the address of the file, program, or directory you are trying to access.

The HTTP_Protocol_Version is the version number of the HTTP protocol that the client/browser can handle.

The GET HTTP Header

The GET method is the default method for following links and passing data on the Internet. When you click on a link, your browser is sending a GET method request header. When you click the Submit button on a form, if the method is undefined in the ACTION field of the <FORM>, the GET method request header is used to call the CGI program that handles the <FORM> data.

When you click on a URL, it usually is of the form http://www.somewhere.com/filename.html. A GET method request header is generated along with any other request header the browser might want to send. The URL is located and returned by the browser, unless an If-Modified-Since request header was sent along with the other request headers.

When the If-Modified-Since header is included in the request headers, the browser will check the modification date of the requested URL and only return a new copy if it has been modified after the date specified.

When you click on a URL and that URL is a request for another Web page, you have sent a GET method request header and lots of other headers to your server.

The Requested URL

The second field in the first line of the FULL method request header is the Requested URL field. The URL tells the server what file or service is requested.

Normally, the FULL Method request header is for a file on the server. When this is the case, the absolute path of the file/URL is included in the Method request header. An example of a GET Method request header is GET / HTTP/1.0

The format of the requested URL is the absolute pathname of the document root. Consider this sample GET method request header:

```
/~yawp/test/env.html/
```

The following bullets refer to the preceding example to explain what the document root pathname means:

- The absolute pathname is the directory and filename of the URL, beginning at the / directory. For this example, I show the absolute pathname to my personal directory ~yawp with a subdirectory of test, and a filename of env.html.
- This / directory is defined by your server administrator as the starting location for all Web pages or URLs on the server. This is also called the document root.
- On my server, the server administrator has defined a public Web directory within every user's home directory, so the actual path to the env.html file is yawp/public-web/-test/env.html. On my commercial server, the document root looks like www-practical-inet.com, but the real path is /usr/local/business/http/practical-inet/.2.

The Proxy GET Method Request Header

If the target of the URL is a proxy server, it should send an absolute URL. An absolute URL includes the domain name and the full pathname to the requested URL. The domain name in the following example is www.w3.org:

```
GET http://www.w3.org/hypertext/WWW/TheProject.html HTTP/1.0
```

The HTTP Version

The last field in the FULL Method request header is HTTP version. The only valid values at this moment are HTTP/1.0 followed by a CRLF.

Table 27.4 summarizes the request/response headers used by the server and client to communicate with each other. They are defined completely in the HTTP specification. This chapter covers several of the more common headers. Several of the headers, such as the Method header are covered in detail in the following sections. A summary of those headers is included in Table 27.4.

The most important thing to remember is that the request/response headers are the means your client and browser use tell each other what is needed and what is available.

Table 27.4. A summary of the HTTP request/response headers.

Request/response header	*Meaning*
Accept	This header tells the server what type of data the browser can accept. Examples are text, audio, images.
Accept-Charset	This header tell the server what character sets the browser prefers. The default is US-ASCII.
Accept-Encoding	This header tells the server what type of data encoding the browser can accept. Examples are compress and gzip.
Accept-Language	This header tells the server what natural language to the browser prefers. The default is English.
Allow	This header tells the browser what request methods are allowed by the server. Examples are GET, HEAD, POST.
Authorization	This header is used by the browser to authenticate itself with the server. It is usually sent in response to a 401 or 411 code.
Content-Encoding	This header is used to identify the type of encoding used on the data transfer. An example is compressed.
Content-Language	This header identifies the natural language of the data transferred.
Content-Length	This header identifies the size of the data transfer in decimal bytes.
Content-Transfer-Encoding	This header identifies the encoding of the message for Internet transfer. The default is binary.
Content-Type	This header identifies the type of data being transferred. An example is text/html.

Request/response header	*Meaning*
Date	This header identifies the GMT date/time the data transfer was initiated.
Expires	This header identifies the date/time the data should be considered stale. This header is often used by caching clients.
Forwarded	This header is used by proxy servers to indicate the intermediate steps between the browser and server.
From	This header should contain the Internet e-mail address of the client. This header is no longer in common use.
If-Modified-Since	This header makes the request method a conditional request. A copy of the requested URL is only returned if it was modified after the time specified.
Last-Modified	This header identifies the date/time when the URL was last modified.
Link	This header is used for describing a relationship between two URLs.
Location	This header is used to define the location of a URL. Typically this header is used to redirect the client to a new URL.
MIME-Version	This header is used to indicate what version of the MIME protocol was used to construct the transferred message.
Orig-URL	This request header is used by the client to specify to the server the original URL of the requested URL.
Pragma	This header is used to specify special directives that should be applied to all intermediaries along the request/response chain. This header is usually used to provide directives to proxy servers or caching clients.
Public	This header is used to list the set of nonstandard methods supported by the server.
Referer	This request header identifies to the server the address (URL) of the link that was used to send the method request header to the server.
Retry-After	This response header is used to identify to the client a length of time to wait before trying the requested URL again.

continues

Table 27.4. continued

Request/response header	*Meaning*
Server	This response header identifies the server software used by the server.
Title	This header identifies the title of the URL.
URL-Header	Uniform resource identifier.
User-Agent	This request header identifies the type of browser making the request.
WWW-Authenticate	This response header is required when status response headers of Unauthorized (401) or Authorization Refused (411) occur. This header is used to begin a challenge/response sequence with the client.

The HTTP Response Header

After the server receives the request headers, it begins to generate the correct response. The server starts by looking up the URL in the GET Method, and then generates the response headers. The GET Method request header tells the server what URL is desired; the other request headers tell the server how to send the data back to the client.

After the server gets a request in, it must choose a valid response. It starts with a response status line, which gives the protocol version followed by a status code. The format of a response status line is as follows:

```
PROTOCOL/Version_Number Status_Code Status_Description
```

The only valid protocol right now is HTTP, and version 1.0 is the standard at the moment.

Figure 27.2 shows the response headers generated when the server receives a GET Method request header.

The following sections briefly discuss these response headers. These are the basic ones that will be returned from almost any request header.

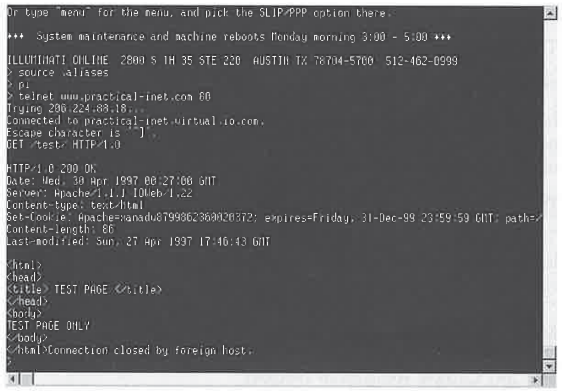
Status Response Line

The Status response line is

```
HTTP/1.0 200 OK
```

Nothing to write home about in this response header. Simple and straightforward. The HTTP version number is 1.0. The status is 200. The status description is OK. So this means our server found our requested URL and is going to return it to the browser.

FIGURE 27.2.
The server response headers to a GET *Method request header.*



The Date Response Header

This is the Date response header:

```
Date: Wed, 09 Apr 1997 22:41:26 GMT
```

This is the time that the server generated the response to the request header. The date must be in Greenwich Mean Time (GMT). The date can be in one of three formats, as described in Table 27.5.

Table 27.5. Greenwich Mean Time (GMT) format.

Format example	*Format description*
`Sun, 13 Apr 1997 06:15:10 GMT`	Originally defined by RFC 822 and updated by RFC 1123, this is the preferred format.
`Sunday, 17-Apr-97 06:15:10 GMT`	Defined by RFC 850 and made obsolete by RFC 1036, this format is in common use; it's based on an obsolete format and lacks a four-digit year.
`Sun Apr 17 06:15:10 1994`	This is the ANSI standard date format represented in C's `asctime()` function.

Only one Date response header is allowed per message. Because it is important for evaluating cached responses, the server should always include a Date response header. Cached responses are beyond the scope of this book but, in short, can be part of a request/response chain, used to speed up URL transfers.

The Server Response Header

This is the Server response header field:

```
Server: Apache/1.1 IOWeb/1.22
```

It contains information about the server software used to create the response. If you are having problems with your CGI working with a particular site, this header field can identify the type of server software your CGI is failing with.

The Content-Type Response Header

The Content-Type header field tells your browser what type of media is appended after the last response header:

```
Content-type: text/html
```

Media types are defined in an appendix at the end of the book.

The Set-Cookie Response Header

The Set-Cookie header is used as a state maintaining header. It is critical to Shopping Cart and other Electronic Commerce CGI applications. The Set-Cookie header is always set by your ISP. You can add Set-Cookie headers just like other HTTP headers. This is the Set-Cookie response header:

```
Set-Cookie: Apache=pentagon21775860625686440;
expires=Friday, 31-Dec-99 23:59:59 GMT; path=/
```

The Content-Length Response Header

The Content-Length header field indicates the size of the appended media in decimal numbers, in 8-bit byte format, referred to in the HTTP specification as OCTETS:

```
Content-length: 1529
```

This header is often used by the server to determine the amount of data sent by the client, when posting `<FORM>` data.

The Last Modified Response Header

In this example we are passing a file, `URL`, that is of type `text/html`, so the Last Modified field is the time the file was last modified:

```
Last-Modified: Wed, 09 Apr 1997 22:37:43 GMT
```

This field is used for caching information. When an If-Modified-Since request header is sent, this field is used in determining whether the data should be transferred at all.

The Enclosed URL

The last line of the response headers is blank; after that, the requested URL is shipped to the client. This is the blank line just before the opening `<HTML>` tag in Figure 27.2.

All of your HTTP response and request header chains must end with a blank line.

The last `print` statement of an HTTP header program you write should print a blank line:

```
print "Last-modified: $last_modified_variable\n\n";
```

Notice in the preceding example that two new lines (`\n`) are printed. One is always required for every HTTP header, but the second new line indicates to the server or client the end of any incoming or outgoing HTTP headers. Everything after that first blank line is supposed to be in the format defined by the Content-Type header.

Environment Variables

Not all environment variables are created equal. The environment variable is the server's way of communicating with your CGI program, and each communication is unique.

The uniqueness of each communication with your CGI program is based on the request headers that are sent by the Web page client when it calls your CGI program. If your Web page client is responding to an Authorization response header from the server, then it will send Authorization request headers. Because the request headers define a number of your environment variables, you can never be sure which environment variables are available.

Environment Variables Based on the Server

However, some of the environment variables are always set for you and are not dependent on the CGI request. These environment variables typically define the server on which your CGI program runs. The following environment variables are based on your server type and should always be available to your CGI program.

GATEWAY_INTERFACE

The environment variable `GATEWAY_INTERFACE` is the version of the CGI specification that your server is using. The CGI specification is defined at `http://hoohoo.ncsa.uiuc.edu/cgi/`. This is an excellent site for further information about CGI. At this time, CGI is at revision 1.1.

SERVER_ADMIN

The environment variable `SERVER_ADMIN` should be the e-mail address of the Web guru on your server. When you can't figure out the answer yourself, this is the person to e-mail. Be careful, though. These people are usually very busy. You want to establish a good relationship early, so he or she will respond to your requests later. Make sure you have tried all of the simple things, everything you know first, before you ask this person questions. This is definitely an area where "crying wolf" can have a negative affect on your ability to get your CGI programs working. When you have a really tough problem that no one seems able to figure out, you want your server administrator to respond to your questions.

SERVER_NAME

The environment variable SERVER_NAME contains the domain name of your server. If a domain name is not available, it will be the IP number of your server. This should be in the same URL format as your CGI program was called.

SERVER_SOFTWARE

The environment variable SERVER_SOFTWARE contains the type of server your CGI program is running under. You can use this variable to figure out what type of security methods are available to you—whether server-side includes are even possible, for example. This way you don't have to ask your Webmaster these simple questions.

Environment Variables Based on the Request Headers

This next set of environment variables give your CGI program information about what is happening during this call to your program. These environment variables are defined when the server receives the request headers from a Web page.

SERVER_PROTOCOL

The SERVER_PROTOCOL environment variable defines the protocol and version number being used by this server. For the time being, this should be HTTP/1.0. The HTTP protocol is the only server protocol used for the WWW at the moment. But like most good designs, this environment variable is designed to allow CGI programs to operate on servers that support other communication protocols.

SERVER_PORT

The SERVER_PORT environment variable defines the TCP port that the request headers were sent to. The port is like the telephone number that is used to call the server. The default port for server communication is 80. When you see a number appended after the domain name server, this is the port number that the request was sent to, for example, www.io.com:80. Because the default port is 80, it is not generally necessary to include the port number when making URL links.

HTTP REQUEST_METHOD

The HTTP REQUEST_METHOD environment variable is the HTTP Method request header converted to an environment variable. As explained earlier in this chapter, the following request methods are possible: GET, POST, HEAD, PUT, DELETE, LINK, and UNLINK. GET and POST are certainly the most common for your CGI program and define where incoming data is available to your CGI program. If the method is GET, the data is available at QUERY_STRING. If it is POST, the data is available at STDIN and the length of the data is defined by the environment variable CONTENT_LENGTH. The HEAD request method is normally used by bots searching the Web for page links. The other methods are not quite as common and tell the server to modify a URL/file on the server.

PATH_INFO

The PATH_INFO environment variable is only set when there is data after the CGI program (URL) and before the beginning of the QUERY_STRING. Remember, the QUERY_STRING begins after the question mark (?) on the link URL or ACTION field URL. PATH_INFO can be used to pass any type of data to your CGI program, but it is usually used to send information about finding files or programs on the server. The server strips everything after it finds the target CGI program (URL) and before it finds the first question mark (?). This information is URL decoded and then placed in the PATH_INFO variable.

PATH_TRANSLATED

The PATH_TRANSLATED environment variable is a combination of the PATH_INFO variable and the DOCUMENT_ROOT variable. It is an absolute path from the root directory of the server to the directory defined by the extra path information added from PATH_INFO. This is called an *absolute* path. This type of path is often used when your CGI program moves in and out of different directories or different shell environments. As long as your server doesn't change, you can use the absolute path regardless of where you put or move your CGI program. Sometimes absolute paths are considered bad because you cannot move your CGI program to another server. You have to decide which is more likely:

1. Your CGI program will change directories.
2. You will change servers.
3. The absolute path will change on your existing server. This can happen when your server adds or removes disks.

PATH

The PATH environment variable is not strictly considered a CGI environment variable. This is because it actually includes information about your UNIX system path.

SCRIPT_NAME

The SCRIPT_NAME environment variable gives you the path and name of the CGI program that was called. The path is a relative path, which starts at the document root path. You can use this to build self-referencing URLs. Suppose you want to return a Web page and you want to generate HTML that includes a link to the called CGI program. The print string would look like this:

```
print "<a href=http://$SERVER_NAME$SCRIPT_NAME>
This is a link to the CGI program you just called </a>";
```

SCRIPT_FILENAME

The SCRIPT_FILENAME environment variable gives the full path to the CGI program. You do not want to use this when building a self-referencing URL. Remember, the server is making some assumptions about how you will access your CGI program. The full pathname would be

appended onto the server's full pathname and totally confuse your poor server. The server starts with the SERVER_NAME; from there it determines the document root, and then it adds on the path to your CGI program.

QUERY_STRING

The QUERY_STRING environment variable contains everything included on the URL after the question mark. The setup for a QUERY_STRING is normally performed by your browser when it builds the request headers. However, you can create the data for your own QUERY_STRING if you want to by including a question mark in your hypertext reference and then URL encoding any data that is included after the question mark. This is just one more way to send data to your program. Two big drawbacks to using the QUERY_STRING are the YUK! factor and the size of the input buffer. The YUK! factor means your data will be displayed back to your client in the Location field. The size problem means you have a limitation on how much data you can send to your program using this method. The amount of data you can send without exceeding the input buffer is server specific, so I can't give you any hard rules. But you should try to limit all data you send using this method to under 1024 bytes.

REMOTE_HOST

The REMOTE_HOST environment variable contains the domain name of the client accessing your CGI program. You can use this information to help figure out how your script was called. If the domain name is unavailable to your server, then this field will be left empty. However, if this field is empty, the REMOTE_ADDR environment variable will be filled in. Your program can read this environment variable from right to left. There can be more that one subhierarchy after the first period (.), so be sure to write your code to deal with more than one level of domain hierarchy to the left of the period.

REMOTE_ADDR

The REMOTE_ADDR environment variable has the numeric Internet Protocol (IP) address of the browser or remote computer calling your CGI program. Read the REMOTE_ADDR from right to left. The right-most number defines today's connection to the remote server. Or at least, this will be the case when your Web browser client connects from a modem to a commercial server.

AUTH_TYPE

The AUTH_TYPE environment variable defines the authentication method used to access your CGI program. The AUTH_TYPE will usually be basic, because this is the primary method for authentication of the Net right now. The AUTH_TYPE defines the protocol-specific authentication method used to validate the user.

REMOTE_USER

The REMOTE_USER environment variable identifies the caller of your CGI program. This value is only available if server authentication is turned on. This is the username authenticated by the USERNAME/PASSWORD response to a response status of Unauthorized Access (401) or Authorization Refused (411).

REMOTE_IDENT

The REMOTE_IDENT environment variable is only set if the remote username is retrieved from the server using the identd. This only occurs if your Web server is running the identd identification daemon, a protocol that identifies the user connecting to your CGI program. However, just having your system running identd is not sufficient; the remote server making the HTTP request must also be running identd.

CONTENT-TYPE

The CONTENT-TYPE environment variable defines the type of data attached with the request method. If no data is sent, then this field is left blank. The content type will be application/x-www-form-urlencoded when posting data from a form.

CONTENT-LENGTH

The CONTENT-LENGTH environment variable specifies the amount of data attached after the end of the request headers. This data is available at STDIN and is identified with the POST or PUT method.

Summary

The fundamental building blocks for CGI programming are outlined in this chapter. The client, your Web browser, sends HTTP request headers to your Web server. The Web server calls your CGI program. Your CGI program uses environment variables to interpret the request from the Web browser and then processes the data (making a sale), collects data from a database, puts an item in a shopping cart, or answers a question from a customer—the possibilities are endless. After the request is processed, your CGI program returns HTTP response headers and HTML to the Web browser.

An entire book could be devoted to the topic of CGI programming, and many books have been written just about CGI—in fact, I wrote one myself, *Teach Yourself CGI Programming with Perl 5 in a Week*. In the 500 pages of this tutorial-style book, you can learn the details of CGI programming and work through lots of CGI programming examples, including mail tools, image mapping, shopping carts, and much more.