CHAPTER 22

The Random Banner Generator

OVERVIEW

One of the basic advertising tools developed for the Web is the random banner advertisement. This chapter discusses the random banner generator application, which is used to display a random advertisement within an HTML document to provide a link to the advertiser. In combination with the advertising tracker discussed in Chapter 24, this application allows site administrators to sell advertising space on their Web pages and track the usage for their advertising partners.

The random banner generator application is a "filter" that reads an HTML page, inserts a randomly generated banner in place of a special tag that you define in your HTML document, and then displays it to a client.

INSTALLATION AND USAGE

The random banner generator application should be placed in a CGI-executable directory and expanded into the root directory **Random_banner**. Figure 22.1 outlines the directory structure and the permissions needed for the application to operate.

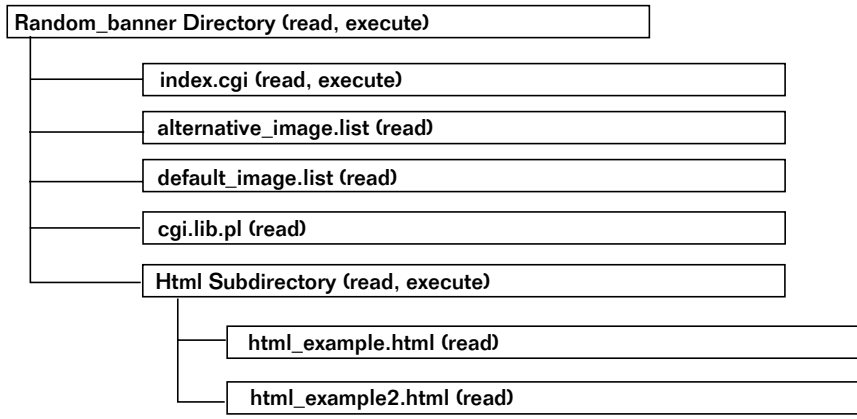

Figure 22.1 *Directory layout for the random banner generator.*

The root **Random_banner** directory must have permissions that allow the Web server to read and execute. It should contain one directory (**Html**) and four text files (**alternative_image.list**, **cgi-lib.pl**, **default_image.list**, and **index.cgi**).

index.cgi is the main application used to output HTML files containing the advertisement banner. The file is called **index.cgi** because Web servers are often designed to automatically execute a file called **index.cgi** if none other has been specified. Also, because this script filters your HTML files, you do not want clients to reference a script such as **random_banner.cgi**. More likely, you want them simply to type in a URL:

```
http://www.foobar.com/
```

Then they are sent directly to the HTML page filtered by the script. Thus, the name **index.cgi** is used to make life simpler and more user-

friendly. It would be fine for you to rename the file, but it must remain readable and executable by the Web server.

alternative_image.list and **default_image.list** are lists of banners and the hyperlink references associated with them. These files are simple text databases with two fields—image location and hyperlink reference—separated by a pipe symbol (|). The text of **alternative_image.list** is shown next:

```
eff_gry_lg.gif|http://www.eff.org/
Icons/ying_yang_icon.gif|http://www.eff.org/~erict/
Icons/yahoo_award_icon.gif|http://www.yahoo.com/
Icons/alerts_bar.gif|http://www.eff.org/pub/Alerts/
```

You can create as many image list datafiles as you want as long as they follow this basic format. These files should be readable by the Web server.

cgi-lib.pl is the library file used to read and parse form data. This file should be readable by the Web server.

Html is a subdirectory that has been included in the accompanying CD-ROM only as an example of a directory for HTML files that use this script. It contains two files—**html_example.html** and **html_example2.html**—which should be readable by the Web server. The directory itself should be readable and executable.

html_example.html is an example of an HTML file that displays a random banner. It should be readable by the Web server. It will be discussed in the "Server-Specific Setup and Options" section.

html_example2.html is another example HTML page.

Server-Specific Setup and Options

There are four variables that are initially defined in the first few lines of **index.cgi**.

`$html_directory_path` is the path of the directory that holds the HTML files to be filtered by this script.

`$default_html_file` is the name of the HTML file to be loaded by default. Usually, you want only one page to display a random advertise-

ment. However, because you may want to have several pages display ads, you must set the default and later override it. Overriding the default is discussed in the "Preparing the HTML File" section.

`$image_url` is the URL of the subdirectory containing the images to be displayed within the HTML page.

`$default_image_list` is the name of the image list that the script uses by default to find images to display if the HTML that calls the script does not include some other list.

`$location_of_cgi_lib` is the location of **cgi-lib.pl**.

Preparing the HTML File

Each HTML file that calls upon **index.cgi** to filter in a random banner must be specially prepared so that **index.cgi** knows where to place the banner. The HTML is tagged with a special line that **index.cgi** will recognize:

```
<!--IMG GOES HERE—>
```

This line should be placed in the location where you want the random banner displayed. The script looks for this exact line, so it must be typed correctly. When the script finds this line, it replaces it with the `<IMG SRC>` and `<A HREF>` tags necessary to create a clickable banner. As an example, the text of **html_example.html** follows:

```
<HTML>
<HEAD>
<TITLE>Random Banner Example</TITLE>
</HEAD>
<BODY BGCOLOR = "FFFFFF" TEXT = "000000">

<CENTER>
<H2>Random Banner Example</H2>
</CENTER>

<BLOCKQUOTE>
This is just a very simple example of how you would reference a random
banner from a basic HTML file.  You will use a normal &LTIMG SRC&GT tag,
which will point to the index.cgi file instead of to an actual graphics
image.  The script will then send back an image that the browser will
put in the appropriate place.REload the screen to get another banner.
```

```
<P>
Below is an example banner:
<P>
<!--IMG GOES HERE-->
<P>
Check out the other
```

Notice also the following link. In this example, we call the second example HTML file using a url-encoded link, which is explained in greater detail in the "Running the Script" section.

```
<A
HREF="index.cgi?image_list=alternative_image.list&html_file=html_exam-
ple2.html">example</A>
</BODY>
</HTML>
```

On the Web, the example HTML file will look like Figure 22.2.

Figure 22.2 *The HTML file with a randomly generated banner.*

Running the Script

Once you have configured the setup variables and the HTML file to the specifics of your server setup, you can try out the random banner generator. To access the script, reference the location of the main script as follows:

```
http://www.foobar.com/cgi-bin/index.cgi
```

This code will run **index.cgi**, which will take advantage of the default image file and default front page. Alternatively, you can use this same script with a different image list and HTML file as follows:

```
http://www.foobar.com/cgi-bin/index.cgi?image_list=xxx&html_file=yyy
```

The `xxx` is the name of the alternative image list, such as **alternative_image.list**, and the `yyy` is the name of the HTML file in which you want the random banner displayed. These two variables are separated with the ampersand (`&`) symbol, and the entire encoded part of the URL is defined after the question mark (`?`) symbol.

Design Discussion

Figure 22.3 summarizes the logic of the script as it responds to the demands of the client.

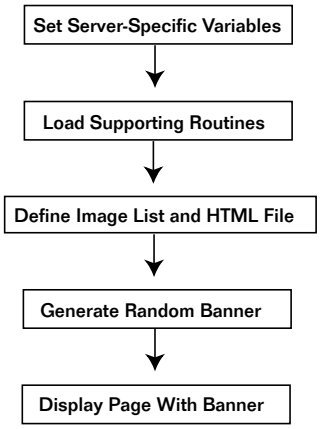

Figure 22.3 *The script logic.*

The script begins by starting the Perl interpreter and printing the HTTP header.

```
#!/usr/local/bin/perl
print "Content-type: text/html\n\n";
```

Defining Server-Specific Variables

Next, a few server-specific variables are defined as discussed in the "Installation and Usage" section.

```
$html_directory_path = "./Html";
$default_html_file =
    "$html_directory_path/html_example.html";
$image_url = "http://www.foobar.com/Images";
$default_image_list = "default_image.list";
$location_of_cgi_lib = "./cgi-lib.pl";
```

Loading the Supporting Routines

Perl's randomizer is then accessed using the `srand` command, which is seeded with the process ID and the time.

```
srand (time|$$);
```

Then **cgi-lib.pl** is loaded and the routine `ReadParse` is used to read and parse any incoming url-encoded data.

```
require "$location_of_cgi_lib ";
&ReadParse(*form_data);
```

Defining the Image List and HTML File

The url-encoded data coming in as form data may include the name of the image file that this script should use to find the locations of the images as well as the hyperlinks associated with them.

If an image file was specified in the URL, the script assigns that value to the variable `$image_database`. Otherwise, the script uses the value of `$default_image_list` as was defined previously.

```
if ($form_data{'image_list'} ne "")
  {
  $image_database = $form_data{'image_list'};
  }
else
  {
  $image_database = "$default_image_list";
  }
```

Similarly, the script determine which HTML file to display with the randomly generated ad. By default, it loads the HTML file defined in `$default_html_file`. However, if the client has specified an alternative file to load in the URL string, the script loads that file instead.

```
 if ($form_data{'html_file'} ne "")
    {
    $html_file = "$html_directory_path/$form_data{'html_file'}";
    }
  else
    {
    $html_file = "$default_html_file";
    }
```

Generating the Random Banner and Associated URL

Next, the script opens the image file, defaulting to `CgiDie` if there is a problem.

```
open (IMAGE_DATABASE, "$image_database") || &CgiDie ("Can't open
$image_database");
```

It then goes through the image list file one line at a time. For every line, it gathers the image and the associated hyperlink by `split`ting the line on the pipe (|) symbol. Then it `push`es the location of the image to the list array `@imagelist`, and the URL associated with the image to `@url_list`. Finally, it closes the image list file.

```
while (<IMAGE_DATABASE>)
  {
  ($image, $url) = split (/\|/, $_);
  push (@imagelist, $image);
  push (@url_list, $url);
  }
close (IMAGE_DATABASE);
```

Now that the script has a list of all the images in the image file, it uses Perl's randomizer function to choose one of them at random. Because `@imagelist` is interpreted by Perl as the number of values in the array, `$random_number` is set to be a random (`rand`) integer value (`int`) from zero to the number of images in the `@imagelist` array.

```
$random_number = int(rand(@imagelist));
```

Then the script takes the number assigned to `$random_number` and accesses the array element in `@imagelist` that is associated with the number. `$random_image` then becomes the name of one of the images in `@imagelist`.

```
$random_image = $imagelist[$random_number];
```

Similarly, `$random_url` is set to the hyperlink associated with the image.

```
$random_url = $url_list[$random_number];
```

Displaying the HTML Page with Banner Inserted

Next, the script opens the HTML file that the client has requested, using `CgiDie` if it cannot be opened for some reason.

```
open (HTML_FILE, "$html_file") || &CgiDie ("Can't open $html_file");
```

The script then reads through the HTML file one line at a time.

```
while (<HTML_FILE>)
{
```

If it comes upon a line that looks like the following,

```
<!—IMG GOES HERE—>
```

the script knows that it is supposed to replace the line with the randomly generated image and the associated hyperlink. So, using the `HERE DOCUMENT` method of printing, it replaces the line with the HTML code.

```
if (/\<!—IMG GOES HERE—\>/)
{
print <<"end_of_html";
<A HREF = "$random_url">
<IMG SRC = "$image_url/$random_image"></A>
end_of_html
}
```


NOTE

Remember that because the greater than (>) and less than (<) symbols are Perl special characters, they must be escaped with a backslash (\) if they are to be treated as patterns to match for.

If the line is not the special tag, the script simply prints the line. Thus, every line in the HTML file is sent to the Web browser except for the special tag line, which is replaced with the `<IMG>` and `<A HREF>` tags.

```
 else
  {
  print "$_";
  }
} # End of while (<HTML_FILE>)
```

Finally, the HTML file is closed and the script exits.

```
close (HTML_FILE);
```