

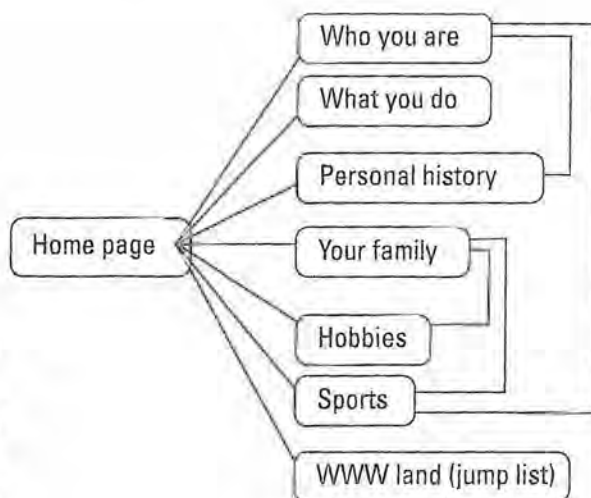


Figure 9-4:
YourWeb's
structure.

Board the whole story

This simple sketch doesn't provide enough information for you to fully visualize your Web. What you really need to do now is to prepare a storyboard for these pages — that is, unless you can mentally picture the elements and links on each of the eight separate pages.

Every movie, TV show, and comic book gets storyboarded before any production takes place. Producing a set of Web pages is a lot like making a TV show, especially if you think of each Web page as a separate scene. Over time, though, an entire collection of Web documents and associated materials will evolve from your work, making it resemble a whole season's worth of TV episodes instead of a single show.

To prepare a storyboard, simply prepare a sketch of the layout of each Web page with the URLs for links written on each one. For small Webs, some Web authors use a white marker board with colored pens. The colors are handy for showing different types of links, forms, or other HTML elements.

For more complex Webs, many authors use a sheet of paper for each Web page, some string, some push pins, and a large bulletin board (cork type, not BBS). This allows for complex arrangements that can be easily changed. Also, the storyboard method is invaluable for identifying potential hypertext links if you attach the text of each Web page to its layout sheet. Whenever you're creating a Web of more than a handful of pages, create a storyboard. Doing this saves you much more time than it takes initially, and after you're finished, you'll appreciate its value.

Anchors Away: Jumping Around Your Documents

We did say that it wasn't too terrible to create Web pages spanning up to three screens if your information demands the extra room. You may even expand your home page to more than one screen if you carefully drop your anchors and don't go overboard on the images.

You can use two different anchor tag attributes for movement within your pages. To provide viewers with links to specific parts within a Web page (called intradocument linking), use the `NAME="text"` anchor to provide the destination of an `HREF="#label"` tag. Use the standard `HREF="URL"` (called interdocument linking) to let users jump from page to page. Or to jump to a specific location within another page, you can combine both approaches and use `HREF="URL#label"`.

There's an important distinction to note when talking about following links inside a browser program. With interdocument linking (between documents), most browsers land the reader at the first line of the target document. On the other hand, intradocument linking (within the same document) takes you to a place other than the default top of the page, unless there's a named anchor at the top of the page and the URL calls this anchor out.

Here's another interesting quirk about browsers — namely, their behavior with named anchors that occur near the bottom of a document. If an anchor appears near the bottom, most browsers do NOT bring the named anchor to the first line on the screen. This is because the browser usually renders a full screen of text; thus, if the anchor is near the bottom of a document, the link may take you to a point toward the bottom of the screen rather than to the absolute bottom of the document itself.

Linking to text in another page

The *HTML For Dummies* home page provides a good example of how you should use the `NAME="text"` attribute in an anchor tag. The HTML line in the home page document

```
<A HREF="search4d.htm#menu"><IMG SRC="graphics/menu.gif" ALT="MENU "
BORDER=0></A>
```

specifies a hypertext relationship between the `menu.gif` graphic (the yellow button with the blue square inside) in the current document and the named anchor `menu` in the target document that is in the `search4d.htm` file. The pound sign (`#`) indicates that the browser should position the reader not at the

top of the page, but at the named anchor in the target document, menu in this case. If the anchor isn't found, you'd get the default instead, which is the top of the document.

The browser displays the information at that location in the file, starting with the heading, "Menu and Order of Pages." This may seem unduly abstract, but you'll catch on if you remember this: The anchor with the `NAME="text"` attribute is the destination for some link. As the author, you have control over how your information is displayed. If you think users will find a nugget of information within a specific page relevant and important, then give it a name with the `NAME` attribute and create a link to it.

Linking to text within a page: table of contents links

You can use the `NAME="text"` attribute to create a really jumping table of contents (TOC) for long text documents. Providing a linked TOC takes a little more time, but it's a great way to impress your users. Remember to provide a link back to the TOC after each block of text in the destination document.

The following HTML code illustrates how to use the TOC links within a large document:

```
<!-- Make this an anchor for return jumps.-->
<A NAME="TOC">Table of Contents</A>
<P>
<!-- This is the link to the section 1. below.-->
<A HREF="#SEC1">Section 1.</A><BR>
<A HREF="#SEC2">Section 2.</A><BR>
<A HREF="#SEC3">Section 3.</A><BR>

<!-- This is a named anchor called "SEC1".-->
<A NAME="SEC1"><H2> CFR Section 1.</H2></A>
<P> Text of section 1 is here.
<BR>
<!-- This is a link back to the TOC at the top of the page-->
<A HREF="#TOC">(TOC)</A>
<P>
<A NAME="SEC2"><H2> CFR Section 2.</H2></A>
<P> Text of section 2 is here.
<BR>
<A HREF="#TOC">(TOC)</A>
<P>
<A NAME="SEC3"><H2> CFR Section 3.</H2></A>
<P> Text of section 3 is here.
<BR>
<A HREF="#TOC">(TOC)</A>
<P>
```

Seeing the (TOC) after each text section may seem strange at first, but your users quickly become accustomed to this method of intradocument linking. We'd recommend using this approach for longer, more complex documents or for a collection of related documents, but not for shorter pieces. An omnipresent TOC in a short document might seem obtrusive to your readers. (See Figure 9-5.)

Figure 9-5:
Table of
Contents
and text
links.

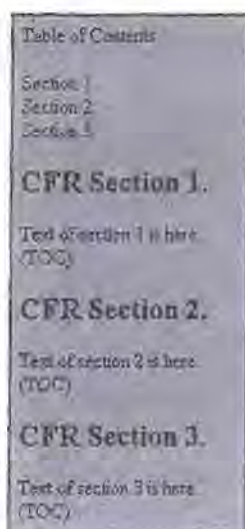


Table of Contents
Section 1
Section 2
Section 3

CFR Section 1.
Text of Section 1 is here.
(TOC)

CFR Section 2.
Text of section 2 is here.
(TOC)

CFR Section 3.
Text of section 3 is here.
(TOC)

You can use this same general method for links to anything within a single HTML document. It may look strange in the HTML document, but this is the way you create hypertext links within a text paragraph. Only use the "text" in each NAME="text" once per document, though, to keep the browser from becoming terminally confused. Otherwise, you never know where your users might wind up!



TIP

Named anchors should be named with text starting with a character from the set {a-z, A-Z}. They should never be exclusively numeric, like `<A NAME="12">blah</A>`. Anchors should also be uniquely named within the same document. These names are case sensitive, so that NAME="foo" and NAME="FOO" are different. So are these: NAME="Three Stooges" is not the same as NAME="ThreeStooges" is not the same as NAME="THREESTOOGES". You get the idea. . . .

Jumping to Remote Pages

Hypermedia links from text in your pages directly to other Web sites amazes and amuses your users. Although you can't create NAME="text" anchors in text at remote sites, you may be able to use anchors already in place there. If you find a linked TOC at another site, you can reference the same links that it uses.



Remember, if you can link to it by using your browser, you can copy the link into the text of your Web pages. Just make sure that you include the full URL in the `HREF="URL"`.

Hypertext links to outside resources

Links to Web sites outside of your own Web require fully qualified URLs, such as

```
URL: <A HREF = "http://www.outer.net/html4dum/html4dum.htm">
http://www.outer.net/html4dum/html4dum.htm</A><BR>
Text - Copyright &copy; 1995, 1996 Ed Tittel & Steve James.<BR>
Dummies Design and Art - Copyright &copy; 1995, 1996 IDG Books
Worldwide, Inc.<BR>
Web Layout - Copyright &copy; 1995, 1996
<A HREF="http://www.io.com/~mcintyre/lanwrght/
lanwrght.htm">LANWrights</A> &
<A HREF="http://impactonline.com"> IMPACT Online.</A><BR>
Revised -- January 15th, 1996 [JMS - IMPACT Online]<BR>
```

The three links (`HREF`) connect the *HTML For Dummies* home page to itself, to the LANWrights Web site, and to the IMPACT Online Web site. Why include the page's own URL as a link? It not only shows the user the URL, but it also allows direct linking if the user has saved the HTML source for this page to his or her own computer.

All these tags and links make for difficult reading unless you view them through a browser, as shown in Figure 9-6. Through the browser, hypertext words are shown in a different color, underlined, or both, depending on your browser's preference settings.

Figure 9-6:
External
links.

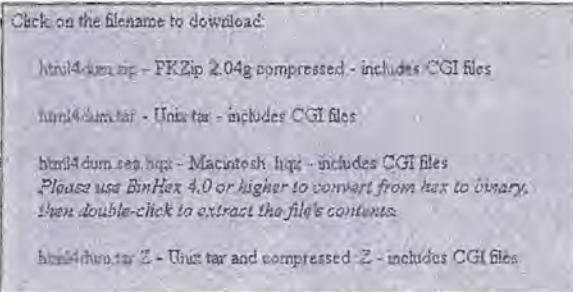
```
E-Mail: HTML for Dummies at html4dum@outer.net
URL: http://www.outer.net/html4dum/html4dum.htm
Text - Copyright © 1995, 1996 Ed Tittel & Steve James.
Dummies Design and Art - Copyright © 1995, 1996 IDG Books Worldwide, Inc.
Web Layout - Copyright © 1995, 1996 LANWrights & IMPACT Online
Revised -- January 15th, 1996 [JMS - IMPACT Online]
```

Jump pages

The term *jump page* refers to a Web page that contains a list of URLs to other Web pages, usually remote sites. You'll find that HTML list tags are invaluable tools that enable you to create visually pleasing and easily understood lists of links. Jump pages differ from basic Web pages only because the viewer primarily sees highlighted hyperlinks. This is appropriate for quick scanning, but not for general reading.

You can use icon images and spacer lines to visually separate sections of your list. You should carefully choose those words you use for each hyperlink; keep the main point of the information to which the link refers in mind. Figure 9-7 illustrates a portion of a jump page.

Figure 9-7:
A portion of
the *HTML*
For
Dummies
jump page.



When typing URLs for links, we strongly recommend that you first link to the destination and capture those URLs by using your browser. Then paste these URLs directly into your HTML file to cut down on typos and syntax errors.

A special <LINK>

The <LINK> tag provides information that links the current Web page to other Web pages or to other URL resources. When you want to be sure that your Web pages tell browsers and other WWW software about themselves, put a <LINK> in the <HEAD>...</HEAD> section. Chapter 5 shows several attributes that you may use in the <LINK> tag. If you start using one of the advanced HTML-generating programs, it may insert several <LINK> tags of various types within the head section of each page. The programs use these links to keep track of the pages themselves.

Perhaps NAME="text" is the most commonly used tag. It is used to provide an anchor from other locations. This named anchor is used for reference access from other locations or documents. Your HTML code should look like this:

```
<HTML>
<HEAD>
<TITLE> The Title of Your Page </TITLE>
<LINK NAME="My Home Page">
</HEAD>
<BODY>
<H1> The Heading of Your Page That Users See </H1>
and so on...
</BODY></HTML>
```

The Nesting Instinct: Lists Within Lists

When you create longer Web pages, you want to keep visual diversity high by using text formatting. However, because you are working with text, you have only headings, emphasized text (bold, strong, font size), and indented lists as tools. Lists within lists create the old, familiar outline form when displayed by most browsers.

The browser display in Figure 9-8 was created by the following HTML code. As you look through this HTML markup, remember that you're seeing only a fragment, not the whole thing:

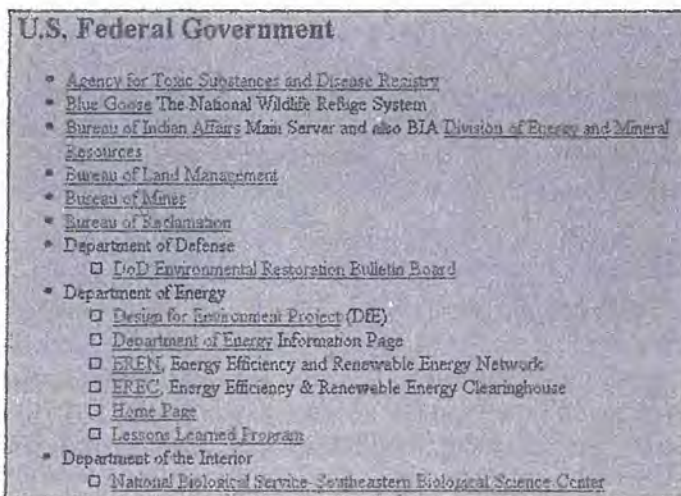


Figure 9-8:
Nested list
example.

```
<p><h2>U.S. Federal Government</h2>
<ul>
<li><a href="http://atsdr1.atsdr.cdc.gov:8080/atsdrhome.html">Agency
for
Toxic Substances and Disease Registry</a>
<li><A HREF="http://bluegoose.arw.r9.fws.gov/">Blue Goose</A> The
National Wildlife Refuge System
<li><A HREF="http://info.er.usgs.gov/doi/bureau-indian-
affairs.html">Bureau
of Indian Affairs</a> Main Server and also BIA <A
HREF="http://snake2.cr.usgs.gov/">Division of Energy and Mineral
Resources</a>
<li><A
HREF="http://info.er.usgs.gov/doi/bureau-land-management.html">Bureau
of
Land Management</a>
<li><A HREF="http://www.usbm.gov/">Bureau of Mines</a>
<li><A HREF="http://info.er.usgs.gov/doi/bureau-of-
reclamation.html">Bureau
of Reclamation</a>
```

```

<li>Department of Defense
<ul>
<li><a href="http://www.dtic.dla.mil/envirodod/envirodod.html">DoD
  Environmental Restoration Bulletin Board</a>
</ul>
<li>Department of Energy
<ul>
<li><a href="http://w3.pnl.gov:2080/DFE/home.html">Design for Environ-
  ment
  Project</a> (DfE)
<li><A HREF="http://web.fie.com/web/fed/doe/">Department of Energy</
  A>
  Information Page
<li><a href="http://www.eren.doe.gov/ee/ee.html">EREN</a>, Energy
  Efficiency and Renewable Energy Network
<li><a href="http://www.nciinc.com/~erec">EREC</a>, Energy Efficiency
  &
  Renewable Energy Clearinghouse
<li><a href="http://www.doe.gov">Home Page</a>
<li><a href="http://venus.hyperk.com/tr1/11/11.html">Lessons Learned
  Program</a>
</ul>
<li>Department of the Interior
<ul>
<li><a href="http://www.nfrcg.gov/">National Biological Service-
  Southeastern
  Biological Science Center</a>
</ul>
</ul>

```

Carefully track the list start () and end () tags. Directly under the U.S. Federal Government heading you see a start tag, and at the bottom of the listing you see its end tag. This placement of the tags indents and bullets all items that fall between them and are marked with the tags. This is a normal unordered list. What about all items that are indented a second time and preceded by a box rather than a bullet, you ask?

Each of these sections is contained within another pair of list tags. For example, immediately under the Department of Defense heading is the second level list start tag, and its end tag is immediately before the Department of Energy heading. The text between them, "DoD Environmental Restoration Bulletin Board," is marked with the tag, which causes the browser to indent it farther and place the box in front of it (in Netscape).

Some browsers keep track of the number of nests you use and change the bullets of each successive nesting to blocks or other symbols. It may be easier to visualize nested lists without the lines:

```
<UL> Start level 1.  
  <UL> Start level 2.  
    <UL> Start level 3.  
    </UL> End level 3.  
  </UL> End level 2.  
</UL> End level 1.
```

Nested lists are a good way to instruct a browser to indent certain lines of text without using the `<PRE> . . . </PRE>` or `<BLOCKQUOTE> . . . </BLOCKQUOTE>` tags. Along with the indentations, your readers will have to cope with either bullets or numbers, but that's fine for lists, as shown in the earlier example.

Browser developers are already working on giving browsers the ability to respond to HTML tag attributes for various types of symbols instead of bullets and blocks in lists. Who knows what they'll think of next?

Sampling Sophisticated Pages

Now it's time for a quick look at a couple of complex Web pages. We also encourage you to surf the Web for pages that strike your fancy. When you find one, view its source to see how the author worked the underlying magic. The Web is one of the few places where you can easily look behind the curtain to see how the illusion is created, so be sure to take full advantage of this opportunity.

The two following Web pages illustrate what you can accomplish if you use these tools and your imagination. You'll see a Web page screen (or two) viewed through Netscape, followed by the HTML markup for those screens. After each figure and HTML example, you'll see our comments on the techniques and coding used.

HTML For Dummies home page

Our own *HTML For Dummies* home page shows what you can do with HTML 2.0. It's eye-catching but not overdone (see Figure 9-9). The information is arranged nicely and gives the user multiple avenues of access. The image map and commands at the top and the buttons near the bottom make it easy to navigate quickly through the site. Users who want to "turn the pages" can use the arrow buttons to move linearly through the site. The "menu" page acts as a table of contents from which users can jump anywhere in the site. The "search" function lets users find anything contained in the site and jump directly there.

The entire home page is less than two screens long. It makes liberal use of color and blank space to keep the text readable. The image map graphic loads quickly because it's been designed with only a few colors and simple graphics components. The image map's functionality is repeated in the text immediately beneath it for users who don't have GUI browsers. The line "Click here for a non-imagemap version" displays an alternate home page (see Figure 9-10) for browsers that can't display image maps.

The footer contains all the requisite information including the page's own URL, which users will find handy should they capture the HTML source and later need to return to the online version. It's just a click away with the URL included as a hyperlink.

Figure 9-9:
HTML For Dummies
home page
using a CGI
image map.

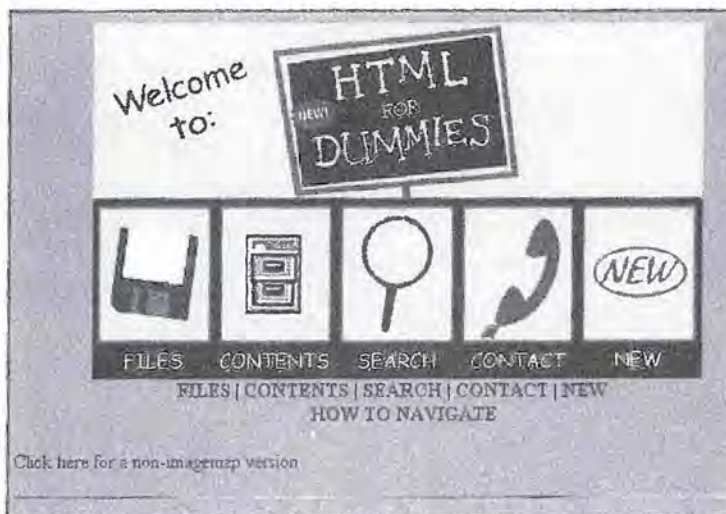
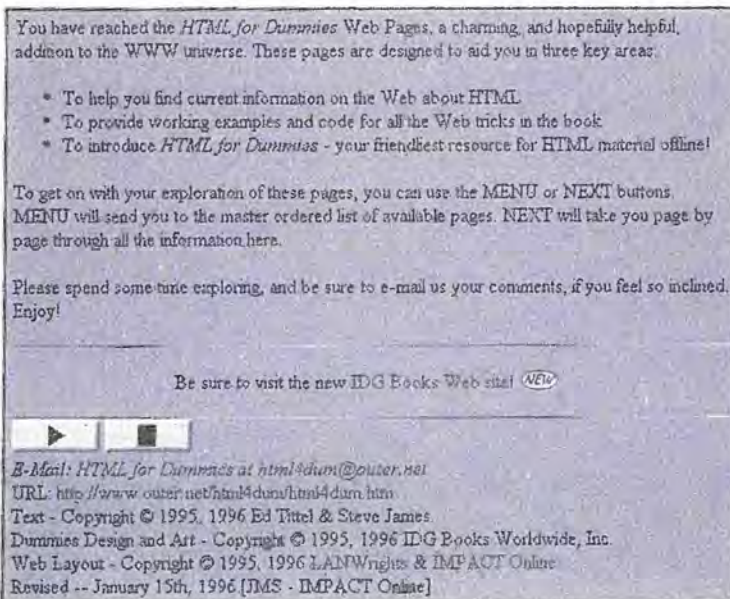


Figure 9-10:
HTML For Dummies
home page
without the
image map.



The HTML code for the *HTML For Dummies* home pages is HTML 2.0-compliant. It shows some interesting tricks that you may find useful. The use of the ALT line after the "line.gif" shows a divider even when the browser doesn't load images automatically. Using "WIDTH=100%" in the "line.gif" tag displays the rainbow separator line across the whole width of the browser's page, regardless of the monitor type or window size. To get the equivalent of a "tab" in HTML, the "space.gif" file is used in several places. This is merely an image with either the same color as the background or a transparent background GIF file. When used with the appropriate width and height parameters, it holds the space to the left of either text or an image, simulating the Tab key. It provides indentations without bullets like those produced by list tags.

```
<HTML>
<HEAD>
<TITLE> HTML For DUMMIES HomePage - IDG Books</TITLE>
<!--Turn a BASE HREF on if you have any problems with the way this
page behaves (e.g. unresolved graphics). Also be sure to provide
an absolute URL that points to the home page on your Web server,
e.g. BASE HREF="http://www.outer.net/html4dum/html4dum.htm"
This is the proper BASE reference for our own HTML For Dummies
server. -->
<BODY>
<A NAME="top"></A>
<IMG ALIGN=MIDDLE WIDTH=60 HEIGHT=0 SRC=graphics/space.gif
  ALT=" ">
<A HREF="http://www.io.com/~mcintyre/cgi-bin/ht4menu.map">
<IMG BORDER=0 ALIGN=TOP SRC="graphics/ht4menu.gif"
  ALT="Navigation Bar" ISMAP></A><P>
<IMG ALIGN=MIDDLE WIDTH=130 HEIGHT=0 SRC=graphics/space.gif
  ALT=" ">
<B><A HREF="ftpstuff.htm">FILES</A> &#32;&#124;
<A HREF="contents.htm">CONTENTS</A> &#32;&#124;
<A HREF="search4d.htm">SEARCH</A> &#32;&#124;
<A HREF="contact.htm">CONTACT</A> &#32;&#124;
<A HREF="whatsnew.htm">NEW</A><BR>
<IMG ALIGN=MIDDLE WIDTH=240 HEIGHT=0 SRC=graphics/space.gif
  ALT=" ">
<A HREF="navigate.htm">HOW TO NAVIGATE</A></B><P>
<A HREF="html4du2.htm">Click here for a non-imagemap version
</A><P>
<IMG WIDTH=100% SRC="graphics/line.gif"
  ALT="-----"><P>
You have reached the <I> HTML For Dummies</I> Web Pages, a
charming, and hopefully helpful, addition to the WWW universe.
These pages are designed to aid you in three key areas:
<UL>
```

```

<LI> To help you find current information on the Web about HTML
<LI> To provide working examples and code for all the Web tricks
    in the book
<LI> To introduce <I> HTML For Dummies</I> - your friendliest
    resource for HTML material offline!

```

```

</UL>

```

```

<P>

```

```

To get on with your exploration of these pages, you can use the
MENU or NEXT buttons. MENU will send you to the master ordered
list of available pages. NEXT will take you page by page through
all the information here.<P>

```

```

Please spend some time exploring, and be sure to e-mail us your
comments, if you feel so inclined. Enjoy!<P>

```

```

<IMG WIDTH=100% SRC="graphics/line.gif"

```

```

    ALT="-----"><BR>

```

```

<P>

```

```

<IMG HEIGHT=1 WIDTH=130 SRC="graphics/space.gif" ALT=" _ ">

```

```

Be sure to visit the new <A HREF="http://www.idgbooks.com">IDG
Books Web site</A>!

```

```

<IMG ALIGN=TOP SRC="graphics/new.gif"><P>

```

```

<P>

```

```

<IMG WIDTH=100% SRC="graphics/line.gif"

```

```

    ALT="-----"><BR>

```

```

<A HREF="contents.htm"><IMG SRC="graphics/next.gif" ALT="NEXT "
    BORDER=0></A>

```

```

<A HREF="search4d.htm#menu"><IMG SRC="graphics/menu.gif"
    ALT="MENU " BORDER=0></A><BR>

```

```

<ADDRESS>

```

```

E-Mail: <A HREF="mailto:html4dum@outer.net">

```

```

HTML For Dummies at html4dum@outer.net</A><BR>

```

```

</ADDRESS>

```

```

URL: <A HREF = "http://www.outer.net/html4dum/html4dum.htm">

```

```

http://www.outer.net/html4dum/html4dum.htm</A><BR>

```

```

Text - Copyright &copy; 1995, 1996 Ed Tittel & Steve James.

```

```

<BR>

```

```

Dummies Design and Art - Copyright &copy; 1995, 1996 IDG Books
Worldwide, Inc.<BR>

```

```

Web Layout - Copyright &copy; 1995, 1996

```

```

<A HREF="http://www.io.com/~mcintyre/lanwrght/lanwrght.htm">

```

```

LANWrights</A> &

```

```

<A HREF="http://impactonline.com"> IMPACT Online.</A><BR>

```

```

Revised - January 15th, 1996 [JMS - IMPACT Online]<BR>

```

```

</BODY>

```

```

</HTML>

```

As you can see in Figure 9-11, you don't have to use an image map to provide your users with nicely arranged, clickable images. This page differs only slightly in appearance from the primary home page and provides the same functionality.

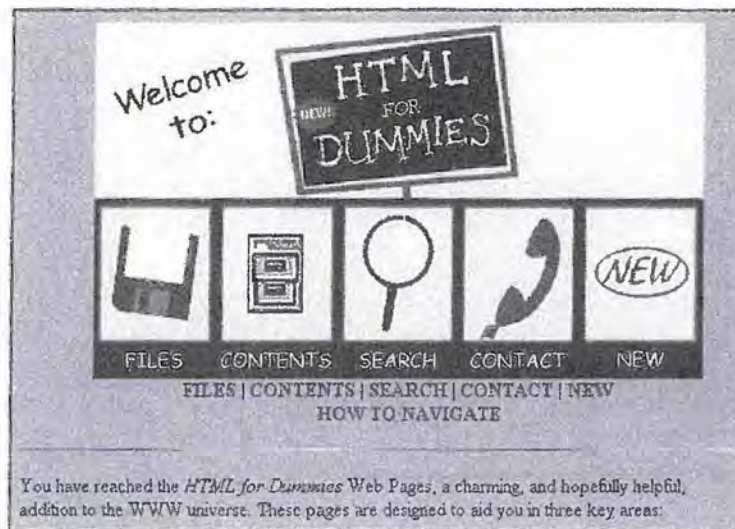


Figure 9-11:
Another
view of the
*HTML For
Dummies*.

The following HTML code shows the lines necessary to display the "Welcome to HTML For Dummies" graphic and the Files, Contents, Search, Contact, and New graphics below it as clickable links to files. If you don't want to deal with image maps and don't mind the separation between the images, this kind of presentation is easy to achieve.

```
<IMG ALIGN=MIDDLE WIDTH=60 HEIGHT=0
SRC=graphics/space.gif ALT=" ">
<A HREF="contents.htm"><IMG SRC=graphics/ht4logo.gif
BORDER=0 ALT="HTML For Dummies Logo"></A><BR>
<IMG ALIGN=MIDDLE WIDTH=60 HEIGHT=0
SRC=graphics/space.gif ALT=" ">
<A HREF="ftpstuff.htm"><IMG SRC="graphics/ht4filei.gif"
BORDER=0 ALT="FILES"></A>
<A HREF="contents.htm"><IMG SRC="graphics/ht4conti.gif"
BORDER=0 ALT="CONTENTS"></A>
<A HREF="search4d.htm"><IMG SRC="graphics/ht4seari.gif"
BORDER=0 ALT="SEARCH"></A>
<A HREF="contact.htm"><IMG SRC="graphics/ht4tacti.gif"
BORDER=0 ALT="CONTACT"></A>
<A HREF="wahtsnew.htm"><IMG SRC="graphics/ht4newi.gif"
BORDER=0 ALT="NEW"></A>
<B><A HREF="ftpstuff.htm">FILES</A> &#32;&#124;
<A HREF="contents.htm">CONTENTS</A> &#32;&#124;
<A HREF="search4d.htm">SEARCH</A> &#32;&#124;
<A HREF="contact.htm">CONTACT</A> &#32;&#124;
<A HREF="wahtsnew.htm">NEW</A>
</B>
<IMG ALIGN=MIDDLE WIDTH=130 HEIGHT=0
SRC=graphics/space.gif ALT=" ">
```

```

<A HREF="contents.htm">CONTENTS</A> &#32;&#124;
<A HREF="search4d.htm">SEARCH</A> &#32;&#124;
<A HREF="contact.htm">CONTACT</A> &#32;&#124;
<A HREF="whatsnew.htm">NEW</A><BR>
<IMG ALIGN=MIDDLE WIDTH=240 HEIGHT=0 SRC=graphics/space.gif ALT=" ">
<A HREF="navigate.htm">HOW TO NAVIGATE</A></B><P>
<IMG WIDTH=100% SRC="graphics/line.gif"
  ALT="-----"><P>

```

Shockwave Productions HTML page

The *Shockwave Productions* HTML page featured in Figure 9-12 shows the magic that can be performed on the Web by using currently available, leading-edge tools. This page illustrates the use of HTML 3.0, Netscape extensions, Java applets, and more. It should give you an idea of the nearly limitless possibilities for your own Web site.

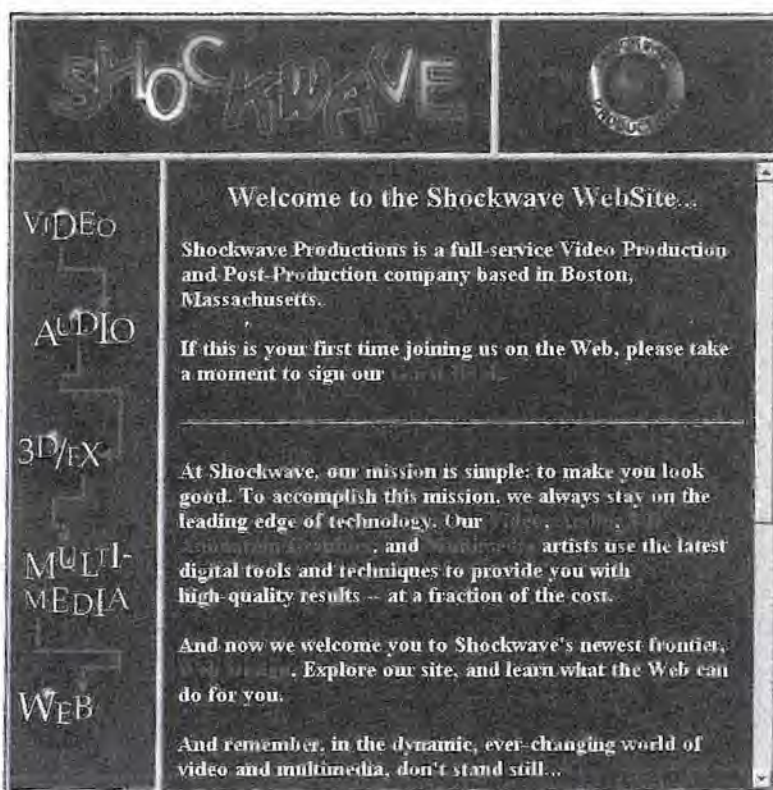


Figure 9-12:
Shockwave
Productions
HTML page.

The view shown below was captured using Netscape 2.0b4 for Windows 95 (the 32-bit version), which allows the Java applet that controls the Shockwave logo in the upper-right frame to flip like a tossed coin. As you'll see, the page also makes use of Netscape's frames, with four of them on the same page. You

can go to different sections of the Web site by clicking on the word in the left frame (Video, Audio, 3D/FX, Multimedia, or Web). Doing so displays the section's pages in the main frame (bottom right). Both of the bottom frames contain scroll bars that move only the information within the frame. This allows the SHOCKWAVE name in the upper-left frame and the flipping logo in the upper-right frame to remain visible while you browse the site.

The following HTML code shown for this page isn't as complex as you may think. However, the work that went into creating the images was considerable. Although it looks like one large graphic, the lower left frame is composed of separate GIF files for each "button," created to join seamlessly with one another.

The call to the Java applet is simple in the following code, but the applet itself is much more complicated. We hope that you have a 32-bit Java-enabled browser to view the flipping Shockwave logo. Although it isn't shown in Figure 9-12, the page also includes an e-mail address and `mailto:` line at the bottom.

This page shows how frames, tables, borders, and background colors can enhance your Web page, if you know how to artistically intertwine them. However, as the warning goes, "Don't try this at home without proper training and practice." If you're dying to try this stuff out, be prepared to spend some serious time researching and learning these topics and technologies!

```
<HTML>
<HEAD>
<TITLE>Shockwave Productions</TITLE>
<FRAMESET ROWS="115,*">
<FRAMESET COLS="400,*" SCROLLING="no">
<FRAME SRC="htitle.html" NAME="title" SCROLLING="no" MARGINHEIGHT="0"
    NORESIZE>
<FRAME SRC="javaplanet.html" NAME="java" SCROLLING="no"
    MARGINHEIGHT="0" NORESIZE>
</FRAMESET>
<FRAMESET COLS="130,*">
<FRAME SRC="hoptions.html" NAME="options" MARGINWIDTH="1"
    NORESIZE>
<FRAME SRC="hmain.html" NAME="main">
</FRAMESET>
</FRAMESET>
<NOFRAMES>
</HEAD>
<BODY BACKGROUND="images/background.gif" TEXT="#FFFFFF"
    LINK="#9F0F9F"
    ALINK="#FF0000" VLINK="#FF0000">
<IMG SRC="images/SWtitle.gif" BORDER=0 ALT="[SHOCKWAVE]">
<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0>
<TR>
```

```

<TD ALIGN=LEFT VALIGN=TOP WIDTH=128>
<IMG SRC="images/buffer.gif" border=0 width=128 height = 1>
<A HREF = "N1/video/videowe1N1.html"><IMG border=0 width=100
height=100 SRC="images/videobutt.gif" ALT="[VIDEO]"></A>
<A HREF = "N1/audio/audiowe1N1.html"><IMG border=0 width=100
height=100 SRC="images/audiobutt.gif" ALT="[AUDIO]"></A>
<A HREF = "N1/grx/grxwe1N1.html"><IMG border=0 width=100
height=100 SRC="images/3dfxbutt.gif" ALT="[3DFX]"></A>
<A HREF = "N1/media/mediawe1N1.html"><IMG border=0 width=100
height=100 SRC="images/mmbutt.gif" ALT="[MULTI-MEDIA]"></A>
<A HREF = "N1/web/webwe1N1.html"><IMG border=0 width=100
height=100 SRC="images/webbutt.gif" ALT="[WEB]"></A>
<IMG SRC="images/buffer.gif" BORDER=0 WIDTH=128 HEIGHT=1>
</TD ALIGN=CENTER>
<TD VALIGN=TOP>
<HR>
<P>
<CENTER>
<H2>Welcome to the Shockwave WebSite...</H2>
<H4><P></CENTER><FONT SIZE=4>
Shockwave Productions is a full-service Video Production and Post-
Production company based in Boston, Massachusetts.<P>If this is
your first time joining us on the Web, please take a moment to
sign our <A HREF="guest/guestN1.html">Guest Book</A>.<P><HR><P>
At Shockwave, our mission is simple: to make you look good.
To accomplish this mission, we always stay on the leading edge of
technology. Our <A HREF="N1/video/videowe1N1.html">Video</A>, <A
HREF="N1/audio/audiowe1N1.html">Audio</A>,
<A HREF="N1/grx/grxwe1N1.html">3-D Animation/Graphics</A>, and
<A HREF="N1/media/mediawe1N1.html">Multimedia</A> artists use the
latest digital tools and techniques to provide you with high-
quality results — at a fraction of the cost.<P>
And now we welcome you to Shockwave's newest frontier, <A HREF="N1/
web/webwe1N1.html">Web Design</A>. Explore our site,
and learn what the Web can do for you.<P>
And remember, in the dynamic, ever-changing world of video and
multimedia, don't stand still...<BR><P>
<CENTER><IMG SRC="images/catch.gif"
ALT="[Catch the Wave!]"><P></CENTER>
<HR><P>
Coming soon! Visit our <A HREF="../vrml/vrmlwe1.shtm">VRML</A>
site!<P>
Check here for more innovations as our new web site becomes fully
active in the next few weeks. Special thanks to all who gave us
suggestions, and the many, many people who visited our old site.

```

```
<P>
Enjoy your visit with Shockwave!<P><HR><P>
<CENTER><A HREF="mailto:info@swave.com">
<IMG SRC="images/address.gif" BORDER=0 ALT="[info@swave.com]">
</A></CENTER>
</TD>
</H4>
</TR>
</TABLE>
</CENTER>
</BODY>
</NOFRAMES>
</HTML>
```

Really Cool: Java, VRML, Shockwave

Okay! Okay! Stop throwing things. You're not alone. Everyone wants their Web pages to be as magical as the Shockwave Productions page. Take your time, learn the basics of HTML, and then progress to using the really cool tools, and your pages can look great, too!

So what are these really cool tools? Because you twisted our arms (ouch!), we'll give you a hint here, even though it's beyond the scope of this book. Keep in mind that, although these tools are available now, you'll be on the cutting edge of technology if you start using them. That's a position similar to Wile E. Coyote after he runs off the mesa top after the Roadrunner. He's hanging in midair but not falling (yet). So if you run right out and try to grab these futuristic tools ... don't look down.

Java

According to Sun Microsystems, Java is "a simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded, and dynamic language." Yeah, right??

Well, it's a programming language that's somewhat based on C++ but is supposed to be much easier to use. It's used to produce small programs, called *applets*, that are distributed on the Web along with HTML documents. Your browser quickly downloads applets to your computer, where they run to produce sound, video, special effects, or just about anything that your computer and browser are capable of doing. In the near future, many Java applets will be available for you to use in your Web pages in a manner similar to the

way you use GIFs and JPEGs now, except applets actually do things. Check out Java's Web site for more information. (WARNING: Java's not expected to be commercially released until mid-1996!)



<http://www.javasoft.com/>

VRML

The Virtual Reality Modeling Language (VRML) is a language for describing multiparticipant interactive simulations — virtual worlds networked via the global Internet and hyperlinked with the World Wide Web. What it will be in the future depends on the limits of your imagination. In real-world terms, it provides a way to add 3-D data to the Web. VRML browsers are already available, and a few Web sites contain VRML files. How fast it grows will depend primarily on the cost of the hardware that replaces your 2-D monitor with 3-D goggles. They're available now, but they are costly and aren't quite ready for prime-time use. Everything that you ever wanted to know about VRML and more can be found at



<http://www.vrml.org/>

Shockwave

Shockwave for Director is the name of the Macromedia product and technology that was designed to deliver Macromedia Director files over the Internet. Shockwave for Director includes the following elements:

- ✓ The Shockwave plug-in for Web browsers, like Netscape Navigator 2.0, allows movies to be played seamlessly within the same window as the browser page.
- ✓ Afterburner is a post-processor program for Director movie source files. Multimedia developers use it to prepare content for Internet distribution. Afterburner compresses movies and makes them ready to upload to an HTTP server, where they can be accessed by Internet users with Shockwave-enabled Web browsers.

These are both rather complicated programming systems, primarily useful to professionals in the multimedia field. Undoubtedly, Shockwave files will be showing up on the commercial Web sites in the near future. They may even be available at sites, much as GIF and JPEG files are currently available. Only time and the market will tell. Check out Shockwave for Director at



<http://www.macromedia.com/Tools/Shockwave/index.html>

Congratulations! You've now gone far enough (or too far?) into HTML to know where home is and how to get back there from the North Forty. Now that you really know your way around a Web site, you're ready to link to the realm of HTML forms. This is where you invite your users talk back to you and give them an easy way to do so.

Chapter 10

Strictly Pro Forma: Using Forms for Feedback

In This Chapter

- Learning what forms are for
 - Dealing with browser and server limitations
 - Learning about form tags
 - Using form tags
 - Forming good attitudes
 - Formulating good layouts
-

When all the pieces come together properly, it's easy to see how the Web brings people or organizations together. At first glance, the Web might look pretty much like a one-way street — that is, an environment where WebMasters communicate aplenty with Web users, with not much interaction between the two. But it doesn't have to be that way.

What HTML Forms Are For

The essence of serving up useful information is relevancy and immediacy. But the best judge of the quality of your information is your audience. Wouldn't it be wonderful if your readers could give you feedback on your Web pages? Then they could tell you what parts they like, what they don't like, and what other things they'd like to see included in your site.

This is where HTML forms come into the picture. Up to this point in the book, we've talked about all of the basics — and even a few advanced techniques — for communicating with your audience. In this chapter, you'll learn how to turn the tables and create HTML text that lets your audience communicate with you!

As it turns out, HTML supports a rich variety of input capabilities to let you solicit feedback. In the pages that follow, you find out about the tags to use, the controls and inputs they enable, and some layout considerations for building forms. You also see some interesting example forms that can help you understand what HTML forms look like and how they behave.

Living within Your Forms Limitations

By the time you read this, forms support in HTML, which you recall from Chapter 4 is a hallmark of HTML 2.0, is final and more or less finished in its current form. This means that the underlying HTML markup for forms will only change if the underlying standard changes. But because forms have been supported for only about a year, they still qualify as *new-and-untried* instead of *old-and-familiar* for many Web users. Nevertheless, we think that working with forms is becoming totally commonplace, so you should become familiar with them and their capabilities to see if you can use them within your Web pages.

Beware of browser!

Although most of the “hot” browsers — Mosaic and its variants, Netscape, Internet Explorer, and the like — already include forms support, other browsers may not. In fact, you won’t know how well your favorite browser handles forms until you start testing your forms tags against it. If you follow our suggestions and test your pages against multiple browsers, you can immediately observe different levels of robustness and capability when it comes to forms implementations.

The bottom line is that not all browsers support forms equally, but that support is pretty commonplace. By the time you read this, support should no longer be a problem — at least for users with current browser versions. But for older browsers, don’t be surprised if forms don’t work that well, or if they don’t work at all.



Assuming that the information your form solicits is important to you, consider adding an FTP URL to your page to let users download a file containing a text-only version of the form’s content. Then they can download the file and complete it by using any text editor. If you include an e-mail address inside the file, they can send it back to you, and you’ll get feedback (even from users who can’t view your forms). That way nobody gets left out!

Sorry, servers . . .

Because the Web is a client/server environment, you should also be aware that just because your browser supports forms doesn't automatically imply that the server installed at your site can handle them. Unfortunately, keeping up with HTML advancements means that Web servers have to change right along with the clients. In other words, your server may not support the input-handling programs necessary to process a form's input when it gets delivered.

However, there is a silver lining in this potentially dark cloud: The most common implementations of the *httpd* server come from NCSA and CERN and run in the UNIX environment. Both of these groups have standardized their forms-handling technology and offer useful, robust forms-handling capability.

These implementations are so common, in fact, that we assume that your Web server works the way that they do. This means that you may have to alter some of the approaches to CGI scripting and other programming that you might use to handle forms on your server. If you're not using UNIX and the NCSA or CERN implementations of *httpd*, you should investigate the particulars that your server's implementation requires and alter our instructions accordingly.

What's in a Form?

When adding forms support to a Web page, you include special tags that let you solicit input from users. You surround these tags with text that prompts user responses. You also include tags that gather the input and ship it to your Web server or to other servers that may offer services — like Gopher or Archie — that your form knows how to query. Here's how the process works:

- ✓ On a particular Web page, you include tags to set up a form and to solicit input from users. Some of your users will work their way through this material and provide the information that you want. This essentially amounts to filling out the form that you've supplied.
- ✓ After a form has been filled out, users can then direct their input to a program running on the Web server that delivered the form. In most cases, they need to select a particular control, called `SUBMIT`, that gathers the information and sends it to the proper destination on your Web server.
- ✓ Assuming that the program is available for use (installed and running properly, that is), it accepts the input information. Then the program decodes and interprets the contents for further action.

- ✓ After the input is received and interpreted, the program can do pretty much anything it wants. In practice, this boils down to recognizing key elements in a form's content and custom-building an HTML document in response. Building a document isn't required, but it is a pretty commonplace capability within forms-handling programs.
- ✓ The custom-built document is then delivered to the user in response to the form's content. At this point, additional interaction can occur (if the "return page" includes another form), requested information can be delivered (in response to requests on the form), and so on.

The information collected from a form can be 1) written to a file, 2) submitted to a database such as Informix or Oracle, or 3) e-mailed to someone in particular. Forms also allow a user to participate in building a Web document such as on one Web site that allows users to dictate how a story is played out; in this case, the users collectively determine the outcome.

Thus, forms not only provide communication from users to servers, but also provide for ongoing interaction between users and servers. This is pretty powerful stuff and adds a lot of value to your Web pages.

Forms involve two-way communication

The input-catching programs on your server rely on an interface between Web browsers and servers called the *Common Gateway Interface* (CGI). This interface codifies how browsers can send information back to servers. It sets up the formatting for the user-supplied input information, so that forms-handling programs know what to expect and how to deal with what they receive.

The `ACTION` attribute in the `<FORM>` tag specifies a URL that indicates a specific CGI script or program that collects and munges the form data that users enter. Likewise, the `METHOD` tag describes the way in which input data is delivered to a forms-handling program.



In this chapter, we concentrate on the input side of HTML forms — that is, you find out how to build forms. This is a pure exercise in building the front end of a form — that is, the part the users see. You won't learn how to build the back end — that is, how to build CGI or equivalent programs so that your server can deal with forms input — until Chapter 13. Not to worry, however — there's plenty of interesting front-end material to master here!

Tag! You're a form . . .



HTML includes several different classes of forms tags (for the details on syntax and usage, please consult Chapter 5). To begin with, all HTML forms occur within the `<FORM> . . . </FORM>` tags. The `<FORM>` tag also includes attributes that specify where and how to deliver input to the appropriate Web server.

Within the `<FORM> . . . </FORM>` tags, all other forms-related tags and text must appear. These tags include methods for

- ✓ specifying input (the `<INPUT>` tag and its many attributes).
- ✓ setting up text input areas (the `<TEXTAREA> . . . </TEXTAREA>` tags).
- ✓ selecting values from a predefined set of possible inputs (the `<SELECT> . . . </SELECT>` tags).
- ✓ managing the form's content (using the `SUBMIT` attribute for `INPUT` to deliver the content to the server, or the `RESET` attribute to clear its contents, and start over).

Forms input tags support multiple ways to interact with users, including

- ✓ creating text input fields, where users can type in whatever they want.
- ✓ generating pull-down menus, often called "pick lists" because they require making one or more selections from a set of predefined choices.
- ✓ creating labeled check boxes or radio buttons on-screen, which users can select to indicate choices. Check boxes allow multiple selections, and radio buttons allow just one selection.

This may not sound like much, but when you combine it with the ability to prompt users for input with surrounding text, it provides a surprisingly powerful way to ask for information right on a Web page. Thus, the real answer to the question at the head of this section: "What's in a Form?" has to be: "Almost anything you want to put there!"

The remainder of this chapter steps you through all the details of building a form, so you can use the capabilities that we just described.

Using Form Tags

To start out, you should set up your `<FORM>` environment to build a form within a Web page. It's okay to add a form to an existing HTML file or to build a separate one just to contain your form. We recommend that you add shorter forms (half a screen or less) to existing files but that you create new files for forms that are longer than half a screen.

Setting the `<FORM>` environment

The two key attributes within the `<FORM>` tag are `METHOD` and `ACTION`. Together, these attributes control how information is sent to the Web server and determine which input-handling program will receive the form's contents.

There's no rhythm to `METHOD`

The `METHOD` indicates how the information is sent to the server when the form is submitted. `METHOD` can take one of two possible values: `POST` or `GET`.

Of these two methods, we prefer `POST` because it causes a form's contents to be parsed one element at a time. `GET`, on the other hand, it concatenates all the field names and their associated values into one long string. Because UNIX (and most systems) has a limit on how long a single string can be (for UNIX it's 255 characters), it's not hard to imagine that some of the information might get lost when its contents are truncated.

That's why you see us use `POST` as our only `METHOD` for submitting forms in this book. That's also why you should do the same, unless you're dead certain that the number of characters in a form will never, ever exceed 255.

Lights, camera... `ACTION`

`ACTION` supplies the URL for the CGI script or other input-handling program on the server that will receive a form's input. The URL can be a full specification or simply a relative reference. Either way, you need to make sure that it points at the right program, in the right location, to do the job that's expected. You also need to make sure that the CGI script or program is executable and that it behaves properly. You'll hear a lot more about this in Chapter 15, which gets into the ins and outs of testing your HTML documents and their related CGI programs.

Let's make an assumption...

Because you won't have to worry about handling input until Chapter 13, we follow two conventions for all the syntax in this chapter:

- ✓ In every `<FORM>` tag, `METHOD="POST"`.
- ✓ For every `ACTION`, `URL="/cgi/form-name"` where we replace the placeholder *form-name* with the actual name of the form under discussion (that is, for the form named `get-inf.html`, `URL="/cgi-bin/get-inf"`).

These conventions make it easy to create sample HTML files to implement the forms in this chapter (you can also find these examples on the disk that comes with this book).



Knowing what's (in)coming: the <INPUT> tags

The `<INPUT>` tag defines a basic form element. It takes at least two attributes, namely `TYPE` and `NAME`. `TYPE` indicates what kind of element should be displayed on the form. `NAME` assigns a name to go with the input field or value that corresponds to the `<INPUT>` tag.

`NAME` is used to identify the contents of a field in the form information that ultimately uploads to the input-handling Web server. In fact, what the server receives is a series of `name-value` pairs. The name that identifies the value is the string supplied in the `NAME="string"` attribute, and the value is what the user enters or selects for that particular field. Read on — you'll see an example in the next section that should make all this clear!

TYPE-casting still works!

The `TYPE` attribute can take any of the following values:

- ✓ **CHECKBOX**: Produces an on-screen check box for users to make multiple selections.
- ✓ **HIDDEN**: Produces no visible input area; use this to pass data needed for other uses through the form. For example, this might be an ongoing series of forms based on an earlier interaction during which the user identifies him- or herself — a **HIDDEN** field contains the `name-value` pair for that data, but it doesn't show it on the current form. (Some browsers display these fields, but at the *bottom* of a form, and each field has no accompanying label, as with NetManage's WebSurfer browser.)
- ✓ **IMAGE**: Lets you designate a graphic as a selectable item in a form. You can use this to include icons or other graphical symbols.
- ✓ **RADIO**: Creates a radio button for a range of selections, from which the user may select only one.
- ✓ **RESET**: Creates a button labeled "reset" in your form. Include this so that users can clear a form's contents and start over. Be sure to place it well away from other controls — you don't want them to clear the form by accident!
- ✓ **SUBMIT**: Creates a button labeled "submit" (by default, or whatever value you supply for the `VALUE` attribute for **SUBMIT**) in your form. The type **SUBMIT** tells the browser to bundle the form data and pass it all to the CGI script indicated by the `ACTION` attribute. In plain English (remember that?) **SUBMIT** is the button used to send in the filled-out form, so a form is useless without an `<INPUT>` field of type **SUBMIT**.

- ✓ **TEXT**: Provides a one-line area for text entry. Use this for short fields only (as in the example that follows). For longer text fields, use the `<TEXTAREA> . . . </TEXTAREA>` tags instead.

These **TYPE** attribute values provide a wide range of input displays and data types for form input. As you look at HTML forms on the Web and in this book with a new (and more trained) eye, you can see how effectively these types can be used.

Other <INPUT> attributes

Most of the remaining attributes exist to modify or qualify the `<INPUT>` attribute with **TEXT** type as the default. Following is a quick review to remind you of what we covered in Chapter 5:

- ✓ **VALUE="value"**: Supplies a default value for a **TEXT** or **HIDDEN** element or supplies the corresponding value for a radio button or check box selection. This can be used to determine the label of a submit or a reset button, like **VALUE="Submit to Admin"** for a submit or **VALUE="Clear Form"** for a reset.
- ✓ **SRC="URL"**: Provides a pointer to the graphic for an **IMAGE**.
- ✓ **CHECKED**: Makes sure that a certain radio button or check box is checked when the form is either visited for the first time or when it is reset. You can control default settings with the **CHECKED** attribute of `<INPUT>`.
- ✓ **SIZE="number"**: Indicates the number of characters that a **TEXT** element can display without scrolling.
- ✓ **MAXLENGTH="number"**: Sets the maximum number of characters that a value in a **TEXT** element can contain.
- ✓ **ALIGN=(TOP|MIDDLE|BOTTOM)**: For **IMAGE** elements, determines how the graphic is aligned on the form, vis-à-vis the accompanying text.

A TEXT-oriented <INPUT> example

That's it for the `<INPUT>` tag. Following is a look at a relatively simple example of a survey form:

```
<HTML>
<HEAD>
<TITLE>Reader Contact Information</TITLE>
<!-- the name of this form is usr-inf.html -->
</HEAD>
<BODY>
<H3>Reader Contact Information</H3>
```

```
<P>Please fill out this form, so we'll know how to get in
touch with you. Thanks!
<FORM METHOD="POST" ACTION="/cgi/usr-inf">
<P>Please enter your name:
<P>First name: <INPUT NAME="first" TYPE="TEXT" SIZE="12"
MAXLENGTH="20">
MI: <INPUT NAME="MI" TYPE="TEXT" SIZE="3" MAXLENGTH="3">
Surname: <INPUT NAME="surname" TYPE="TEXT" SIZE="15"
MAXLENGTH="25">
<P>
<P>Please give us your mailing address:
<P>Address 1: <INPUT NAME="adr1" TYPE="TEXT" SIZE="30"
MAXLENGTH="45">
<P>Address 2: <INPUT NAME="adr2" TYPE="TEXT" SIZE="30"
MAXLENGTH="45">
<P>City: <INPUT NAME="city" TYPE="TEXT" SIZE="15"
MAXLENGTH="30">
<P>State: <INPUT NAME="state" TYPE="TEXT" SIZE="15" MAXLENGTH="15">
ZIP&#47;Postal Code: <INPUT NAME="zip" TYPE="TEXT" SIZE="10"
MAXLENGTH="10">
<P>Country: <INPUT NAME="country" TYPE="TEXT" SIZE="15"
MAXLENGTH="15">
<P>
<P>Thank you! <INPUT TYPE="SUBMIT"> <INPUT TYPE="RESET">
</FORM>
<ADDRESS>
Sample form for <I>HTML For Dummies</I> Version 2.2
1//05/96 http://www.noplace.com/HTML4D/usr-inf.html
</ADDRESS>
</BODY></HTML>
```

Figure 10-1 shows this HTML form on Netscape. Note the positions of the one-line text boxes immediately after the field names, and note the ability to set these boxes on individual lines (as with Address1 and Address2) or together (as with First name, middle initial (MI), and last name (Surname)). These options make it easy to build simple, usable forms.

Figure 10-1:
The Reader
Contact
Information
form
on-screen.

Reader Contact Information

Please fill out this form, so we'll know how to get in touch with you. Thank!

Please enter your name:

First name: MI: Surname:

Please give us your mailing address:

Address 1:

Address 2:

City:

State: ZIP/Postal Code:

Country:

Thank you!

Sample form for HTML for Dummies, Version 2.2, 1/05/96 <http://www.nagel.com/HTML/Editor-01.html>

Being *<SELECT>*ive

The `<SELECT> . . . </SELECT>` pair works much like a list style, except that it builds a pickable list of `<OPTION>` elements instead of the `<LI>` list items. Within the `<SELECT>` tag, the following attributes can occur:

- ✓ **NAME="text"**: Provides the name that is passed to the server as the identifying portion of the name-value pair for this element.
- ✓ **SIZE="number"**: Controls the number of elements that the pick list displays; although you can still define more than this many elements, it keeps the size of the list more manageable on-screen.
- ✓ **MULTIPLE**: Indicates that multiple selections from a list are possible; if this flag isn't present in a `<SELECT>` statement, your users can select only a single element from the pick list.

There's really not that much work involved in building a `<SELECT>` field for your form. In the following example, you see how easy it is to construct a list of spices from which a user can select and order:

```
<HTML>
<HEAD>
<TITLE>&lt;SELECT&gt; Spices</TITLE>
  <!-- the name of this form is sel-spi.html -->
</HEAD>
<BODY>
  <H3>This Month's Spicy Selections!</H3>
```

```

<P>Spice up your life. Order from this
month's special selections.<BR> All items
include 2 oz. of the finest condiments,
packed in tinted glass bottles for best
storage.

<HR>
<FORM METHOD="POST" ACTION="/cgi/sel-spi">

<P>Pepper Selections:
<SELECT NAME="pepper" SIZE="4" MULTIPLE>
<OPTION>Plain-black
<OPTION>Malabar
<OPTION>Telicherry
<OPTION>Green-dried
<OPTION>Green-pickled
<OPTION>Red
<OPTION>White
</SELECT>
<P>
Please pick a button to indicate how the pepper<BR>
should be delivered:<BR>
Ground <INPUT TYPE="RADIO" NAME="grind" VALUE="ground"> <BR>
Whole <INPUT TYPE="RADIO" NAME="grind" VALUE="whole"> <BR>
<P>
<HR>
<P>Imported and Domestic Oregano:
<SELECT NAME="oregano" SIZE="4" MULTIPLE>
<OPTION> Italian-whole
<OPTION> Italian-crumbled
<OPTION> Greek-whole
<OPTION> Indian
<OPTION> Mexican
<OPTION> Organic-California
</SELECT>
<P>Thanks for your order! <INPUT TYPE="SUBMIT" VALUE="Send Order">
<INPUT TYPE="RESET">
</FORM>
<ADDRESS>
Sample form for <I>HTML For Dummies</I> Version 2.2
1/5/96 http://www.noplace.com/HTML4D/usr-inf.html</ADDRESS></
BODY></HTML>

```

Figure 10-2 shows what nice results you can get from using `<SELECT>` elements to provide options for your users to pick from. Notice also the radio buttons that enable users to specify whether they want whole or ground pepper. By giving both radio buttons the same NAME, we indicate that only one option can be chosen.

Figure 10-2:
The
`<SELECT>`
tag creates
scrolling
pick lists of
choices for
users to
select.

This Month's Spicy Selections!

Spice up your life. Order from this month's special selections.
All items include 2 oz. of the finest ingredients, packed in tinted glass bottles for best storage.

Pepper Selections:

- Plain-black
- Malabar
- Tellicherry
- Green-dried

Please pick a button to indicate how the pepper should be delivered:

Ground ☐

Whole ☐

Imported and Domestic Oregano:

- Italian-whole
- Italian-crumbled
- Greek-whole
- Indian

Thanked For your order!

`<TEXTAREA>` lets users wax eloquent... or profane!

The `<TEXTAREA>` . . . `</TEXTAREA>` tags let you create input elements of more or less arbitrary size on a form. Any text that appears between the opening and closing tags will be displayed within the text area on-screen (and if left unaltered, supplies the default value delivered by the form).

`<TEXTAREA>` takes three important attributes:

- ✓ `NAME="text"`: Provides the identifier part of the all-important name-value pair delivered to the server.
- ✓ `ROWS="number"`: Specifies the number of lines of text that the text area will contain.
- ✓ `COLS="number"`: Specifies the number of characters that can fit onto any one row of the text area; this value also sets the width of the text area on-screen.

The example that follows shows how a text area is used to provide space for free-form feedback or information as part of a survey-style form:

```

<HTML>
<HEAD>
<TITLE>&lt;TEXTAREA&gt; On Display</TITLE>
  <!-- the name of this form is txt-ara.html -->
</HEAD>
  <BODY>
    <H3>The Widget Waffle Iron Survey</H3>
    <P>Please fill out the following information so that we
      can register your new Widget Waffle Iron.
    <HR>
    <FORM METHOD="POST" ACTION="/cgi/txt-ara">
    <P>Model Number
      <SELECT NAME="mod-num" SIZE="3">
        <OPTION>102 (Single Belgian)
        <OPTION>103 (Double Belgian)
        <OPTION>104 (Single Heart-shaped)
        <OPTION>105 (Double Heart-shaped)
        <OPTION>204 (Restaurant Waffler)
        <OPTION>297 (Cone Waffler)
      </SELECT>
    <HR>
    <B>Please complete the following purchase information:</B><BR>
    <P>Serial number: <INPUT NAME="snum" TYPE="TEXT" SIZE="10"
      MAXLENGTH="10">
    <P>Purchase Price: <INPUT NAME="price" TYPE="TEXT" SIZE="6"
      MAXLENGTH="10">
    <P>Location: <INPUT NAME="location" TYPE="TEXT" SIZE="15"
      MAXLENGTH="30">
    <HR>
    <B>Please tell us about yourself:</B>

    <P>Male <INPUT NAME="sex" TYPE="CHECKBOX" VALUE="male">
      Female <INPUT NAME="sex" TYPE="CHECKBOX" VALUE="female">
    <P>Age:
      under 25 <INPUT NAME="age" TYPE="CHECKBOX" VALUE="lo">
      25-50 <INPUT NAME="age" TYPE="CHECKBOX" VALUE="med">
      over 50 <INPUT NAME="age" TYPE="CHECKBOX" VALUE="hi">
    <P>
    <HR>
    Please share your favorite waffle recipe with us. If we like
    it, we'll include it in our next Widget Waffler cookbook!
    Here's an example to inspire you.

```

(continued)

```

<P><TEXTAREA NAME="recipe" ROWS="10" COLS="65">
    Banana Waffles
    Ingredients:
    2 c. waffle batter (see Widget Waffler cookbook for recipe)
    2 ripe bananas, peeled, sliced 1/4" thick
    1 tsp. cinnamon
    Preparation:
    Mix ingredients together.
    Preheat Widget Waffler (wait 'til light goes off).
    Pour 1/2 c. batter in Waffler (wait 'til light goes off).
    Keep browned waffles warm in oven until ready to serve.
</TEXTAREA>
<P>Thank you! <INPUT TYPE="SUBMIT" VALUE="Register now">
<INPUT TYPE="RESET">
</FORM>
<ADDRESS>
    Sample form for <I>HTML For Dummies</I> Version 2.2
    1/05/96 http://www.noplace.com/HTML4D/usr-inf.html</ADDRESS>
</BODY></HTML>

```

The screen that results from this HTML document is shown in part in Figure 10-3. Notice the use of check boxes for survey information, coupled with the text input area for recipes. Makes us wonder: "What time's breakfast?"

Figure 10-3:
Notice how
you can
supply
example
information
for the text
area in a
form.

The Widget Waffle Iron Survey

Please fill out the following information so that we can register your new Widget Waffle Iron.

Model Number: 102 (Single Belgian)
103 (Double Belgian)
104 (Single Heart-shaped)

Please complete the following purchase information:

Serial number:

Purchase Price:

Location:

Please tell us about yourself:

Male ☐ Female ☐

Age: under 25 ☐ 25-39 ☐ over 40 ☐

At this point, you've seen about all the nifty little tricks — we like to call them *widgets* — that work within forms, but you can't really appreciate what forms can do until you've browsed the Web to look at the many examples out there. Our examples barely scratch the surface, so there's a lot more to see!

Formulating Good Attitudes

Whenever you create an HTML form, it's especially important to test it against as many browsers as you possibly can. Don't forget to work with character-mode browsers, such as Lynx, as well as more exciting graphical browsers. Remember also that, although some browsers support some pretty keen extensions, the effort that it takes to use them is wasted on those readers who pass through in other browsers that don't support them.



Ultimately, the HTML rules regarding layout versus content apply to forms: If you can create a clear, readable layout and make the form interesting to your users, you'll probably be a lot happier with the information returned than if you spend extra hours tuning and tweaking graphics elements and precise placement of type, widgets, and fields. Remember, too, that the form's just the front end for your user interaction or data collection. In Chapter 13, you pick up the back end of the forms business as you tackle the demanding but rewarding task of building CGI scripts and other input-handling programs for your server. If you need more details on this subject, don't forget to check out our other book: *Foundations of WWW Programming with HTML and CGI* by Ed Tittel, Mark Gaither, Sebastian Hassinger, and Mike Erwin (IDG Books Worldwide, Inc.). It even includes an analysis of the HTML and CGI code for the *HTML For Dummies* registration form!

In the next chapter, you find out how to turn graphics into addressable on-screen selectors called clickable maps. These add zest to forms and other HTML documents, because they let users select links at different points within a graphic.

Chapter 11

The Map's the Thing!

In This Chapter

- ▶ Using clickable maps
- ▶ Finding out what it takes to present maps
- ▶ Carving up maps for use
- ▶ Knowing the limitations to map use
- ▶ Mapping your way to perfection

You've already found out how to insert graphics into your HTML documents by using the `<IMG>` tag. You've even seen examples of using graphics as hypertext links within anchor tags (`<A HREF= . . . > <IMG SRC= . . . > </A>`). In this chapter, we show you how to take the next logical step and treat a graphic as a collection of selectable regions, each of which points to some kind of hypertext link or resource.

Where Are You? (Using Clickable Maps)

Geographically speaking, a map takes a land mass and divides it up along boundary lines into named regions: typically, these might be countries, counties, or other kinds of territories. When it comes to using graphics in this way on the Web, the boundaries are obvious in the graphic that's displayed, and users simply select the portion of the graphic that attracts their interest. Users familiar with graphical interfaces have no trouble interacting with on-screen buttons, icons, and other kinds of interface controls. Graphical maps add this kind of capability to a single image displayed on a Web page.

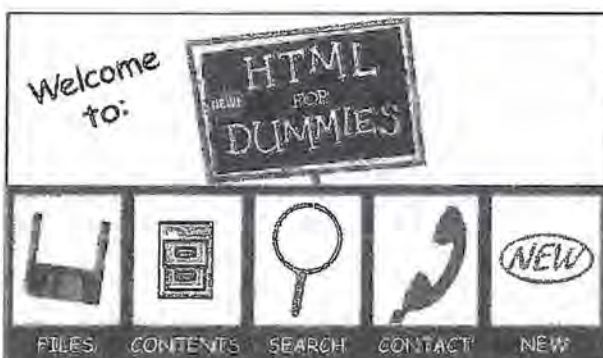
In Web-speak such graphical maps are usually called *image maps* or *clickable maps*. We prefer the latter term because it emphasizes the important aspects of this graphical element:

- ✓ It breaks a graphic into discrete regions that function as a map of individual hyperlinks.
- ✓ Regions are selected by putting the cursor inside them and clicking the mouse.

You should already have a clue about the fundamental limitation inherent in a clickable map: It absolutely requires a graphical browser. The image that represents the map and drives the selection process doesn't appear in a character-mode browser, period. Therefore, if you use clickable map elements, you must implement alternate methods for users with text-only browsers.

An example of a clickable map should lend some reality to this concept. Figure 11-1 shows the *HTML For Dummies* home page main graphic (named HT4MENU.GIF in the /GRAPHICS subdirectory). This graphic features a set of buttons at the bottom, where each button contains a major access category for the *HTML For Dummies* pages. As part of a set of Web pages for this book, it acts as the gateway to a page, or set of pages, for each category or topic mentioned.

Figure 11-1:
The *HTML For Dummies* home page (server version) includes a row of buttons on the bottom.



In this chapter, you see how to set up an image as a clickable map and how to use it to drive page navigation. Please note: The approach that we used to build the identical graphic for the pages on the diskette (the one that comes with this book) breaks the image up into pieces. The top part of the graphic represents one piece (HT4LOGO1.GIF) and each button has its own associated icon file. We did this because a local HTML file cannot use an image map; such a restriction exists because there's no server in the background to map the user's selection coordinates into a corresponding URL. Instead, each button's icon is directly linked to a URL, and clicking on a graphical element automatically selects the right link.

Cosmic Cartography: What It Takes to Present Maps on the Web

Building a clickable map requires three ingredients:

- ✓ **Creating (or selecting) a usable image:** This can be an existing graphic or a custom-built one. Our *HTML For Dummies* button bar uses five custom-built icons, one for each button.

- ✓ **Creating the *map* file:** Requires a step-by-step investigation of the image file inside a graphics program that can give you the pixel addresses (coordinates) of each point on the boundary of the regions you want to create (or use of a map-building utility, like the one we mention later in the chapter).

Our icons are all about the same size, so working through this process is very easy: The image starts in the upper-left corner of the button bar (at vertical location 142, or about the middle of this 285-pixel-high image), and is consistently 143 pixels high. The individual buttons vary slightly in width, producing the following set of coordinates:

(0,142)---	(99,142)---	(199,142)---	(299,142)---	(399,142)---	(499,142)
button 1	button 2	button 3	button 4	button 5	
(0,285)---	(99,285)---	(199,285)---	(299,285)---	(399,285)---	(499,285)

Unfortunately, the only way to produce this collection of numbers is to view the graphic from within a graphics program (We used PaintShop Pro 3.0, a widely-available shareware program for Windows.) or to use an image-map construction program. If you do it by hand with a graphics program, you must position the cursor on each boundary point and write down the corresponding x and y values. That's what our preceding diagram represents: It's the map information for our button bar.



If you use a tool, it can generate the map for you, after you tell it what kind of shapes you're outlining. One example is Tom Boutell's excellent program called *mapedit*, which builds map files for you on command. It's available for lots of computer types including Macintosh, DOS/Windows, Windows NT, and UNIX. To find a suitable version, use "mapedit" as the search string in your favorite Web search engine. (A quick jump to Yahoo! turned up over 40 sites that offer one or more versions of this program.)

- ✓ **Establishing the right HTML information in your page:** You need to link the image, the map file, and a CGI script to handle decoding map coordinates and use that information to select the appropriate link to follow.

Here, we take you through the generalities of making this work in your HTML document. Later on, we show you how to build a complete back-end CGI script to translate the pixel coordinates for a user's map selection into a corresponding HTML link. In this chapter, we only cover the techniques and generalities needed to construct the image map and to create links between map regions and hypertext documents or resources.

By obtaining the information for the images and map files and establishing a convention to call the script that handles coordinate-to-link translation, you have most of what you need to know to set up a clickable map.



Just as we did in Chapter 10, we suggest that you use the name of the image map as the name for its corresponding script file: Thus, if the image is named HT4MENU.GIF, the script would be called HT4MENU.MAP, or simply HT4MENU. If your scripts are in a CGI directory, one level down from your HTML files, the corresponding URL for this script would then be `/cgi-bin/ht4menu.map`.

Warning: different maps for different servers

Unfortunately, clickable maps are a topic where the two most popular *httpd* servers — NCSA and CERN — differ from one another. You find format differences in defining image maps for one kind of server versus the other. You also find Web servers that don't support either format, especially if they're not running UNIX. (Windows NT currently supports about 15 different Web servers, many of which use their own proprietary image map formats; the leading Macintosh Web server — WebSTAR — supports NCSA format.)

In order to get your clickable map to work, you must work within the requirements of the server where the map resides. If you don't know what those requirements are, we strongly suggest that you contact your local WebMaster — or at least, the system administrator for your Web server. He or she should be able to set you straight right away and can probably help you find some useful information about how to build clickable maps for your system, above and beyond what we tell you here.

Throughout the chapter, where differences between the CERN and NCSA requirements exist, we fill you in. If you're not using an *httpd* server of either variety, you'll want to investigate your server's requirements immediately and adjust our examples and recommendations to meet those requirements!

Dealing with shapes in maps

You can use various ways to identify boundaries when assembling coordinates to build a clickable map. Both CERN and NCSA image map definitions recognize the following regions:

- ✓ **Circle:** (Specified by the coordinates for a point at the center and the number of pixels for the radius.) Use this to select a circular (or nearly circular) region within an image.
- ✓ **Rectangle:** (Specified by the coordinates for the upper-left and lower-right corners.) Use this to select a square or rectangular region in your image. (This is the one we use in our button bar map.)
- ✓ **Polygon:** (Specified by the coordinates for the point at the vertex of each edge.) Use this to outline the boundaries of regularly or irregularly shaped regions that aren't circular or rectangular. Although it takes more effort,

the more points you pick to define the outline, the more the region behaves as the user expects it to when they click it.

- ✓ **Point:** (Specified by its x and y coordinates.) Use this only when a specific point is easy for a user to select. (A point is usually too small a region on-screen and requires exact control to select — we recommend surrounding a point with a small circle or square to give users a little room to be sloppy.) We've never actually used a point reference in an image map, except as the vertices for a polygon or rectangle or as the center of a circle.

Selecting boundaries for map regions is what determines the selection of the corresponding links. Even though users see a nicely shaped graphic to click on, what really drives the selection process are the areas that you outline on top of that graphic.



The better the map regions fit the figure layout, the more the map behaves like users expect it to! The moral of the story is: Take your time and, when in doubt, pick more points to outline something, rather than less. (Even better, use a tool that follows your cursor movement to build the image map for you.)



We mentioned Tom Boutell's *mapedit* program earlier; you can find alternatives galore on the Web. For example, an interactive, Web-based image map tool called *MapMaker* is available at

<http://www.tns.lcs.mit/cgi-bin/mapmaker>

If you provide the program with a URL for the graphics file that you want to build a map for, it guides you through the rest of the process using your very own Web browser!

Building and Linking to CERN Map Files

Map files for CERN *httpd* servers take a general form that looks something like this:

```
default URL
circle (x,y)-r URL
rectangle (x1,y1) (x2,y2) URL
polygon (x1,y1) (x2,y2) (x3,y3) ... (xn,yn) URL
point (x,y) URL
```

The shapes are pretty much self-evident, except for the polygon, which represents an attempt to trace a region's outline by connecting a series of points. If this sounds like "connect-the-dots" you've definitely got the underlying concept!



Don't forget to close your polygons; make sure that the last segment fills the gap between your last point and your first.

The other entry that might seem mysterious is the default URL: Define the default so that if a user clicks on a location that's not defined in the map, a fail-safe connection can still be chosen. This can be a script that sends a message back to the user, saying, "Click within the lines!" or "You have selected an area of the image that is not defined. Please try again."

The menu bar map file

Thus, for our menu bar example, the CERN map would be

```
rectangle (0,142) (98,285) http://www.outer.net/html4dum/ftpstuff.htm
rectangle (99,142) (198,285) http://www.outer.net/html4dum/contents.htm
rectangle (199,142) (298,285) http://www.outer.net/html4dum/search4d.htm
rectangle (299,142) (398,285) http://www.outer.net/html4dum/contact.htm
rectangle (399,142) (499,285) http://www.outer.net/html4dum/whatsnew.htm
default http://www.outer.net/html4dum/contents.htm
```

Because the button bar is a collection of rectangles, filling in the coordinates is easy. (Why do you think we picked this example?) Then, we provide a default link to the contents page if somebody insists on staying outside the nice little boxes that we gave them to play in! Notice, too, that we use URLs here that are relative to where the current document is in the file system of the server. Absolute URLs are easier to debug and to relocate, but relative ones are less work to enter. You plays your keystrokes, and you takes your chances!

Creating and storing map files

You can create the map file with any plain text editor, but you should store it on the server in a special directory for your map definition files. For this example, we call the file HT4MENU.MAP and store it in the `http://www.outer.net/cgi-bin/` directory, along with all our other scripts.

With either CERN or NCSA servers, image maps need to be placed in a certain part of the file system. Contact your system administrator or your WebMaster to find out where this is and if you have *write* permissions. If you don't, you'll need to enlist their help in getting those files installed. (Normally, you can e-mail them to the administrator with an explanation of what they are, what they should be named, and where you assume they'll be installed; if you don't know those things yet, ask your administrator to supply those details before you mail him or her anything!)

Using map files

To use a map file with the CERN *httpd*, your system must already have a program that handles image maps. The name of this program, which is included with the CERN distribution *httpd* materials, is *htimage*. If it's not installed, you must arrange for that to happen in order to use image maps on your system. After it's available, you must also know how to invoke it on the server. For the purposes of this example, we assume it lives on the directory path `/CGI-BIN/`.

Defining a clickable map in your HTML document

After the map's defined and stored in the right location, it's time to bring all three elements together in your HTML file. Here's how you do it:

```
<A HREF="http://www.outer.net/cgi-bin/ht4menu.map">  
<IMG SRC="graphics/ht4menu.gif" ISMAP>  
</A>
```

Here's what's going on in this series of statements:

- ✓ The opening anchor tag combines the *htimage* location, which handles the coordinate-to-URL translation, with the full specification for the map file. Even though there's no space between the name of the program (*htimage*) and the map file specification, the server still knows what to do.
- ✓ The `IMG` tag points to the button bar graphic, but adds the `ISMAP` attribute to indicate that it's a clickable map.
- ✓ The closing anchor tag `</A>` indicates that the graphic specified by `IMG` is the target for the map file specified in the opening anchor tag.

There you are! Once you've made sure that all the right pieces are in place on your CERN server, you can try this, too.

Building and Linking to NCSA Map Files

Map files for NCSA *httpd* servers look an awful lot like those for CERN servers, but there are some differences. They take a general form that looks something like this:

```
default URL  
circle URL x,y r  
rect URL x1,y1 x2,y2  
poly URL x1,y1 x2,y2 x3,y3 ... xn,yn  
point URL x,y
```

The shapes are the same as the CERN variety and are defined by the same kinds of coordinates, but the names are shorter, and the URLs come first in the list of attributes. Here again, defaults work the same way: to provide a handler for people who click outside the image frame.

The button bar map file

For the *HTML For Dummies* home page main graphic, for example, the NCSA map would be

```
rect http://www.outer.net/html4dum/ftpstuff.htm (0,142) (98,285)  
rect http://www.outer.net/html4dum/contents.htm (99,142) (198,285)  
rect http://www.outer.net/html4dum/search4d.htm (199,142) (298,285)  
rect http://www.outer.net/html4dum/contact.htm (299,142) (398,285)  
rect http://www.outer.net/html4dum/whatsnew.htm (399,142) (499,285)  
default http://www.outer.net/html4dum/contents.htm
```

Except for a slight change in the shape's name (rect instead of rectangle) and a reordering of the arguments (URL first, then coordinates), the map is pretty much identical to the CERN variety.

Creating and storing map files

You can create the NCSA map file with any plain text editor, just like the CERN variant. It should also be stored on the server in a special directory for your map definition files. For this example, we use the same name, HT4MENU.MAP, and store it in the same `http://www.outer.net/cgi-bin/directory`, along with all our other scripts.

Using map files

Like CERN, in order to use a map file with the NCSA *httpd*, your system must already have a program that handles image maps. The name of this program, which is included with the NCSA distribution *httpd* materials, is *imagemap*. If it's not installed, you must arrange for that to happen in order to use image maps on your system. You also want to check that you have the latest version; check that information in

```
http://hoohoo.ncsa.uiuc.edu/docs/setup/admin/imapmap.txt
```

against what's installed on your server. If the date is more recent at the NCSA server, download the file, rename it to `imapmap.c`, and recompile it. Throw the old version away and use your new one instead.

After you have *imapmap* available, you must also know how to invoke it on the server. For the purposes of this example, we assume it lives on the directory path `/CGI-BIN/`.

Defining a clickable map in your HTML document

After the map's defined and stored in the right location, again it is time to bring all three elements together in your HTML file, but this time the NCSA way. Here's how you do it:

```
<A HREF="http://www.outter.net/html4dum/ht4menu.map">  
<IMG SRC="graphics/ht4menu.gif" ISMAP>  
</A>
```

Here's what's going on in this series of statements:

- ✓ The opening anchor tag combines the *imapmap* location, which handles the coordinate-to-URL translation, with the full specification for the map file. Even though there's no space between the name of the program (*imapmap*) and the map file specification, the server still knows what to do.
- ✓ The `IMG` tag points to the button bar graphic, but adds the `ISMAP` attribute to indicate that it's a clickable map.
- ✓ The closing anchor tag `</A>` indicates that the graphic specified by `IMG` is the target for the map file specified in the opening anchor tag.

There you are! After you've made sure that all the right pieces are in place on your NCSA server, you can try this, too.

Even though the differences between the two flavors of *httpd* may seem trivial — and we think they are, too — they're still essential to creating clickable images that actually work. When it comes to computers in general, and the Web in particular, the devil is in the details. If you don't get your details straight, you get bedeviled instead! That's why you need to get the details straight before you try anything too exotic.

"The Map Is Not the Territory"

Although Alfred Korzybski wasn't thinking of clickable images when he uttered the title of this section, it's still a point worth pondering. Because not all users can see an image map, you need to be prepared to give them the same set of selections that your graphically-able users get in visual form. How might you go about doing this?

Because what you're providing in the image map is an array of choices, you could also add an equivalent set of text-based links right below the image. Here's what the HTML for this would look like:

```
<HTML>
<HEAD>
<TITLE> HTML For DUMMIES HomePage - IDG Books</TITLE>
<!--Turn a BASE HREF on if you have any problems with the way
this page behaves (e.g. unresolved graphics). Also be sure to
provide an absolute URL that points to the home page on your Web
server, e.g.
BASE HREF="http://www.outer.net/html4dum/html4dum.htm"
This is the proper BASE reference for our own HTML For Dummies
server. -->
<BODY>
<A NAME="top"></A>
<IMG ALIGN=MIDDLE WIDTH=60 HEIGHT=0 SRC=graphics/space.gif
ALT=" ">
<A HREF="http://www.outer.net/cgi-bin/html4dum.map">
<IMG BORDER=0 ALIGN=TOP SRC="graphics/ht4menu.gif"
ALT="Navigation Bar" ISMAP></A><P>
<IMG ALIGN=MIDDLE WIDTH=130 HEIGHT=0 SRC=graphics/space.gif
ALT=" ">
<B><A HREF="ftpstuff.htm">FILES</A> &#32;&#124;
<A HREF="contents.htm">CONTENTS</A> &#32;&#124;
<A HREF="search4d.htm">SEARCH</A> &#32;&#124;
<A HREF="contact.htm">CONTACT</A> &#32;&#124;
<A HREF="whatsnew.htm">NEW</A><BR>
<IMG ALIGN=MIDDLE WIDTH=240 HEIGHT=0 SRC=graphics/space.gif
ALT=" ">
<A HREF="navigate.htm">HOW TO NAVIGATE</A></B><P>
<A HREF="html4du2.htm">Click here for a non-imagemap version</A>
<P>
<IMG WIDTH=100% SRC="graphics/line.gif"
ALT="===== "><P>
```

You have reached the <I> HTML For Dummies</I> Web Pages, a charming, and hopefully helpful, addition to the WWW universe.

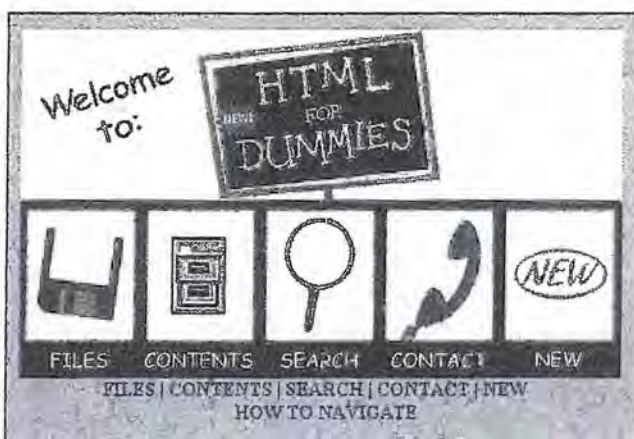
```

These pages are designed to aid you in three key areas:
<UL>
<LI> To help you find current information on the Web about HTML
<LI> To provide working examples and code for all the Web tricks
    in the book
<LI> To introduce <I> HTML For Dummies</I> - your friendliest
    resource for HTML material offline!
</UL>
<!-- rest of file continues here, but this is all we include -->

```

As shown in Figure 11-2, this results in a text bar (right underneath the graphic) that offers the same choices that the graphic does. Users with graphical browsers won't be discommoded by this redundancy, and character-mode browsers will see a reasonable facsimile of what their graphically advantaged brethren see in living color. We call this *mastering the art of compromise!*

Figure 11-2:
A text-based alternative combined with the button bar keeps everybody in the know.



Of Clickable Maps and URLs

Image links can sometimes play hob with relative URL specifications within HTML documents. One unforeseen side effect of following the links through a map-reading script — to the map and back to the target page — can be a complete mangling of the context within which URLs are addressed. In English, this means it's a really, really, really good idea to use full URLs, instead of relative references, in HTML documents that also include clickable maps.

Another solution is to include a `<BASE>` tag inside the `<HEAD> . . . </HEAD>` tags, with the URL for the document in which it's included. This forces the server to treat the directory where the current document resides as the default context for relative references, even if clickable map references are used.

Either way, it's smart to be cagey when using relative URL references in documents with clickable maps. You should test this practice thoroughly to make sure that everything works as it should or use the `<BASE>` tag. Avoiding trouble is, in general, the best way to cure the URL "relative reference blues"!

Get ready for some real action in the next chapter, as you finally roll up your sleeves and get down and dirty on the back-end side of the Web environment. Here, you'll get the lowdown on what happens with CGI scripts, input-handling programs for forms, search engines, and other tools that extend what the Web can do for your users. Better buckle up!

Chapter 13

The Common Gateway Interface (CGI)

In This Chapter

- ▶ Getting acquainted — a quick CGI overview
- ▶ Specifying CGI
- ▶ Making smart programming choices
- ▶ Forming connections — handling forms data
- ▶ Sampling the capabilities

Building HTML forms and handling user interaction through Web pages require action on both sides of the client/server connection. So far, this book has concentrated mainly on the client-side, except for the clickable image map files that we described in Chapter 11.

In this chapter, we move across the network connection from the client to the server-side and describe the Common Gateway Interface (CGI). The CGI lets Web pages communicate with programs on the server to provide customized information or to build interactive exchanges between clients and servers.

Along the way, we show you the history and foundations of CGI and cover the basic details of its design and use. We also introduce you to the issues surrounding your choice of a *scripting*, or programming, language for CGI, and give you the chance to read through some interesting example programs. Because you're getting these programs — and the code to run them — as part of this book (the code is on the diskette), we hope that you end up using them as part of your own Web pages!



If you've been reading carefully, you've already noticed one important thing about CGI programs: They run on the Web server, not on the client-side (where you run your browser to access Web-based information). This means that you can't use our CGI programs on your own machine, unless it can run Web server software (usually called an *HTTP daemon*, or more simply, *httpd*) at the same time that it runs your browser.

On Windows NT, Macs, and UNIX machines, running both a Web server and a browser is quite doable; on Windows 3.x and Windows 95 computers, it's a bit trickier. For the latter machines, Quarterdeck's \$99 WebServer software package fits the bill quite nicely: It supports the latest version of CGI (1.1) and lets you work both sides of the client/server street with ease. For more information on this product, visit Quarterdeck's Web site at



<http://www.qdeck.com>

The "Common Gateway" Is NOT a Revolving Door!

Gateway *scripts*, or programs, add the capability for true interaction between browsers and servers across the Web. This is a powerful capability that's limited only by your imagination and the tools at hand. Gateway scripts supply the underlying functions that let you perform searches on Web documents or databases that provide the capacity to accept and process forms data, and that deliver the intelligence necessary to customize Web pages based on user input.

If you build Web pages, you must manage this interaction across the Web: That is, you must build the front-end information that users see and interact with, as well as the back-end programming that accepts, interprets, and responds to user input and information. This does require some effort and some programming, but if you're willing to take the time to learn CGI, you are limited only by the amount of time and energy you have for programming your Web pages.

Describing CGI programs

It doesn't really matter whether you call your server-side work a program or a script — in fact, this distinction is often used to describe the tools you'd use to build one. That is, scripts get built with scripting tools or languages; programs get built with programming languages. Like a script read by an actor, a scripting language creates a set of actions and activities that must be performed in the prescribed order each time a script is executed; that's why scripts are often said to be *interpreted*. Programming languages, on the other hand, must usually be transformed into a special executable form by a special program called a *compiler* that takes the programming language statements and turns them into equivalent computer instructions; that's why programs are often said to be *executed*. Because both approaches work equally well with the Web's Common Gateway Interface, for convenience, we simply refer to them as *CGI programs*. You can call them whatever you want!

CGI is the method that UNIX-based CERN and NCSA Web servers use to mediate interaction between servers and programs. Because UNIX was the original Web platform, it sets the model (yet again) for how the Web handles user interaction. Although you might find other platforms that support Web servers—like Windows NT and the Macintosh—they, too, must follow the standard set for the CERN and NCSA implementations of *httpd*. Whether or not they conform to the CERN or NCSA models, Web servers must provide CGI capabilities, or a similar set of functions to match what CGI can do.

What's going on in a CGI program?

You can think of a CGI program as an extension to the core WWW server services. In fact, CGI programs are like worker bees that do the dirty work on behalf of the server. The server is an intermediary between the client and the CGI program. It's good to be the queen bee, and to make all those workers do their things at your behest!

The server invokes CGI programs, based on information provided by the browser (as in the `<FORM>` tag, where the `ACTION` attribute supplies a URL for the particular program that services the form). The browser request sets the stage for a series of information hand-offs and exchanges:

- ✓ The browser makes a request to a server for a URL, which actually contains the name of a CGI program for the server to run.
- ✓ The server fields the URL request, figures out that it points to a CGI program (usually by *parsing* the filename and its extension or through the directory where the file resides), and fires off the CGI program.
- ✓ The CGI program performs whatever actions it's been built to supply, based on input from the browser request. These actions could include obtaining the date and time from the server's underlying system, accumulating a counter for each visit to a Web page, searching a database, and so on.
- ✓ The CGI program takes the results of its actions and returns the proper data back to the server; often the program formats these results as a Web page for delivery back to the server (if the Content-Type is `text/html`).
- ✓ The server accepts the results from the CGI program and passes them to the browser, which renders them for display to the user.
- ✓ If this exchange is part of an ongoing, interactive Web session, these results could include additional form tags to accept further user input, along with a URL for this or another CGI program. Thus, the cycle begins anew.

This is a kind of disjointed way to have a “*conversation*” over the Web, but it does allow information to move both ways. The real beauty of CGI programs is that they extend a simple WWW server in every conceivable direction and make that server's services more valuable.

What's in CGI Input?

A request for a CGI program is encoded in HTML in a basic form as shown in this example:

```
<A HREF="http://www.hal.com/hal-bin/silly_quote.pl">Silly Quote</A>
```

The URL declaration says to execute the `silly__quote.pl` CGI program on the `www.hal.com` WWW server from the `hal-bin` directory. This request has no additional input data to pass to the CGI program. (The clue is that no "?" is appended to the URL — see the explanation of the question mark argument later in this chapter.) The result of the CGI program is a Web page created on-the-fly and returned to the browser.

The information gathered by an HTML form or requested by a user (with a search request or other information query), passes to CGI programs in one of two ways:

- ✓ As an appendage to a CGI program's URL (most commonly, for WAIS requests or other short information searches). This way uses the `METHOD="GET"` option.
- ✓ As a stream of bytes through the UNIX default standard input device (*stdin*) in response to the `ACTION` setting for an HTML `<FORM>` tag. This way is best used with the `METHOD="POST"` option.

In the next section, you see how forms create special formats for information that is intended for use in CGI programs. Also, you see how that information is delivered and used in those programs.

Short and sweet: the "extended URL" approach

Most search engines use what's called a *document-based query* to obtain information from users. This query consists of nothing more than special characters appended to the end of the search engine's URL. Document-based queries are intended to solicit search terms or key words from a browser and then deliver them to a CGI program that uses them to search a database or a collection of files. Their simplicity is what makes document-based queries so good for soliciting small amounts of input from users and why you see them in so many Web pages.

Document-based queries depend on three ingredients for their successful operation:

- ✓ The `<ISINDEX>` tag within the `<HEAD>` section of an HTML document enables the browser to search the document.
- ✓ A special URL format is generated by adding the contents of a query to the URL, with the search terms added at the end and denoted by question marks.
- ✓ Special arguments in your underlying CGI program.

Here's how the process actually works:

- ✓ The `<ISINDEX>` tag in the `<HEAD>` of the document causes the browser to supply a *SEARCH widget* that allows the user to enter keywords (a widget is a bit of software that performs a particular task; in this case, it handles packages and sending search requests). These keywords are then bundled into an HTTP request and are passed to the corresponding CGI program. If the CGI program finds that no arguments are appended to the URL, it returns a default page that includes the search widget to the browser. This sometimes happens the first time a search is requested because the complete search widget may not be included on every Web page.
- ✓ At the prompt, the reader enters a string to search for, and presses Enter (or otherwise causes the string to be shipped to the CGI program for handling).
- ✓ The browser calls the same URL as before, except it appends the search string following a question mark. So, if the search engine program's URL is

```
http://www.HTML4d.com/cgi-bin/searchit
```

and the string to be searched for is "tether," the new URL then becomes

```
http://www.HTML4d.com/cgi-bin/searchit?tether
```

- ✓ The server receives the URL exactly as formatted and passes it to the `searchit` program, with the string after the question mark passed as an argument to `searchit`.
- ✓ This time, the program performs an actual search and returns the results as another HTML page (instead of the default prompt page that was sent the first time).

Every part of this operation depends on the others: The browser activates the `<ISINDEX>` tag that allows the query to be requested and entered; then the browser also appends the query string to the URL and passes on the query as an argument to the search program. The search program uses the query value as the focus of its search operation and returns the search results to the browser, via another custom-built HTML document.

Long-winded and thorough: the input-stream approach

As you build an HTML form, you have important definitions to make, including the assignment of names and associated values to your variables or selections. When users fill out HTML forms, they're actually instructing the browser to build a list of associated `name=value` pairs for each selection made or for each field that's filled in.

Name=value pairs take the following form:

```
name=value&
```

The equal sign (=) separates the name of the field from its associated value. The ampersand (&) separates the end of the value's string from the next item of text information in a completed form. For `<SELECT>` statements where MULTIPLE choices are allowed, the resulting list has multiple `name=value` pairs where the name remains the same, but the value assignment changes for each value chosen.



Reading through forms information delivered to a CGI program's standard input device (*stdin*) is a matter of checking certain key environment variables (covered in the next section), and then parsing (separating into individual words or units of information) the input data. This reading consists of separating `name=value` pairs, and using the names, with their associated values, to guide subsequent processing. The easiest way to do this, from a programming perspective, is to first parse and split out `name=value` pairs by looking for the ending ampersand (&), and then divide the pairs into the name and value parts by looking for the equal sign (=).



Here are some Perl code fragments that you can use to parse your forms' input data. (It assumes that, in keeping with our recommendation, you use `METHOD="POST"` for passing data.)

```
# this reads the input stream from the Standard Input
# device (stdin) into the buffer variable $buffer, using
# the environment variable CONTENT_LENGTH to know how
# much data to read
read(stdin, $buffer, $ENV{'CONTENT_LENGTH'});

# Split the name=value pairs on '&'
@pairs = split (/&/, $buffer);

# Go through pairs and determine the name and value for
# each named form field
```

```

for each $pair (@pairs) {
# Split name from value on "="
($name, $value) = split(/=/, $pair);
# Translate URL syntax of + for blanks
$value =~ tr/+// /;
# Substitute hexadecimal characters with their normal
  equivalents
$value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
# Deposit the value in the FORMS array, associated to name
$FORM($name) = $value;

```

Handling environment variables

As part of the CGI environment, the *httpd* server's software version and configuration are of interest, as are the multiple variables associated with the server. You can use the following shell program to produce a complete listing of such information; it is a valuable testing tool when installing or modifying a Web server.

```

#!/bin/sh

echo Content-type: text/plain
echo

echo CGI/1.1 test script report:
echo

echo argv is $#. argv is "$*".
echo

echo SERVER_SOFTWARE = $SERVER_SOFTWARE
echo SERVER_NAME = $SERVER_NAME
echo GATEWAY_INTERFACE = $GATEWAY_INTERFACE
echo SERVER_PROTOCOL = $SERVER_PROTOCOL
echo SERVER_PORT = $SERVER_PORT
echo REQUEST_METHOD = $REQUEST_METHOD
echo HTTP_ACCEPT = $HTTP_ACCEPT

echo PATH_INFO = $PATH_INFO
echo PATH_TRANSLATED = $PATH_TRANSLATED
echo SCRIPT_NAME = $SCRIPT_NAME
echo QUERY_STRING = $QUERY_STRING
echo REMOTE_HOST = $REMOTE_HOST
echo REMOTE_ADDR = $REMOTE_ADDR
echo REMOTE_USER = $REMOTE_USER
echo CONTENT_TYPE = $CONTENT_TYPE
echo CONTENT_LENGTH = $CONTENT_LENGTH

```



This UNIX shell script is widely distributed around the Net. We found this version in the NCSA hoohoo collection at

```
http://hoohoo.ncsa.uiuc.edu/cgi-bin/test-cgi
```

Running this script on a Web server (in this case, the NCSA Web server hoohoo) produces the following output:

```
CGI/1.1 test script report:

argc is 0. argv is .

SERVER_SOFTWARE = NCSA/1.4b2
SERVER_NAME = hoohoo.ncsa.uiuc.edu
GATEWAY_INTERFACE = CGI/1.1
SERVER_PROTOCOL = HTTP/1.0
SERVER_PORT = 80
REQUEST_METHOD = GET
HTTP_ACCEPT = */*, image/gif, image/x-xbitmap, image/jpeg
PATH_INFO =
PATH_TRANSLATED =
SCRIPT_NAME = /cgi-bin/test-cgi
QUERY_STRING =
REMOTE_HOST = etittel.zilker.net
REMOTE_ADDR = 198.252.182.167
REMOTE_USER =
AUTH_TYPE =
CONTENT_TYPE =
CONTENT_LENGTH = n
```

Each capitalized variable name (on the left-hand side of the equal sign) and its associated output are environment variables set by CGI. These are always available for use in your programs.

Two environment variables are especially worthy to note:

- ✓ The `QUERY_STRING` variable is associated with the `GET` method of information-passing (as is common with search commands) and must be parsed for such queries or information requests.
- ✓ The `CONTENT_LENGTH` variable is associated with the `POST` method of information-passing (as is recommended for forms or other lengthier types of input data). The browser accumulates the variable while assembling the forms data to deliver to the server; the variable then tells the CGI program how much input data it has to read from the standard input device.

The environment variables also identify other items of potential interest, including the name of the remote host and its corresponding IP address, the request method used, and the types of data that the server can accept. As you become more proficient in building CGI programs, you find further uses for many of these values.

Forming Up: Input-Handling Programs

When you create a form in HTML, each input field has an associated unique NAME. Filling out the form usually associates one or more values with each name. As shipped from the browser to the Web server (and on to the CGI program targeted by the URL), the form data is a stream of bytes, consisting of `name=value` pairs separated by ampersand characters (&).

Each of these `name=value` pairs is URL-encoded, which means that spaces are changed into plus signs (+) and some characters are encoded into hexadecimal. Decoding these URL encodings is what caused the interesting translation contortions in our Perl code sample in the preceding section.

If you visit the NCSA CGI archive, you can find links to a number of input-handling code libraries that can help you build forms. You can find all this information at



ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/cgi/

- ✓ **Bourne Shell:** The AA Archie Gateway, which contains calls to *sed* and *awk* to convert a GET form data string into separate environmental variables (`AA-1.2.tar.Z`)
- ✓ **C:** The default scripts for NCSA *httpd*, including C routines and example programs for translating the query string into various structures (`ncsa-default.tar.Z`)
- ✓ **Perl:** The Perl CGI Library contains a group of useful Perl routines to decode and manage forms data (`cgi-lib.pl.Z`)
- ✓ **TCL:** The TCL argument processor includes a set of TCL routines to retrieve forms data and insert it into associated TCL variables (`tcl-proc-args.tar.Z`)

What this means to you, gentle reader, is that most of the work of reading and organizing forms information is already widely and publicly available in one form or fashion. This simplifies your programming efforts because you can concentrate on writing the code that interprets the input and builds the appropriate HTML document that's returned to your reader as a response.

Coding CGI

We conclude this chapter by examining three different CGI programs. We've also included their code and a great deal of associated information on the diskette that comes with this book. Each of these programs is available in three versions: AppleScript (for use on an Apple *httpd* server), Perl (for use on any system that supports a Perl interpreter/compiler), and C (for use on any system that supports a C compiler).

Ladies and gentlemen: Choose your weapons!

Before we launch you into these excellent examples, we'd like to encourage you to use them in your own HTML documents and CGI programs (we're giving the license to this code away, so you can use it without restrictions). We want to conclude the discussion portion of this chapter with an investigation of why we chose to implement each of these programs in three forms, and how you can choose a suitable language for writing your CGI programs.

It's quite true that you can build your CGI programs with just about any programming or scripting language that your Web server supports. Nothing can stop you from ignoring all of the options that we cover here and use something completely different.

Nevertheless, we can think of some good reasons why you should consider using these options and equally good reasons why you should ignore other options. We cheerfully concede that there are probably as many opinions on this subject as there are CGI programmers, but we'd like you to consider carefully before deciding on a CGI language that you're likely to spend considerable time and effort learning and using.

For example, we included the NCSA test-cgi script earlier in this chapter. It's written in the basic C shell, a command language that is common on many UNIX systems and makes an adequate scripting language for many uses. Nevertheless, we don't think that UNIX shells are suitable for heavy-duty CGI programming, because they mix UNIX system commands freely within their own syntax.

The problem with CGI programming under UNIX is that it depends on the standard input (*stdin*) and standard output (*stdout*) devices as the methods for moving data between Web servers and browsers. Each new UNIX process automatically creates its own *stdin* and *stdout*; sometimes UNIX shells can get confused regarding where its input is coming from and where its output is going to. This confusion can be a side effect of *spawning* tasks (or having one running program start another program up to perform a specific task and report its results back to the original program when its finished) or running system

commands, but whatever the cause, it can lose the input or output for CGI programs. That's the main reason why we don't recommend shell scripts of any kind for heavier-duty CGI applications (for example, forms-processing versus query-handling).

On the plus side, Perl offers straightforward access to UNIX system calls and capabilities within a tightly structured environment. It includes the positive features of languages like C, Pascal, *awk*, *sed*, and even Basic, and offers powerful string-handling and output-management functions. Perl is emerging as the favorite of many Web programmers (and is certainly our favorite CGI gurus' language of choice). Best of all, Perl implementations are already available for UNIX, DOS, Windows NT, the Macintosh OS, and the Amiga, with numerous other implementations under way. We've had excellent luck moving Perl from one platform to another, with only small changes.

We include C because it's a powerful programming language and remains a tool of choice in the UNIX environment. What features and functions it doesn't offer as built-ins are readily available in the form of system APIs (Application Programming Interfaces, or the set of routines used to invoke system functions and other kinds of prepackaged functionality within a program) and code libraries. C is also very portable (barring the use of system APIs, which changes from one system to another); one version of C is available for just about every platform, and, with multiple implementations, are available for popular platforms and operating systems. We're especially fond of the GNU C and the related GNU Tools from the Free Software Foundation pioneered by Richard Stallman.



If you use the Macintosh as a Web server, AppleScript is pretty much your only option. Even so, it has proven to be a worthwhile tool for building CGI programs and is widely used in the Macintosh Web community. Be sure to consult Chapter 21 for some excellent pointers on Macintosh tools and technologies for the Web, if you're a real Mac-o-phile!

Whatever language you choose for your CGI programs, be sure that it provides good string-handling capabilities and offers reasonable output controls. Because you'll be reading and interpreting byte stream input and creating HTML documents galore, look for these important capabilities. Also, we recommend that you pick a language that is already widely used in the Web community; you'll likely find lots of related modules, libraries, and code widgets that may save you programming time and make your job easier. But hey—it's your choice!



In this chapter, we just barely scratch the surface of CGI as a topic. For more information, consult your favorite search engine and use "CGI" or "CGI script" as your search string. It turns up tons of useful references. Also, one of the authors of this book has cowritten two other books that are largely devoted to CGI programming. They are *The Foundations of WWW Programming with HTML and CGI* and *World Wide Web Programming Secrets with Perl and CGI* both by Ed Tittel, Mark Gaither, Sebastian Hassinger, and Mike Erwin. (The former book

was published in 1995, the latter in 1996. Both books were published by IDG Books Worldwide, Inc., Programmers Press, and both have list prices of \$39.99 in the U.S.). Although you can also find other books on this subject, you won't find any others that your authors like more!

Example 1: What time is it?

This short Perl program accesses the system time on the server and writes an HTML page with the current time to the user's screen.

```
#!/usr/local/bin/perl
#
# time-cgi.pl
#
# CGI to return the time on the server
# v 1.0 - 2/14/95
# singe@outer.net <Sebastian Hassinger>
#
# called from simple anchor
# (i.e. <A HREF="http://www.outer.net/cgi-bin/time.pl">)
# the first line tells the UNIX shell that this script should be run
# using Perl in the /usr/local/bin directory
#
#
$| = 1;      # output *not* buffered - This is a switch used by
              # Perl internally. It tells Perl not to buffer
              # the text the script sends back to the Web client.
              # All data will be output immediately to the client
              # thereby avoiding any data loss when the connection
              # to the client is broken.

print "Content-Type: text/html\n\n\n";
# This is the first line the CGI
# must return to signal the Web client that the data
# to follow should be interpreted as HTML.
# The line states the MIME type of the document to
# follow, and the carriage returns after the
# 'Content-Type:' line (\n) are required.
```

```

chop ( $time = `/bin/date +%r` );
    # Sets the $time variable to the current time
    # using the UNIX 'date' command and the %r format
    # built-in to that command to return the time in
    # AM/PM 12-hour format. The ' ' (back-ticks) tell
    # Perl to use the text in between as input to the
    # Bourne shell and to return the results. The 'chop'
    # command wrapping variable assignment deletes the
    # trailing carriage return from the shell output
    # (i.e., the raw data returned to the variable will
    # be '11:15:30 AM\n' and the chop command takes away
    # the \n).

print "<HTML><HEAD><TITLE>Current Time on this Web Server
    </TITLE>",</HEAD><BODY>\n";
    # Set up the top of the page for the client. From
    # here on out, everything we output is going to be
    # HTML-formatted - in essence, we are creating an
    # HTML document on-the-fly.
    # the print command can be broken up over several
    # lines using comma-separated chunks of double-
    # quoted text.

print "<H2>The time on the server is now:<BR><STRONG>$time
    </STRONG><BR></H2>";

    # We just outputted our variable using HTMLtag to
    # pretty up our text. Note that \n carriage returns
    # do not affect a carriage return on the client -
    # only <BR> tags do.

print "<HR><ADDRESS>This page generated by time-cgi.pl<P>",<br>
    "Contact: singe@outer.net</ADDRESS></BODY></HTML>";
    # Finish off the page with contact info.

# The script will now complete its execution, the shell opened by
# the httpd server to run the CGI script will close, and the
# connection between the server and the client will be broken.

```

Example 2: Counting page visits

This AppleScript program establishes a counter that tracks the number of times a page is visited. This kind of tool can provide useful statistics for individuals or organizations curious about how much traffic their pages actually receive.

- counter.acgi (AppleScript CGI to return the number of times the CGI has been accessed)

- singe@outer.net <Sebastian Hassinger>
- 2/15/95

- simple example of AppleScript CGI for running on a Mac equipped with MacHTTP 2.0.1

- First, set up variables as properties so that they will persist inside the object that is called when the client accesses the CGI. Regular variables in AppleScript don't need to be declared as properties before use, but they would lack the scope to be used inside event handling objects. Also, the counter and access_date properties will retain their values even if you quit and launch the script again, so long as it isn't edited.

property crlf : (ASCII character 13) & (ASCII character 10)

- Set up a UNIX-style carriage return/line feed to make sure the lines break properly on the Web client.

property http_10_header : "HTTP/1.0 200 OK" & crlf & "Server: MacHTTP" & crlf & "MIME-Version: 1.0" & crlf & "Content-type: text/html" & crlf & crlf

- This is a standard http header, we're constructing it for use later in the script, to signal the client what type of data we're returning. MacHTTP is unlike the NCSA UNIX-based httpd server in that it does not supply any missing http headers for the CGI's output, so in addition to the usual 'Content-type:' header we used in the UNIX CGIs, we supply the HTTP version, the server name, MIME version and result code (200, in this case, signalling the client that we're returning a complete HTML document).

property html_footer : "</BODY><HR><ADDRESS>This page generated by counter.acgi.<P>" & "Example AppleScript CGI<P>Contact singe@outer.net with questions." & "</ADDRESS>"

- This is a boilerplate for the bottom of the page we'll return to the client.

property access_time : current date

- Create the variable used to return the current time to the client, initialize to date and time.

- Properties persist from one running of the script to another,
- so this should tell us how long the script has been running,
- too.

```
property counter : 0
```

- Initialize counter variable to zero.

```
on event WWWsdoc
```

- This is the event trigger for the message MacHTTP will send
- to the CGI.

```
set counter to counter + 1
```

- increment counter

```
try -wrap the whole script in an error handler
```

```
return http_10_header & "<TITLE>AppleScript Counter  
CGI</TITLE>" &
```

```
"<H2>Counter CGI</H2><P>" &
```

```
"<P>This script has been run " &
```

```
counter & " times since " & access_time & html_footer
```

- Spit back at the client our http document header, the body
- of the HTML document, our counter and access_time variables,
- and the HTML footer boilerplate.

```
on error msg number num
```

- report the error message and number to the WWW client

```
return http_10_header & "Error " & num & ", " & msg
```

```
end try
```

```
end event WWWsdoc
```

Example 3: Decoding clickable map coordinates

This C program can distinguish whether the right or left side of a graphic is selected (as well as define a default to handle when the graphic isn't selected at all). It shows how a script handles the definitions inside an image map file.

```
/*
```

```
* Copyright (c) 1995 Mike W. Erwin
```

```
* mikee@outer.net
```

```
* All rights reserved.
```

```
*
```

```
* Redistribution and use in source and binary forms, with or
```

```
* without modification, are permitted provided that: (1) source
```

```
* code distributions retain the above copyright notice and this
```

```
* paragraph in its entirety, (2) distributions including binary
```

```
* code include the above copyright notice and this paragraph in
```

```

* its entirety in the documentation or other materials provided
* with the distribution, and (3) all advertising materials
* mentioning features or use of this software display the following
* acknowledgement:
* THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR
* IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION,
  THE IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
* PURPOSE.
*
* x and y coordinates are passed by the Web client on the command
* line, and this program will store them in the argv variable.
* The 'hot spots' will be read from the configuration file
* '/www/conf/ismapper.conf,' which should be in the form:
* <top left coordinates>,<bottom right coordinates> <url>
* with one line for each rectangle, and one line in the form:
* default <url> in case the click does not fall into any of the
* hot spots. Note that there is no space between the comma and
* the coordinates, and whitespace between the coordinates and the
* URL.
*/
/*
*   C CGI used to return a URL to hot spots being clicked on
*   Version 1.0.0
*   2/28/95
*/
/*
*   To compile: "cc -o ismapper ismapper.c"
*/

/*
*   Standard UNIX include files
*/

#include <stdio.h>
#include <ctype.h>

/*
*   Globals
*/

#define MAX_LINE_SIZE 255
#define true 1
#define false 0

```

```

char *author = "mikee@outer.net";
char *config_file = "/www/conf/ismapper.conf";

char URL [ MAX_LINE_SIZE ];
char default_URL [ MAX_LINE_SIZE ];
char global_error [ MAX_LINE_SIZE ];

/*****
 * main()
 *
 * This function takes the x and y mouse down as its arguments and
 * will return a pointer to the start of the URL in the file
 * that matches the boundaries of the rectangle specified in the
 * file.
 *
 * The format of the config file is:
 *
 * 6,6:73,144      http://www.outer.net/ismap.test/left.html
 *
 *****/

char *read_config_file (int x, int y)
{
    FILE *config;
    char *line, new_line[MAX_LINE_SIZE];
    char *url = NULL;
    short found = false;

    if ((config = fopen(config_file, "r")) == NULL) {
        strcpy(global_error, "Configuration file not found");
        return (NULL);
    }

    do {
        int top, left, bottom, right;

        line = fgets(new_line, MAX_LINE_SIZE, config);

        sscanf(new_line, "default%*[\t] %s", default_URL);
        sscanf(new_line, "%d,%d:%d,%d%*[\t] %s", &top, &left,
            &bottom, &right, URL);
    } while (line != NULL);
}

```

```

        if ((x >= top) && (x <= bottom) && (y >= left) &&
            (y <= right))
            found = true;
    }
    while (line && !found);

    if (found)
        url = URL;
    else
        url = default_URL;

    fclose(config);
    return(url);
}

*****
* main()
*
* This is the official start of the C program. It makes a single
* call to our "get_time" function above then formats the output for
* HTML on the standard output
*
*****/

main (int argc, char **argv)
{
    int x_coordinate = 0 , y_coordinate = 0;
    char *destination_url;

    if (argc != 3) {
        printf("Content-Type: text/html\n\n\n");
        printf("<HTML><HEAD><TITLE>CGI Error</TITLE>< HEAD><BODY>\n");
        printf("<H2>Invalid number of arguments</H2>"); exit (1);
    }

    x_coordinate = (int) atoi(argv[1]);
    y_coordinate = (int) atoi(argv[2]);

    destination_url = read_config_file(x_coordinate, y_coordinate);

    if (destination_url != NULL) {
        printf("Location:\t%s\n", destination_url);
    }
    else {

```

```
printf("Content-Type: text/html\n\n");
printf("<HTML><HEAD><TITLE>Ismapper ERROR");
printf("</TITLE></HEAD><BODY>\n");
printf("<H1>No area matched and no default URL is defined.");
printf("%s</H1>\n",global_error);
printf("<HR>Contact %s for help.\n",author);
}
exit(1);
}
```

Installing and Using CGIs

Most Web servers are configured to look for CGI programs in a particular directory that's under the server's control. Normal users (including you) probably won't be able to copy CGI scripts into this directory without obtaining help from their friendly neighborhood systems administrator or WebMaster. Before you can install and use any CGI program on a particular server, you want to talk to these individuals (if not both), tell them what you're trying to do, and ask for their help and advice in making it happen.

Don't be hurt or surprised if this process takes some time: Systems administrators and WebMasters tend to be chronically busy people. You may have to wait a while to get their attention, and then discuss your needs with them. Make your initial approach by e-mail or a phone call, and briefly explain what you want to do. (For example: "I want to include a counter CGI on my home page to track visitors.") Among other things, you may discover that they already have a CGI available for that very purpose on your server, and they'll simply tell you how to include its reference in your CGI invocation of your home page. (You may not have to use our code at all!)

Also, most such individuals are responsible for the safe and proper operation of the server and/or the Web site where you want to run a CGI program. Don't be offended if you hear them say, "Well, I need to look it over first to make sure that it's okay before you can use it." That's because they're only trying to make sure that you're not planning on introducing software that could cripple or compromise their system. So, please, don't take it personally when this happens — these people are just doing their jobs, and asking for a review shows you that they care about what happens on their (and your) server. This is actually a good thing!

Finally, the systems administrator or WebMaster can show you how to install and use CGIs more quickly and efficiently than if you try to figure it out for yourself. So, be ready to wait your turn to get some time and be ready to listen and learn when your turn comes. You won't be disappointed!

As the sample CGI programs in this chapter should illustrate, there are as many ways to skin the proverbial CGI as there are ideas and approaches about how to implement them. We sincerely hope that you can use the paltry tools we include with this chapter, and that you investigate the contents of the diskette that comes with the book. On the disk, you can find C, Perl, and AppleScript versions for all three programs. In the next chapter, we extend our coverage of server-side Web activities, as we investigate search engines, Webcrawlers, and other interesting server-side services.