CHAPTER 23

The Fortune Cookie

OVERVIEW

The fortune cookie script goes through a datafile of “fortunes,” chooses one of them at random, and then displays it on the Web. The script can also be configured to loop so that new fortunes are redisplayed automatically at a predefined interval. In this way, the client need not use the browser’s reload option.

This script is a fun addition to any site, because it allows the site administrator to develop a database of short statements to be reloaded every time a client accesses the site. If you have funny or interesting fortunes, it may give the client an extra reason to visit your site a second and third time.

The fortunes can be displayed in either of two ways. You can use Netscape frames to display the messages in one automatically reloading frame along with an HTML page in another frame. Or you can display one fortune at a time on its own page or embedded in another document. The script can also be configured to choose from a variety of datafiles so that you can choose from multiple genres of fortunes for different types of pages.

INSTALLATION AND USAGE

The fortune cookie application expands into the root directory **Fortune_cookie**, which is depicted in Figure 23.1.

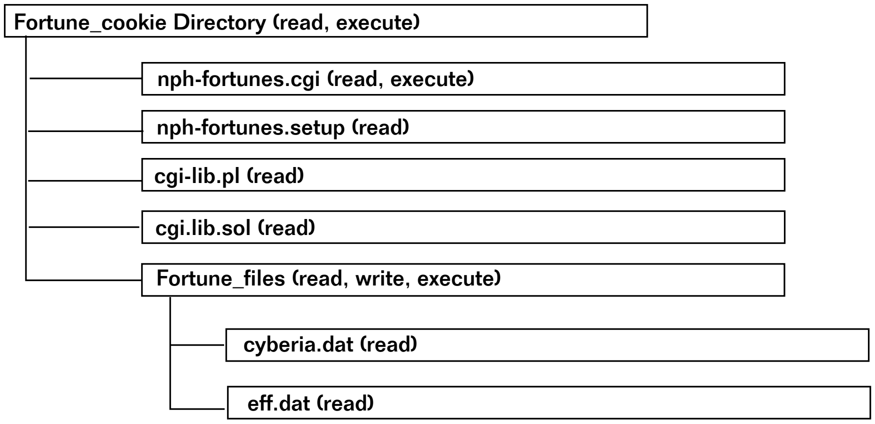

Fortune_cookie Directory (read, execute)
nph-fortunes.cgi (read, execute)
nph-fortunes.setup (read)
cgi-lib.pl (read)
cgi.lib.sol (read)
Fortune_files (read, write, execute)
cyberia.dat (read)
eff.dat (read)

Figure 23.1 *Directory structure for the fortune cookie.*

Fortune_cookie is the root directory for the application. It contains three files (**nph-fortunes.cgi**, **cgi-lib.pl**, and **nph-fortunes.setup**) and one subdirectory (**Fortune_files**) and must be readable and executable by the Web server.

nph-fortunes.cgi, the main body of the program, randomly generates fortunes from the fortunes files. The Web server must have read and execute privileges for this file.

nph-fortunes.setup is the file you use to set up the script for your local installation. The file must be readable by the Web server and will be discussed in the "Server-Specific Setup and Options" section.

cgi-lib.pl is a supporting library file used to parse incoming form data. It must be readable by the Web server.

Fortune_files is a subdirectory that contains the fortune data files from which the script generates random fortunes. Both **eff.dat** and **cyberia.dat** are example fortune files in the accompanying CD-ROM. These files must be readable by the Web server. The subdirectory itself must be readable and executable by the Web server.

Server-Specific Setup and Options

The Setup File

nph-fortunes.cgi requires that you set a few variables in the setup file. These variables configure **nph-fortunes.cgi** to run on your server with your own predefined options.

`$number_of_fortunes_to_display` is the number of fortunes to display before stopping. If this variable is set to 3, for example, the script will generate three fortunes and display them one after another like a slide show. Although you can set this number as high as you want, you may want to limit the slide show to a specific number of fortunes. The **eff.dat** file is huge, with hundreds of fortunes, and you probably don't want the fortunes to be generated forever.

`$default_data_file` is the name of the datafile that **nph-fortunes.cgi** should grab fortunes from if the user does not specify a different one.

`$number_seconds_to_display` should be set to the number of seconds that the script should wait for the user to read the fortune before it loads another one.

`$fortune_file_directory` is the location of the directory that contains your fortune files.

The text of **nph-fortunes.setup** is shown next:

```
$number_of_fortunes_to_display = "10";
$default_data_file = "eff";
$fortune_file_directory = "./Fortune_files";
$number_seconds_to_display = "8";
```

The Fortune File

The accompanying CD-ROM includes both **eff.dat** and **cyberia.dat**, and you can also create your own datafiles from which the fortunes are generated. These files are in a specific format that must be followed exactly.

Each fortune must be separated by two percent (%%) signs, and this same marker must be the last line of the file. Here is an example fortune file:

```
Hello
%%
Goodbye
%%
```

Based on this example, the script would randomly choose either “Hello” or “Goodbye.”

Furthermore, every fortune file must end in the extension **.dat**. This is essential, because **nph-fortunes.cgi** receives the name of the file from client-defined input. Theoretically, if clients had free reign to load any file, they might choose to bypass your fortune files and load “/etc/passwd” instead! We have hard-coded the **.dat** extension to prevent this.

Running the Script

Using the default datafile, the script can be called by this standard hypertext reference:

```
http://www.foobar.com/nph-fortunes.cgi
```

Or it can be called to access a user-defined datafile as follows:

```
http://www.foobar.com/nph-fortunes.cgi?fortune_file=cyberia
```

Notice that we use url-encoding to pass to the script the name of the alternative datafile. This syntax assumes that you place any subsequent fortune files in the **Fortune_files** subdirectory, that they end in the **.dat** extension, and that they are made readable by the Web server.

Design Discussion

The logic of **nph-fortunes.cgi** is depicted in Figure 23.2.

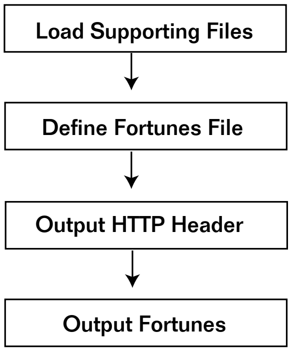


Figure 23.2 *Script logic.*

First, the script starts the Perl interpreter. Also, `srand` is seeded with the process ID and the time so that it will generate random numbers. The script also tells the Perl interpreter to skip buffering so that fortunes will be seamlessly animated.

```
#!/usr/local/bin/perl
srand(time|$$);
$| = 1;
```

Then the script loads the setup file and **cgi-lib.pl** and uses the `ReadParse` subroutine to parse the incoming data.

```
require "cgi-lib.pl";
require "nph-fortunes.setup";
&ReadParse(*form_data);
```

Loading the Fortunes File

Next, the script assigns the incoming client-defined `fortune_file` (if there is one) to the `$fortune_file` variable. If there is no incoming fortune file, the script uses the default value specified in the setup file. Notice that we hard-coded the extension **.dat** for the reasons explained in the "Server-Specific Setup and Options" section.

```
if ($form_data{fortune_file} eq "")
  {
  $fortune_file =
  "$fortune_file_directory/$default_data_file.dat";
  }
else
  {
  $fortune_file =
"$fortune_file_directory/$form_data{'fortune_file'}.dat;
  }
```

Sending the NPH Header

Then the script prints the HTTP header, letting the browser know that it will send multiple documents and that new documents should replace old ones. (The use of NPH headers is covered in greater depth in Chapter 21.) The browser knows that the CGI script is sending a new document when the script sends it the flag –`ARandomString\n`.

```
print "$ENV{'SERVER_PROTOCOL'} 200 OK\n";
print "Server: $ENV{'SERVER_SOFTWARE'}\n";
print "Content-type: multipart/x-mixed-
replace;boundary=ARandomString\n\n";
print "--ARandomString\n";
```

The fortunes are then printed. The following `for` loop counts from zero to the number that the fortune administrator has set for `$number_of_fortunes_to_display`. Thus, for every number from zero to `$number_of_fortunes_to_display`, the script prints the following HTML code as well as the randomly generated fortune (as discussed in the next section).

```
for ($loop = 1;
     $loop <= $number_of_fortunes_to_display;
     $loop++)
  {
  print "Content-type: text/html\n\n";
  print "<HTML><HEAD><TITLE></TITLE></HEAD>
         <BODY BGCOLOR = \"FFFFFF\">";
  print "<BLOCKQUOTE>";
```

Getting a Random Fortune

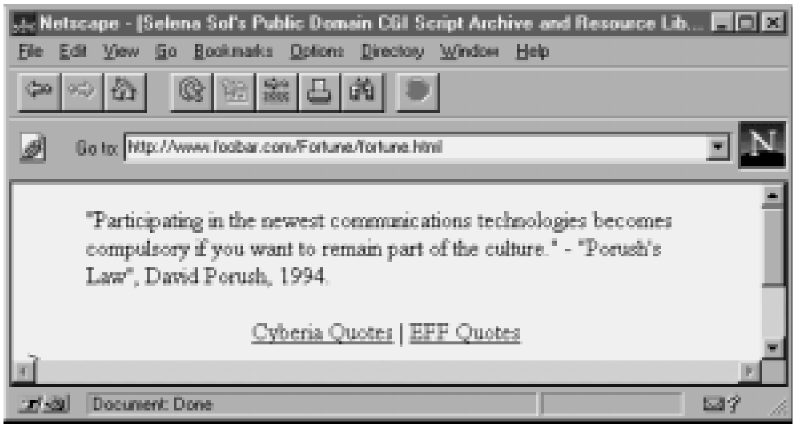
The script first grabs a randomly generated fortune using the subroutine `get_fortune` at the end of this script. The script sends the name of the fortune file to the subroutine as a parameter so that the subroutine will know which fortune file to grab the fortune from.

```
print &get_fortune($fortune_file);
```

Finally, the script prints the HTML footer using the `HERE DOCUMENT` method. In the accompanying CD-ROM, we have created hyperlinks to the sample fortune files.

```
 print <<" end_of_html";
</BLOCKQUOTE>
<CENTER>
<A HREF = "nph-fortunes.cgi?fortune_file=cyberia">Cyberia
Quotes</A> \|
<A HREF = "nph-fortunes.cgi?fortune_file=eff">EFF
Quotes</A>
</CENTER></BODY></HTML>
 end_of_html
```

Figure 23.3 shows a fortune displayed on the Web.


Netscape - [Selena Sol's Public Domain CGI Script Archive and Resource Lib...
File Edit View Go Bookmarks Options Directory Window Help
Go to: http://www.foobar.com/Fortune/fortune.html
Document: Done

"Participating in the newest communications technologies becomes compulsory if you want to remain part of the culture." - "Porush's Law", David Porush, 1994.

Cyberia Quotes | EFF Quotes

Figure 23.3 *A sample fortune.*

The script then pauses while the client reads the fortune. Using the `sleep` function, it waits for as many seconds as the administrator has set for `$number_seconds_to_display`. Then the browser is told that it is about to be sent a new fortune to display, and the script goes on to the next fortune (the `for` loop is incremented).

```
sleep ($number_seconds_to_display);
print "\n—ARandomString\n";
}
```

The get_fortune Subroutine

`get_fortune` is used to gather a random fortune from the fortune file.

```
sub get_fortune
  {
```

The routine begins by assigning to the local variable `$fortune_file` the filename sent from the main routine.

```
local ($fortune_file) = @_;
```

Then it opens that file for reading, defaulting to `CgiDie` if there is a problem.

```
open (FORTUNE_FILE, "$fortune_file")  ||
     &CgiDie ("Can't open $fortune_file");
```

Next, it reads the file one line at a time.

```
while (<FORTUNE_FILE>)
    {
```

If the line it reads is not a double percent (`%%`) followed by a newline (`\n`), however, the script adds the line to a continually growing variable, `$fortune`.

```
if ($_ ne "%%\n")
  {
  $fortune .= "$_";
  }
```

NOTE

".=" means to add to the end of the variable rather than resetting the variable. "$_" is another name for the current line we are reading.

If the line is `%%` followed by a newline (`\n`), the script adds the current value of `$fortune` to the array `@fortunes` and then resets `$fortune`. This process creates a huge array in which each element is a separate fortune. The double percent (`%%`) defines the beginning and end of fortunes.

```
else
  {
  push (@fortunes, $fortune);
  $fortune = "";
  }
}
```

NOTE

Can you see why you must have a double percent (%%) as the last line of the datafile? Otherwise, we would lose the last fortune, because it would never get `push`**ed into the array.**

Once the script goes through all the lines in the fortune file, it is closed.

```
close (FORTUNE_FILE);
```

Then the script returns a randomly selected fortune to the main routine from the array `@fortunes`.

```
 splice(@fortunes,int(rand(@fortunes)),1)."\n";
}
```

In this code, the script figures out how many elements are in `@fortunes`. Fortunately, the scalar `@fortunes` equals the number of elements in the fortunes array; if there are 10 fortunes in the array `@fortunes`, the scalar value of `@fortunes` will equal 10.

Then the script picks a random number from 0 to the number of fortunes in the `@fortunes` array (perhaps 0 to 10), using the function `rand(@fortunes)`.

Because of the way Perl chooses random numbers, the result generated by the `rand` function will be funky such as 5.43245231. So the script rounds off that number to an integer using the `int` function. We can then reference the original `@fortunes` array and pull out a fortune.

Once the script has a random integer from zero to the number of elements in `@fortunes`, it figures out which number in the array `@fortunes` corresponds to the random number and returns that element to the calling routine for printing.