CHAPTER 18

Page Tracking Script

OVERVIEW

The page tracking script follows a user from page to page on a particular site. In addition to tracking which pages a user views, the script records the major “category” of the pages.

By collecting these kinds of statistics, an information provider can determine which pages are accessed more often and by whom. For example, an online store having different categories of items may want to keep track of visitors who often view the electronics section the most so that it can E-mail them selectively with sale information. Similarly, an information provider may want to keep track of a user’s favorite categories so that the provider can program another CGI script to dynamically display content related to the user’s individual interests.

The page tracking script uses the authentication library discussed in Chapter 9 to have the user log in to the system. Once the user logs in, each page can be tracked based on its category.

INSTALLATION AND USAGE

The page tracking files should be installed in a directory called **Page_track**. The files and subdirectories associated with this application, along with their required permissions, are shown in Figure 18.1.

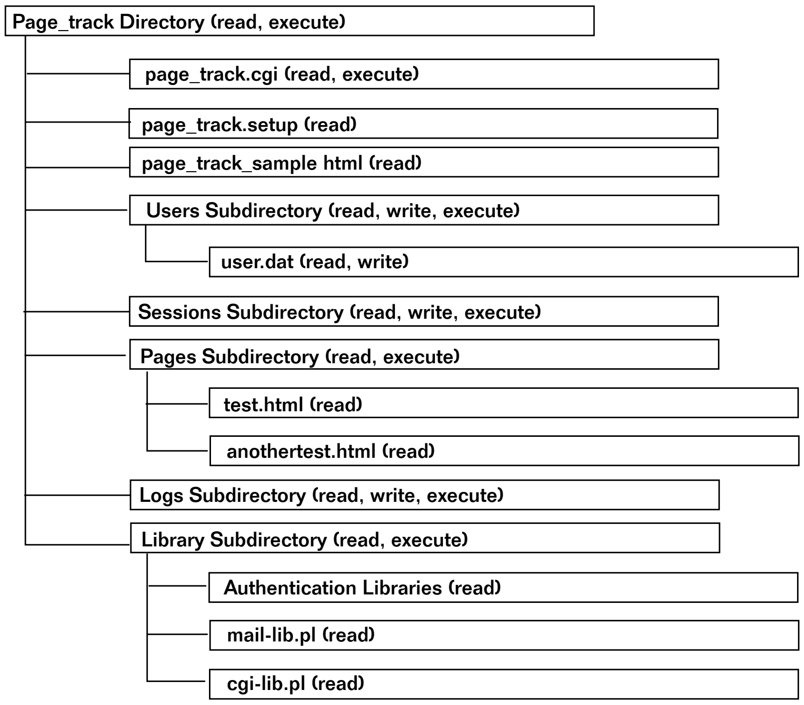

Page_track Directory (read, execute)
page_track.cgi (read, execute)
page_track.setup (read)
page_track_sample html (read)
Users Subdirectory (read, write, execute)
user.dat (read, write)
Sessions Subdirectory (read, write, execute)
Pages Subdirectory (read, execute)
test.html (read)
anothertest.html (read)
Logs Subdirectory (read, write, execute)
Library Subdirectory (read, execute)
Authentication Libraries (read)
mail-lib.pl (read)
cgi-lib.pl (read)

Figure 18.1 *Page tracking script directory structure and permissions.*

Page_track is the root application directory. It must be readable and executable by the Web server. In addition to the application files, the **Logs**, **Pages**, **Library**, **Users**, and **Sessions** subdirectories reside here.

page_track.cgi is the CGI script that filters the HTML and records the user accesses to each file. This file must be readable and executable.

page_track.setup is the setup file for the **page_track.cgi** script. This file must be readable.

page_track_sample.html is a sample usage of the page tracking script. If your server is configured so that HTML files cannot be viewed from a CGI directory, you will need to move this file and update the reference to **page_track.cgi** to point to the new location. This file must be readable.

The **Library** subdirectory stores all the external libraries the script needs to access. This directory should be readable and executable. All the files within **Library** should be readable. These files include **cgi-lib.pl** (used for processing HTML forms), **mail-lib.pl** (used for sending E-mail), and the authentication library files (for logging a user into the page tracking script so that it knows which user it is tracking).

The **Users** subdirectory is used by the authentication library to store the user list. This directory needs to be readable, writable, and executable. The script creates a **user.dat** file in this directory the first time a user registers at the site. This file must be readable and writable.

The **Sessions** subdirectory is used by the authentication library to store the files related to each user's session after he or she first logs on to the site. This directory must be readable, writable, and executable.

The **Pages** subdirectory stores the sample HTML pages included on the accompanying CD-ROM. They contain special tags to tell the page tracker script how to handle the HTML. This directory must be readable and executable. In addition, it must contain page files that are readable.

The **Logs** subdirectory stores the logs related to the tracking of the Web pages. This directory must be readable, executable, and writable. Log files will be created by the **page_track.cgi** program and should be readable and writable by the Web server.

Server-Specific Setup and Options

The **page_track.setup** file contains the page tracker configuration variables. The following is a list of these setup items.

`$page_track_log_file` is the location and name of the file where HTML page accesses are stored. This variable needs to be set relative to the location of the **page_track.cgi** script.

`$page_track_directory` is the directory path where the HTML pages to be filtered are stored. This variable needs to be set relative to the location of the **page_track.cgi** script.

The rest of the variables set up the authentication library. These variables are discussed in great detail in Chapter 9. All the authentication variables start with the `auth_` prefix. By default, the page tracker in this book turns CGI-based authentication on, registration is allowed, and the user gains access to the script right away so that it can begin tracking new users.

The following is an example of all the setup variables in the **page_track.setup** file.

```
$page_track_log_file =  "./Logs/pages.log";
$page_track_directory = "./Pages";
$auth_lib =              "./Library";
$auth_server =               "off";
$auth_cgi =                       "on";
$auth_user_file =          "Users/user.dat";
$auth_alt_user_file =         "";
$auth_allow_register =       "on";
$auth_allow_search =         "on";
$auth_default_group =          "normal";
$auth_generate_password =      "off";
$auth_check_duplicates =       "on";
$auth_add_register =            "on";
$auth_email_register =            "on";
$auth_admin_from_address =     "wwwadmin\@foobar.com";
$auth_admin_email_address =     "gunther\@foobar.com";
$auth_session_length = 2;
$auth_session_dir = "./Sessions";
$auth_register_message =
     "Thanks, you may now log on with your new username
     and password.";
$auth_password_message =
        "Thanks for applying to our site,
        your password is";
@auth_extra_fields = ("auth_first_name",
                      "auth_last_name",
                      "auth_email",
                      "auth_phone");
@auth_extra_desc = ("First Name",
                    "Last Name",
                    "Email",
                    "Phone Number");
```

Running the Script

To use the **page_track.cgi** program, you need to refer to it along with two URL variables: `page` and `sessionid`. `page` is set to the HTML page you want tracked and filtered. Initially, `sessionid` is blank; it is later generated by the authentication library after the user logs in. When an HTML file gets filtered by **page_track.cgi**, the `sessionid` variable is replaced with an exact `sessionid` value assignment so that the user's session is tracked throughout the site. The following is a sample URL for this script if it is installed in a **Page_track** subdirectory under a **cgi-bin** directory:

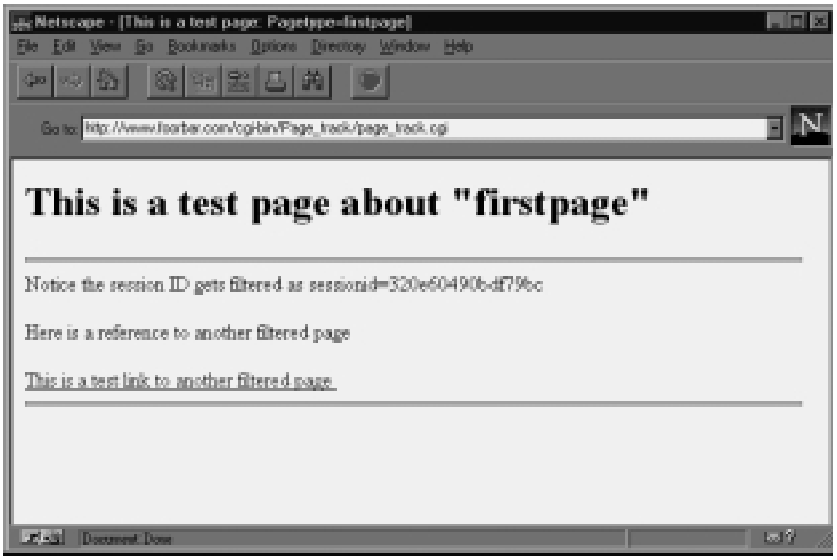
```
http://www.foobar.com/cgi-bin/Page_track/page_track.cgi?file=test.html
```

A sample HTML file that also has a sample version of this link appears in the next section. It also includes a reference to the log file that is generated by the default page tracking application setup that comes with this book. Figure 18.2 shows an example of what it looks like when displayed on a Web browser.

PAGE_TRACK_SAMPLE.HTML

```
<HTML>
<HEAD>
<TITLE>This Is A Test Filter</TITLE>
</HEAD>
<BODY>
<H1>Test Of Filter</H1>
<HR>
<A HREF="page_track.cgi?page=test.html">
Click Here For A Test Of The Filtering
</A>
<P>
<A HREF="Logs/pages.log">
```

```
See The Log Generated So Far By The Test
</A>
<HR>
</BODY>
</HTML>
```


Netscape - [This is a test page: Pagetype=firstpage]
File Edit View Go Bookmarks Options Directory Window Help
Go to: http://www.foobar.com/cgi-bin/Page_track/page_track.cgi
Document: Done

This is a test page about "firstpage"

Notice the session ID gets filtered as sessionid=320e60490b-df79bc

Here is a reference to another filtered page

This is a test link to another filtered page

Figure 18.2 *page_track_sample.html.*

Notice that the hypertext reference in the preceding HTML page to the **page_track.cgi** script will filter **test.html** when selected. You may be wondering what will happen when this page is viewed. To examine this, let's look at the structure of **test.html**.

TEST.HTML

The first line of **test.html** contains an HTML comment field indicated by the <!– and --> tags. This comment is special to **page_track.cgi**. It tells the script what the page type of the HTML document is. Whenever **page_track.cgi** sees an HTML comment at the top of a file with the keyword `pagetype:` flush to the left of the comment, two things happen. First, the script reads the rest of the comment line to see what the page type is, and then the script writes the user and page type information to a log file.

```
<!--pagetype:The First Page—>
<HTML>
<HEAD>
<TITLE>This is a test page </TITLE>
</HEAD>
<BODY>
<H1>This is a test page about "firstpage"</H1>
<HR>
Notice the session ID gets filtered as sessionid
<P>
Here is a reference to another filtered page
<P>
<A HREF=page_track.cgi?page=anothertest.html&sessionid>
This is a test link to another filtered page
</A>
<HR>
</BODY>
</HTML>
```

In addition to this page tracking operation, the **page_track.cgi** script looks for any occurrence of `sessionid` and replaces it with `sessionid=[the real current session id]`. This technique lets the script track the user's current session from page to page without the user having to log on again. When the authentication library receives a valid session ID, it does not have to keep figuring out who the user is. Instead, it can read the information from its previously written session file. This process of storing session information is covered in Chapter 9 in the discussion about the authentication library.

ANOTHERTEST.HTML

The **test.html** file has a reference to another HTML file, **anothertest.html**. This file demonstrates that the **page_track.cgi** script can keep tracking a user through travels in multiple files.

The following HTML file uses the `pagetype:` comment tag to demonstrate yet another page type reference.

```
<!--pagetype:Another Page Type-->
<HTML>
<HEAD>
<TITLE>This is a another test page</TITLE>
</HEAD>
<BODY>
```

```
<H1>This is a test page about "Another Page Type"</H1>
<HR>
</BODY>
</HTML>
```

PAGES.LOG

As a user goes through the previous HTML pages using **page_track.cgi**, a log file is created and appended to with subsequent accesses. A sample of this log file follows. "John Doe" has first visited "The First Page" page type and subsequently has accessed the "Another Page Type" page type.

```
Page Type:The First Page: jdoe,John,Doe,join@foobar.com,555-1234
Page Type:Another Page Type: jdoe,John,Doe,join@foobar.com,555-1234
```

Design Discussion

The purpose of **page_track.cgi** is to track a user's accesses to HTML pages. This aim differs from that of the **html-auth.cgi** application that was used to demonstrate the authentication library in Chapter 9; **html-auth.cgi** protects documents from being viewed, whereas **page_track.cgi** goes one step further, making notes on the types of pages a user likes to go to. Figure 18.3 shows a diagram of the page tracking logic.

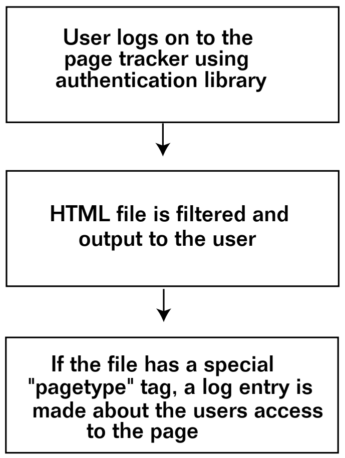

User logs on to the page tracker using authentication library
HTML file is filtered and output to the user
If the file has a special "pagetype" tag, a log entry is made about the users access to the page

Figure 18.3 *Basic flowchart for page tracking.*

The first line of the following code sets up the location of the program's supporting files. By default, `$lib` is set to the **Library** subdirectory under the current directory. **cgi-lib.pl** is loaded along with the **page_track.setup** file.

```
$lib = "./Library";
require "$lib/cgi-lib.pl";
require "./page_track.setup";
```

`$auth_lib` is defined in **page_track.setup**. By default, it is set to the same location as `$lib`. The authentication library, discussed in Chapter 9, is then loaded from the directory specified by `$auth_lib`.

```
require "$auth_lib/auth-lib.pl";
```

First, the standard "Content-type: text/html\n\n" HTTP header is printed using the `PrintHeader` command from **cgi-lib.pl**. In addition, the incoming form variables are processed into the `%in` associative array by `ReadParse`.

```
print &PrintHeader;
&ReadParse;
```

The `$sessionid` variable is read from the incoming form variable `sessionid`. The first time this script is called, `$sessionid` is blank, because the authentication library must first log the user in. This is done when the script calls the `GetSessionInfo` routine.

The conventions for calling `GetSessionInfo` are discussed thoroughly in Chapter 9. For our purposes here, it is enough to know that `GetSessionInfo` gathers the user's username, group, first name, last name, E-mail address, and phone number.

```
$sessionid = $in{"sessionid"};

 ($sessionid, $username, $group, $first_name,
  $last_name, $email,$phone) =
  &GetSessionInfo($sessionid,"page_track.cgi",*in);
```

Next, `$page` is assigned the value of the incoming form variable `page`. This variable is used to determine which page the script needs to process and display to the user.

```
$page = $in{"page"};
```

The `DoFilter` function is called with the user information. This routine does the work of filtering and displaying the HTML file. Then the program exits.

```
&DoFilter($page, $sessionid, $username,
       $first_name, $last_name,
       $email, $phone);
exit;
```

The DoFilter Subroutine

This routine filters the HTML page. It accepts as parameters the page, the session ID, and the user information fields. The page is stored in `$page`, and the session ID is stored in `$session`. The user information fields are stored in the `@fields` array. Each of these fields is defined as a variable that is local to the subroutine.

```
sub DoFilter {
    local($page,$session, @fields) = @_;
```

`$page_type`, `$field_list`, and `$line` are declared as local variables. `$page_type` is used to find out the current page type, and `$field_list` is used to store the user information fields in one long string. `$line` is used to keep track of each line as it is read from the HTML file.

```
    local($page_type,$field_list,$line);
```

First, the file itself is opened. If there is a problem with the file, `CgiDie` is called to print an error message and then stop the program.

```
    open (FILTERPAGE,"<$page_track_directory/$page") ||
     &CgiDie("$page could not be opened\n");
```

`$field_list` is set up as a string containing all the user information fields in a comma-delimited format. This is done using the `join` operator.

```
    $field_list = join(",", @fields);
```

The while loop goes through every line of the HTML file. The program finds a line that contains sessionid. It replaces the line with one that assigns the real session ID value to the $session variable.

```
    while (<FILTERPAGE>) {
     s/sessionid/sessionid=$session/;
```

To keep track of the current line, we stuff it into the $line variable.

```
     $line = $_;
```

If the line has a pagetype: comment, the $page_type is set to the characters that follow pagetype: within the HTML comment block. The parentheses that surround the .* in the regular expression tell Perl to stuff $1 with the results of the pattern match.

```
     if ($line =~ /^<!--pagetype:(.*)-->$/) {
         $page_type = $1;
```

Then the log file is opened, and the previously collected information is written to it.

```
         open(LOGFILE, ">>$page_track_log_file") ||
                      &CgiDie("Could not open log file\n");
         print LOGFILE
                      "Page Type:$page_type: $field_list\n";
         close (LOGFILE);
          }
```

Finally, the line is printed.

```
           print $line;
       }
```

When the file is finished, the current HTML page is closed and the subroutine ends.

```
       close (FILTERPAGE);
} #End of DoFilter
```