Solutions & Examples for PHP Programmers

PHP Cookbook

O'REILLY®

David Sklar & Adam Trachtenberg

PHP Cookbook

David Sklar and Adam Trachtenberg

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

CHAPTER 8

Web Basics

8.0 Introduction

Web programming is probably why you're reading this book. It's why the first version of PHP was written and what continues to make it so popular today. With PHP, it's easy to write dynamic web programs that do almost anything. Other chapters cover various PHP capabilities, like graphics, regular expressions, database access, and file I/O. These capabilities are all part of web programming, but this chapter focuses on some web-specific concepts and organizational topics that will make your web programming stronger.

Recipes 8.1, 8.2, and 8.3 show how to set, read, and delete cookies. A cookie is a small text string that the server instructs the browser to send along with requests the browser makes. Normally, HTTP requests aren't "stateful"; each request can't be connected to a previous one. A cookie, however, can link different requests by the same user. This makes it easier to build features such as shopping carts or to keep track of a user's search history.

Recipe 8.4 shows how to redirect users to a different web page than the one they requested. Recipe 8.5 explains the session module, which lets you easily associate persistent data with a user as he moves through your site. Recipe 8.6 demonstrates how to store session information in a database, which increases the scalability and flexibility of your web site. Discovering the features of a user's browser is shown in Recipe 8.7. Recipe 8.8 shows the details of constructing a URL that includes a GET query string, including proper encoding of special characters and handling of HTML entities.

The next two recipes demonstrate how to use authentication, which lets you protect your web pages with passwords. PHP's special features for dealing with HTTP Basic authentication are explained in Recipe 8.9. Sometimes it's a better idea to roll your own authentication method using cookies, as shown in Recipe 8.10.

The three following recipes deal with output control. Recipe 8.11 shows how to force output to be sent to the browser. Recipe 8.12 explains the output buffering functions. Output buffers enable you to capture output that would otherwise be printed or delay output until an entire page is processed. Automatic compression of output is shown in Recipe 8.13.

Recipes 8.14 to 8.19 cover error handling topics, including controlling where errors are printed, writing custom functions to handle error processing, and adding debugging assistance information to your programs. Recipe 8.18 includes strategies for avoiding the common "headers already sent" error message, such as using the output buffering discussed in Recipe 8.12.

The next four recipes show how to interact with external variables: environment variables and PHP configuration settings. Recipes 8.20 and 8.21 discuss environment variables, while Recipes 8.22 and 8.23 discuss reading and changing PHP configuration settings. If Apache is your web server, you can use the techniques in Recipe 8.24 to communicate with other Apache modules from within your PHP programs.

Recipe 8.25 demonstrates a few methods for profiling and benchmarking your code. By finding where your programs spend most of their time, you can focus your development efforts on improving the code that has the most noticeable speed-up effect to your users.

This chapter also includes two programs that assist in web site maintenance. Program 8.26 validates user accounts by sending an email message with a customized link to each new user. If the user doesn't visit the link within a week of receiving the message, the account is deleted. Program 8.27 monitors requests in real time on a per-user basis and blocks requests from users that flood your site with traffic.

8.1 Setting Cookies

Problem

You want to set a cookie.

Solution

Use `setcookie()`:

```
setcookie('flavor','chocolate chip');
```

Discussion

Cookies are sent with the HTTP headers, so `setcookie()` must be called before any output is generated.

You can pass additional arguments to `setcookie()` to control cookie behavior. The third argument to `setcookie()` is an expiration time, expressed as an epoch timestamp. For example, this cookie expires at noon GMT on December 3, 2004:

```
setcookie('flavor','chocolate chip',1102075200);
```

If the third argument to `setcookie()` is missing (or empty), the cookie expires when the browser is closed. Also, many systems can't handle a cookie expiration time greater than 2147483647, because that's the largest epoch timestamp that fits in a 32-bit integer, as discussed in the introduction to Chapter 3.

The fourth argument to `setcookie()` is a path. The cookie is sent back to the server only when pages whose path begin with the specified string are requested. For example, the following cookie is sent back only to pages whose path begins with */products/*:

```
setcookie('flavor','chocolate chip','','/products/');
```

The page that's setting this cookie doesn't have to have a URL that begins with */products/*, but the following cookie is sent back only to pages that do.

The fifth argument to `setcookie()` is a domain. The cookie is sent back to the server only when pages whose hostname ends with the specified domain are requested. For example, the first cookie in the following code is sent back to all hosts in the *example.com* domain, but the second cookie is sent only with requests to the host *jeannie.example.com*:

```
setcookie('flavor','chocolate chip','','','.example.com');
setcookie('flavor','chocolate chip','','','jeannie.example.com');
```

If the first cookie's domain was just *example.com* instead of *.example.com*, it would be sent only to the single host *example.com* (and not *www.example.com* or *jeannie.example.com*).

The last optional argument to `setcookie()` is a flag that if set to 1, instructs the browser only to send the cookie over an SSL connection. This can be useful if the cookie contains sensitive information, but remember that the data in the cookie is stored in the clear on the user's computer.

Different browsers handle cookies in slightly different ways, especially with regard to how strictly they match path and domain strings and how they determine priority between different cookies of the same name. The `setcookie()` page of the online manual has helpful clarifications of these differences.

See Also

Recipe 8.2 shows how to read cookie values; Recipe 8.3 shows how to delete cookies; Recipe 8.12 explains output buffering; Recipe 8.18 shows how to avoid the "headers already sent" error message that sometimes occurs when calling `setcookie()`; documentation on `setcookie()` at *http://www.php.net/setcookie*; an expanded cookie specification is detailed in RFC 2965 at *http://www.faqs.org/rfcs/rfc2965.html*.

8.2 Reading Cookie Values

Problem

You want to read the value of a cookie that's been previously set.

Solution

Look in the `$_COOKIE` superglobal array:

```
if (isset($_COOKIE['flavor'])) {
    print "You ate a $_COOKIE[flavor] cookie.";
}
```

Discussion

A cookie's value isn't available in `$_COOKIE` during the request in which the cookie is set. In other words, the `setcookie()` function doesn't alter the value of `$_COOKIE`. On subsequent requests, however, each cookie is stored in `$_COOKIE`. If `register_globals` is `on`, cookie values are also assigned to global variables.

When a browser sends a cookie back to the server, it sends only the value. You can't access the cookie's domain, path, expiration time, or secure status through `$_COOKIE` because the browser doesn't send that to the server.

To print the names and values of all cookies sent in a particular request, loop through the `$_COOKIE` array:

```
foreach ($_COOKIE as $cookie_name => $cookie_value) {
    print "$cookie_name = $cookie_value<br>";
}
```

See Also

Recipe 8.1 shows how to set cookies; Recipe 8.3 shows how to delete cookies; Recipe 8.12 explains output buffering; Recipe 8.18 shows how to avoid the "headers already sent" error message that sometimes occurs when calling `setcookie()`; Recipe 9.7 for information on `register_globals`.

8.3 Deleting Cookies

Problem

You want to delete a cookie so a browser doesn't send it back to the server.

Solution

Call `setcookie()` with no value for the cookie and an expiration time in the past:

```
setcookie('flavor','',time()-86400);
```

Discussion

It's a good idea to make the expiration time a few hours or an entire day in the past, in case your server and the user's computer have unsynchronized clocks. For example, if your server thinks it's 3:06 P.M. and a user's computer thinks it's 3:02 P.M., a cookie with an expiration time of 3:05 P.M. isn't deleted by that user's computer even though the time is in the past for the server.

The call to `setcookie()` that deletes a cookie has to have the same arguments (except for value and time) that the call to `setcookie()` that set the cookie did, so include the path, domain, and secure flag if necessary.

See Also

Recipe 8.1 shows how to set cookies; Recipe 8.2 shows how to read cookie values; Recipe 8.12 explains output buffering; Recipe 8.18 shows how to avoid the "headers already sent" error message that sometimes occurs when calling `setcookie()`; documentation on `setcookie()` at *http://www.php.net/setcookie*.

8.4 Redirecting to a Different Location

Problem

You want to automatically send a user to a new URL. For example, after successfully saving form data, you want to redirect a user to a page that confirms the data.

Solution

Before any output is printed, use `header()` to send a `Location` header with the new URL:

```
header('Location: http://www.example.com/');
```

Discussion

If you want to pass variables to the new page, you can include them in the query string of the URL:

```
header('Location: http://www.example.com/?monkey=turtle');
```

The URL that you are redirecting a user to is retrieved with GET. You can't redirect someone to retrieve a URL via POST. You can, however, send other headers along with the `Location` header. This is especially useful with the `Window-target` header, which indicates a particular named frame or window in which to load the new URL:

```
header('Window-target: main');
header('Location: http://www.example.com/');
```

The redirect URL must include the protocol and hostname; it can't just be a pathname:

```
// Good Redirect
header('Location: http://www.example.com/catalog/food/pemmican.php');

// Bad Redirect
header('Location: /catalog/food/pemmican.php');
```

See Also

Documentation on `header()` at *http://www.php.net/header*.

8.5 Using Session Tracking

Problem

You want to maintain information about a user as she moves through your site.

Solution

Use the session module. The `session_start()` function initializes a session, and accessing an element in the global `$_SESSION` array tells PHP to keep track of the corresponding variable.

```
session_start();
$_SESSION['visits']++;
print 'You have visited here '.$_SESSION['visits'].' times.';
```

Discussion

To start a session automatically on each request, set `session.auto_start` to `1` in *php.ini*. With `session.auto_start`, there's no need to call `session_start()`.

The session functions keep track of users by issuing them cookies with a randomly generated session IDs. If PHP detects that a user doesn't accept the session ID

cookie, it automatically adds the session ID to URLs and forms.* For example, consider this code that prints a URL:

```
print '<a href="train.php">Take the A Train</a>';
```

If sessions are enabled, but a user doesn't accept cookies, what's sent to the browser is something like:

```
<a href="train.php?PHPSESSID=2eb89f3344520d11969a79aea6bd2fdd">Take the A Train</a>
```

In this example, the session name is `PHPSESSID` and the session ID is `2eb89f3344520d11969a79aea6bd2fdd`. PHP adds those to the URL so they are passed along to the next page. Forms are modified to include a hidden element that passes the session ID. Redirects with the `Location` header aren't automatically modified, so you have to add a session ID to them yourself using the `SID` constant:

```
$redirect_url = 'http://www.example.com/airplane.php';
if (defined('SID') && (! isset($_COOKIE[session_name()]))) {
    $redirect_url .= '?' . SID;
}

header("Location: $redirect_url");
```

The `session_name()` function returns the name of the cookie that the session ID is stored in, so this code appends the `SID` constant only to `$redirect_url` if the constant is defined, and the session cookie isn't set.

By default, PHP stores session data in files in the */tmp* directory on your server. Each session is stored in its own file. To change the directory in which the files are saved, set the `session.save_path` configuration directive in *php.ini* to the new directory. You can also call `session_save_path()` with the new directory to change directories, but you need to do this before accessing any session variables.

See Also

Documentation on `session_start()` at *http://www.php.net/session-start*, `session_save_path()` at *http://www.php.net/session-save-path*; the session module has a number of configuration directives that help you do things like manage how long sessions can last and how they are cached; these are detailed in the "Sessions" section of the online manual at *http://www.php.net/session*.

* Before PHP 4.2.0, this behavior had to be explicitly enabled by building PHP with the `--enable-trans-sid` configuration setting.

8.6 Storing Sessions in a Database

Problem

You want to store session data in a database instead of in files. If multiple web servers all have access to the same database, the session data is then mirrored across all the web servers.

Solution

Set `session.save_handler` to `user` in *php.ini* and use the `pc_DB_Session` class shown in Example 8-1. For example:

```
$s = new pc_DB_Session('mysql://user:password@localhost/db');
ini_get('session.auto_start') or session_start();
```

Discussion

One of the most powerful aspects of the session module is its abstraction of how sessions get saved. The `session_set_save_handler()` function tells PHP to use different functions for the various session operations such as saving a session and reading session data. The `pc_DB_Session` class stores the session data in a database. If this database is shared between multiple web servers, users' session information is portable across all those web servers. So, if you have a bunch of web servers behind a load balancer, you don't need any fancy tricks to ensure that a user's session data is accurate no matter which web server they get sent to.

To use `pc_DB_Session`, pass a data source name (DSN) to the class when you instantiate it. The session data is stored in a table called `php_session` whose structure is:

```
CREATE TABLE php_session (
  id CHAR(32) NOT NULL,
  data MEDIUMBLOB,
  last_access INT UNSIGNED NOT NULL,
  PRIMARY KEY(id)
)
```

If you want the table name to be different than `php_session`, set `session.save_path` in *php.ini* to your new table name. Example 8-1 shows the `pc_DB_Session` class.

Example 8-1. pc_DB_Session class

```
require 'PEAR.php';
require 'DB.php';

class pc_DB_Session extends PEAR {

    var $_dbh;
    var $_table;
```

Example 8-1. pc_DB_Session class (continued)

```
    var $_connected = false;
    var $_gc_maxlifetime;
    var $_prh_read;
    var $error = null;

    /**
     * Constructor
     */
    function pc_DB_Session($dsn = null) {
        if (is_null($dsn)) {
            $this->error = PEAR::raiseError('No DSN specified');
            return;
        }

        $this->_gc_maxlifetime = ini_get('session.gc_maxlifetime');
        // Sessions last for a day unless otherwise specified.
        if (! $this->_gc_maxlifetime) {
            $this->_gc_maxlifetime = 86400;
        }

        $this->_table = ini_get('session.save_path');
        if ((! $this->_table) || ('/tmp' == $this->_table)) {
            $this->_table = 'php_session';
        }

        $this->_dbh = DB::connect($dsn);
        if (DB::isError($this->_dbh)) {
            $this->error = $this->_dbh;
            return;
        }

        $this->_prh_read = $this->_dbh->prepare(
            "SELECT data FROM $this->_table WHERE id LIKE ? AND last_access >= ?");
        if (DB::isError($this->_prh_read)) {
            $this->error = $this->_prh_read;
            return;
        }

        if (! session_set_save_handler(array(&$this,'_open'),
                                       array(&$this,'_close'),
                                       array(&$this,'_read'),
                                       array(&$this,'_write'),
                                       array(&$this,'_destroy'),
                                       array(&$this,'_gc'))) {
            $this->error = PEAR::raiseError('session_set_save_handler() failed');
            return;
        }

        return $this->_connected = true;
    }
```

Example 8-1. pc_DB_Session class (continued)

```
    function _open() {
        return $this->_connected;
    }

    function _close() {
        return $this->_connected;
    }

    function _read($id) {
        if (! $this->_connected) { return false; }
        $sth =
            $this->_dbh->execute($this->_prh_read,
                                 array($id,time() - $this->_gc_maxlifetime));
        if (DB::isError($sth)) {
            $this->error = $sth;
            return '';
        } else {
            if (($sth->numRows() == 1) &&
                ($ar = $sth->fetchRow(DB_FETCHMODE_ORDERED))) {
                return $ar[0];
            } else {
                return '';
            }
        }
    }

    function _write($id,$data) {
        $sth = $this->_dbh->query(
            "REPLACE INTO $this->_table (id,data,last_access) VALUES (?,?,?)",
            array($id,$data,time()));
        if (DB::isError($sth)) {
            $this->error = $sth;
            return false;
        } else {
            return true;
        }
    }

    function _destroy($id) {
        $sth = $this->_dbh->query("DELETE FROM $this->_table WHERE id LIKE ?",
                                  array($id));
        if (DB::isError($sth)) {
            $this->error = $sth;
            return false;
        } else {
            return true;
        }
    }

    function _gc($maxlifetime) {
        $sth = $this->_dbh->query("DELETE FROM $this->_table WHERE last_access < ?",
                                  array(time() - $maxlifetime));
```

Example 8-1. pc_DB_Session class (continued)

```
        if (DB::isError($sth)) {
            $this->error = $sth;
            return false;
        } else {
            return true;
        }
    }
}
```

The `pc_DB_Session::_write()` method uses a MySQL-specific SQL command, `REPLACE INTO`, which updates an existing record or inserts a new one, depending on whether there is already a record in the database with the given `id` field. If you use a different database, modify the `_write()` function to accomplish the same task. For instance, delete the existing row (if any), and insert a new one, all inside a transaction:

```
        function _write($id,$data) {
            $sth = $this->_dbh->query('BEGIN WORK');
            if (DB::isError($sth)) {
                $this->error = $sth;
                return false;
            }
            $sth = $this->_dbh->query("DELETE FROM $this->_table WHERE id LIKE ?",
                                      array($id));
            if (DB::isError($sth)) {
                $this->error = $sth;
                $this->_dbh->query('ROLLBACK');
                return false;
            }
            $sth = $this->_dbh->query(
                "INSERT INTO $this->_table (id,data,last_access) VALUES (?,?,?)",
                array($id,$data,time()));
            if (DB::isError($sth)) {
                $this->error = $sth;
                $this->_dbh->query('ROLLBACK');
                return false;
            }
            $sth = $this->_dbh->query('COMMIT');
            if (DB::isError($sth)) {
                $this->error = $sth;
                $this->_dbh->query('ROLLBACK');
                return false;
            }
                 return true;
        }
```

See Also

Documentation on `session_set_save_handler()` at *http://www.php.net/session-set-save-handler*; a handler using PostgreSQL is available at *http://www.zend.com/codex.php?id=456&single=1*; the format for data source names is discussed in Recipe 10.3.

8.7 Detecting Different Browsers

Problem

You want to generate content based on the capabilities of a user's browser.

Solution

Use the object returned by `get_browser()` to determine a browser's capabilities:

```
$browser = get_browser();

if ($browser->frames) {
    // print out a frame-based layout
} elseif ($browser->tables) {
    // print out a table-based layout
} else {
    // print out a boring layout
}
```

Discussion

The `get_browser()` function examines the environment variable `$_ENV['HTTP_USER_AGENT']` (set by the web server) and compares it to browsers listed in an external browser capability file. Due to licensing issues, PHP isn't distributed with a browser capability file. The "Obtaining PHP" section of the PHP FAQ (*http://www.php.net/faq.obtaining*) lists *http://www.cyscape.com/asp/browscap/* and *http://www.amrein.com/apps/page.asp?Q=InowDownload* as sources for a browser capabilities file, and there is also one at *http://asp.net.do/browscap.zip*.

Once you download a browser capability file, you need to tell PHP where to find it by setting the `browscap` configuration directive to the pathname of the file. If you use PHP as a CGI, set the directive in the *php.ini* file:

```
browscap=/usr/local/lib/browscap.txt
```

If you use Apache, you need to set the directive in your Apache configuration file:

```
php_value browscap "/usr/local/lib/browscap.txt"
```

Many of the capabilities `get_browser()` finds are shown in Table 8-1. For user-configurable capabilities such as `javascript` or `cookies` though, `get_browser()` just tells you if the browser can support those functions. It doesn't tell you if the user has disabled the functions. If JavaScript is turned off in a JavaScript-capable browser or a user refuses to accept cookies when the browser prompts him, `get_browser()` still indicates that the browser supports those functions.

Table 8-1. Browser capability object properties

Property	Description
`platform`	Operating system the browser is running on (e.g., Windows, Macintosh, UNIX, Win32, Linux, MacPPC)
`version`	Full browser version (e.g., 5.0, 3.5, 6.0b2)
`majorver`	Major browser version (e.g., 5, 3, 6)
`minorver`	Minor browser version (e.g., 0, 5, 02)
`frames`	1 if the browser supports frames
`tables`	1 if the browser supports tables
`cookies`	1 if the browser supports cookies
`backgroundsounds`	1 if the browser supports background sounds with `<embed>` or `<bgsound>`
`vbscript`	1 if the browser supports VBScript
`javascript`	1 if the browser supports JavaScript
`javaapplets`	1 if the browser can run Java applets
`activexcontrols`	1 if the browser can run ActiveX controls

See Also

Documentation on `get_browser()` at *http://www.php.net/get-browser*.

8.8 Building a GET Query String

Problem

You need to construct a link that includes name/value pairs in a query string.

Solution

Encode the names and values with `urlencode()` and use `join()` to create the query string:

```
$vars = array('name' => 'Oscar the Grouch',
              'color' => 'green',
              'favorite_punctuation' => '#');
$safe_vars = array();
foreach ($vars as $name => $value) {
    $safe_vars[] = urlencode($name).'='.urlencode($value);
}

$url = '/muppet/select.php?' . join('&',$safe_vars);
```

Discussion

The URL built in the solution is:

```
/muppet/select.php?name=Oscar+the+Grouch&color=green&favorite_punctuation=%23
```

The query string has spaces encoded as +. Special characters such as # are hex-encoded as %23 because the ASCII value of # is 35, which is 23 in hexadecimal.

Although urlencode() prevents any special characters in the variable names or values from disrupting the constructed URL, you may have problems if your variable names begin with the names of HTML entities. Consider this partial URL for retrieving information about a stereo system:

```
/stereo.php?speakers=12&cdplayer=52&amp=10
```

The HTML entity for ampersand (&) is & so a browser may interpret that URL as:

```
/stereo.php?speakers=12&cdplayer=52&=10
```

To prevent embedded entities from corrupting your URLs, you have three choices. The first is to choose variable names that can't be confused with entities, such as _amp instead of amp. The second is to convert characters with HTML entity equivalents to those entities before printing out the URL. Use htmlentities():

```
$url = '/muppet/select.php?' . htmlentities(join('&',$safe_vars));
```

The resulting URL is:

```
/muppet/select.php?name=Oscar+the+Grouch&color=green&favorite_punctuation=%23
```

Your third choice is to change the argument separator from & to ; by setting the configuration directive arg_separator.input to ;. You then join name-value pairs with ; to produce a query string:

```
/muppet/select.php?name=Oscar+the+Grouch;color=green;favorite_punctuation=%23
```

You may run into trouble with any GET method URLs that you can't explicitly construct with semicolons, such as a form with its method set to GET, because your users' browsers use & as the argument separator.

Because many browsers don't support using ; as an argument separator, the easiest way to avoid problems with entities in URLs is to choose variable names that don't overlap with entity names. If you don't have complete control over variable names, however, use htmlentities() to protect your URLs from entity decoding.

See Also

Documentation on urlencode() at *http://www.php.net/urlencode* and htmlentities() at *http://www.php.net/htmlentities*.

8.9 Using HTTP Basic Authentication

Problem

You want to use PHP to protect parts of your web site with passwords. Instead of storing the passwords in an external file and letting the web server handle the authentication, you want the password verification logic to be in a PHP program.

Solution

The `$_SERVER['PHP_AUTH_USER']` and `$_SERVER['PHP_AUTH_PW']` global variables contain the username and password supplied by the user, if any. To deny access to a page, send a `WWW-Authenticate` header identifying the authentication realm as part of a response with status code 401:

```
header('WWW-Authenticate: Basic realm="My Website"');
header('HTTP/1.0 401 Unauthorized');
echo "You need to enter a valid username and password.";
exit;
```

Discussion

When a browser sees a 401 header, it pops up a dialog box for a username and password. Those authentication credentials (the username and password), if accepted by the server, are associated with the realm in the `WWW-Authenticate` header. Code that checks authentication credentials needs to be executed before any output is sent to the browser, since it might send headers. For example, you can use a function such as `pc_validate()`, shown in Example 8-2.

Example 8-2. pc_validate()

```
function pc_validate($user,$pass) {
    /* replace with appropriate username and password checking,
       such as checking a database */
    $users = array('david' => 'fadj&32',
                   'adam'  => '8HEj838');

    if (isset($users[$user]) && ($users[$user] == $pass)) {
        return true;
    } else {
        return false;
    }
}
```

Here's an example of how to use `pc_validate()`:

```
if (! pc_validate($_SERVER['PHP_AUTH_USER'], $_SERVER['PHP_AUTH_PW'])) {
    header('WWW-Authenticate: Basic realm="My Website"');
    header('HTTP/1.0 401 Unauthorized');
```

```
        echo "You need to enter a valid username and password.";
        exit;
    }
```

Replace the contents of the `pc_validate()` function with appropriate logic to determine if a user entered the correct password. You can also change the realm string from "My Website" and the message that gets printed if a user hits "cancel" in their browser's authentication box from "You need to enter a valid username and password."

HTTP Basic authentication can't be used if you're running PHP as a CGI. If you can't run PHP as a server module, you can use cookie authentication, discussed in Recipe 8.10.

Another issue with HTTP Basic authentication is that it provides no simple way for a user to log out, other then to exit his browser. The PHP online manual has a few suggestions for log out methods that work with varying degrees of success with different server and browser combinations at *http://www.php.net/features.http-auth*.

There is a straightforward way, however, to force a user to log out after a fixed time interval: include a time calculation in the realm string. Browsers use the same username and password combination every time they're asked for credentials in the same realm. By changing the realm name, the browser is forced to ask the user for new credentials. For example, this forces a log out every night at midnight:

```
if (! pc_validate($_SERVER['PHP_AUTH_USER'],$_SERVER['PHP_AUTH_PW'])) {
    $realm = 'My Website for '.date('Y-m-d');
    header('WWW-Authenticate: Basic realm="'.$realm.'"');
    header('HTTP/1.0 401 Unauthorized');
    echo "You need to enter a valid username and password.";
    exit;
}
```

You can also have a user-specific timeout without changing the realm name by storing the time that a user logs in or accesses a protected page. The `pc_validate()` function in Example 8-3 stores login time in a database and forces a log out if it's been more than 15 minutes since the user last requested a protected page.

Example 8-3. pc_validate2()

```
function pc_validate2($user,$pass) {
    $safe_user = strtr(addslashes($user),array('_' => '\_', '%' => '\%'));
    $r = mysql_query("SELECT password,last_access
                      FROM users WHERE user LIKE '$safe_user'");

    if (mysql_numrows($r) == 1) {
        $ob = mysql_fetch_object($r);
        if ($ob->password == $pass) {
            $now = time();
            if (($now - $ob->last_access) > (15 * 60)) {
                return false;
            } else {
```

Example 8-3. pc_validate2() (continued)

```
                // update the last access time
                mysql_query("UPDATE users SET last_access = NOW()
                             WHERE user LIKE '$safe_user'");
               return true;
            }
        }
    } else {
        return false;
    }
}
```

For example:

```
if (! pc_validate($_SERVER['PHP_AUTH_USER'],$_SERVER['PHP_AUTH_PW'])) {
    header('WWW-Authenticate: Basic realm="My Website"');
    header('HTTP/1.0 401 Unauthorized');
    echo "You need to enter a valid username and password.";
    exit;
}
```

See Also

Recipe 8.10; the HTTP Authentication section of the PHP online manual at *http://www.php.net/features.http-auth*.

8.10 Using Cookie Authentication

Problem

You want more control over the user login procedure, such as presenting your own login form.

Solution

Store authentication status in a cookie or as part of a session. When a user logs in successfully, put their username in a cookie. Also include a hash of the username and a secret word so a user can't just make up an authentication cookie with a username in it:

```
$secret_word = 'if i ate spinach';
if (pc_validate($_REQUEST['username'],$_REQUEST['password'])) {
    setcookie('login',
              $_REQUEST['username'].','.md5($_REQUEST['username'].$secret_word));
}
```

Discussion

When using cookie authentication, you have to display your own login form:

```
<form method="post" action="login.php">
Username: <input type="text" name="username"> <br>
Password: <input type="password" name="password"> <br>
<input type="submit" value="Log In">
</form>
```

You can use the same `pc_validate()` function from the Recipe 8.9 to verify the username and password. The only difference is that you pass it `$_REQUEST['username']` and `$_REQUEST['password']` as the credentials instead of `$_SERVER['PHP_AUTH_USER']` and `$_SERVER['PHP_AUTH_PW']`. If the password checks out, send back a cookie that contains a username and a hash of the username, and a secret word. The hash prevents a user from faking a login just by sending a cookie with a username in it.

Once the user has logged in, a page just needs to verify that a valid login cookie was sent in order to do special things for that logged-in user:

```
unset($username);
if ($_COOKIE['login']) {
    list($c_username,$cookie_hash) = split(',',$_COOKIE['login']);
    if (md5($c_username.$secret_word) == $cookie_hash) {
        $username = $c_username;
    } else {
        print "You have sent a bad cookie.";
    }
}

if ($username) {
    print "Welcome, $username.";
} else {
    print "Welcome, anonymous user.";
}
```

If you use the built-in session support, you can add the username and hash to the session and avoid sending a separate cookie. When someone logs in, set an additional variable in the session instead of sending a cookie:

```
if (pc_validate($_REQUEST['username'],$_REQUEST['password'])) {
    $_SESSION['login'] =
        $_REQUEST['username'].','.md5($_REQUEST['username'].$secret_word));
}
```

The verification code is almost the same; it just uses `$_SESSION` instead of `$_COOKIE`:

```
unset($username);
if ($_SESSION['login']) {
    list($c_username,$cookie_hash) = explode(',',$_SESSION['login']);
    if (md5($c_username.$secret_word) == $cookie_hash) {
        $username = $c_username;
```

```
    } else {
        print "You have tampered with your session.";
    }
}
```

Using cookie or session authentication instead of HTTP Basic authentication makes it much easier for users to log out: you just delete their login cookie or remove the login variable from their session. Another advantage of storing authentication information in a session is that you can link users' browsing activities while logged in to their browsing activities before they log in or after they log out. With HTTP Basic authentication, you have no way of tying the requests with a username to the requests that the same user made before they supplied a username. Looking for requests from the same IP address is error-prone, especially if the user is behind a firewall or proxy server. If you are using sessions, you can modify the login procedure to log the connection between session ID and username:

```
if (pc_validate($_REQUEST['username'],$_REQUEST['password'])) {
    $_SESSION['login'] =
        $_REQUEST['username'].','.md5($_REQUEST['username'].$secret_word));
    error_log('Session id '.session_id().' log in as '.$_REQUEST['username']);
}
```

This example writes a message to the error log, but it could just as easily record the information in a database that you could use in your analysis of site usage and traffic.

One danger of using session IDs is that sessions are hijackable. If Alice guesses Bob's session ID, she can masquerade as Bob to the web server. The session module has two optional configuration directives that help you make session IDs harder to guess. The `session.entropy_file` directive contains a path to a device or file that generates randomness, such as */dev/random* or */dev/urandom*. The `session.entropy_length` directive holds the number of bytes to be read from the entropy file when creating session IDs.

No matter how hard session IDs are to guess, they can also be stolen if they are sent in clear text between your server and a user's browser. HTTP Basic authentication also has this problem. Use SSL to guard against network sniffing, as described in Recipe 14.10.

See Also

Recipe 8.9; Recipe 8.17 discusses logging errors; Recipe 14.3 discusses verifying data with hashes; documentation on `setcookie()` at *http://www.php.net/setcookie* and on `md5()` at *http://www.php.net/md5*.

8.11 Flushing Output to the Browser

Problem

You want to force output to be sent to the browser. For example, before doing a slow database query, you want to give the user a status update.

Solution

Use `flush()`:

```
print 'Finding identical snowflakes...';
flush();
$sth = $dbh->query(
    'SELECT shape,COUNT(*) AS c FROM snowflakes GROUP BY shape HAVING c > 1');
```

Discussion

The `flush()` function sends all output that PHP has internally buffered to the web server, but the web server may have internal buffering of its own that delays when the data reaches the browser. Additionally, some browsers don't display data immediately upon receiving it, and some versions of Internet Explorer don't display a page until they've received at least 256 bytes. To force IE to display content, print blank spaces at the beginning of the page:

```
print str_repeat(' ',300);
print 'Finding identical snowflakes...';
flush();
$sth = $dbh->query(
    'SELECT shape,COUNT(*) AS c FROM snowflakes GROUP BY shape HAVING c > 1');
```

See Also

Recipe 18.17; documentation on `flush()` at *http://www.php.net/flush*.

8.12 Buffering Output to the Browser

Problem

You want to start generating output before you're finished sending headers or cookies.

Solution

Call `ob_start()` at the top of your page and `ob_end_flush()` at the bottom. You can then intermix commands that generate output and commands that send headers. The output won't be sent until `ob_end_flush()` is called:

```
<?php ob_start(); ?>

I haven't decided if I want to send a cookie yet.

<?php setcookie('heron','great blue'); ?>

Yes, sending that cookie was the right decision.

<?php ob_end_flush(); ?>
```

Discussion

You can pass `ob_start()` the name of a callback function to process the output buffer with that function. This is useful for postprocessing all the content in a page, such as hiding email addresses from address-harvesting robots:

```
<?php
function mangle_email($s) {
    return preg_replace('/([^@\s]+)@([-a-z0-9]+\.)+[a-z]{2,}/is',
                        '<$1@...>',
                        $s);
}

ob_start('mangle_email');
?>

I would not like spam sent to ronald@example.com!

<?php ob_end_flush(); ?>
```

The `mangle_email()` function transforms the output to:

```
I would not like spam sent to <ronald@...>!
```

The `output_buffering` configuration directive turns output buffering on for all pages:

```
output_buffering = On
```

Similarly, `output_handler` sets an output buffer processing callback to be used on all pages:

```
output_handler=mangle_email
```

Setting an `output_handler` automatically sets `output_buffering` to `on`.

See Also

Recipe 10.10 uses output buffering in a database error logging function; documentation on `ob_start()` at *http://www.php.net/ob-start*, `ob_end_flush()` at *http://www.php.net/ob-end-flush*, and output buffering at *http://www.php.net/outcontrol*.

8.13 Compressing Web Output with gzip

Problem

You want to send compressed content to browsers that support automatic decompression.

Solution

Add this setting to your *php.ini* file:

```
zlib.output_compression=1
```

Discussion

Browsers tell the server that they can accept compressed responses with the `Accept-Encoding` header. If a browser sends `Accept-Encoding: gzip` or `Accept-Encoding: deflate`, and PHP is built with the *zlib* extension, the `zlib.output_compression` configuration directive tells PHP to compress the output with the appropriate algorithm before sending it back to the browser. The browser uncompresses the data before displaying it.

You can adjust the compression level with the `zlib.output_compression_level` configuration directive:

```
; minimal compression
zlib.output_compression_level=1

; maximal compression
zlib.output_compression_level=9
```

At higher compression levels, less data needs to be sent from the server to the browser, but more server CPU time must be used to compress the data.

See Also

Documentation on the *zlib* extension at *http://www.php.net/zlib*.

8.14 Hiding Error Messages from Users

Problem

You don't want PHP error messages visible to users.

Solution

Set the following values in your *php.ini* or web server configuration file:

```
display_errors =off
log_errors     =on
```

These settings tell PHP not to display errors as HTML to the browser but to put them in the server's error log.

Discussion

When `log_errors` is set to `on`, error messages are written to the server's error log. If you want PHP errors to be written to a separate file, set the `error_log` configuration directive with the name of that file:

```
error_log   = /var/log/php.error.log
```

If `error_log` is set to `syslog`, PHP error messages are sent to the system logger using *syslog(3)* on Unix and to the Event Log on Windows NT.

There are lots of error messages you want to show your users, such as telling them they've filled in a form incorrectly, but you should shield your users from internal errors that may reflect a problem with your code. There are two reasons for this. First, these errors appear unprofessional (to expert users) and confusing (to novice users). If something goes wrong when saving form input to a database, check the return code from the database query and display a message to your users apologizing and asking them to come back later. Showing them a cryptic error message straight from PHP doesn't inspire confidence in your web site.

Second, displaying these errors to users is a security risk. Depending on your database and the type of error, the error message may contain information about how to log in to your database or server and how it is structured. Malicious users can use this information to mount an attack on your web site.

For example, if your database server is down, and you attempt to connect to it with `mysql_connect()`, PHP generates the following warning:

```
<br>
<b>Warning</b>:  Can't connect to MySQL server on 'db.example.com' (111) in
<b>/www/docroot/example.php</b> on line <b>3</b><br>
```

If this warning message is sent to a user's browser, he learns that your database server is called *db.example.com* and can mount an attack on it.

See Also

Recipe 8.17 for how to log errors; documentation on PHP configuration directives at *http://www.php.net/configuration*.

8.15 Tuning Error Handling

Problem

You want to alter the error-logging sensitivity on a particular page. This lets you control what types of errors are reported.

Solution

To adjust the types of errors PHP complains about, use `error_reporting()`:

```
error_reporting(E_ALL);               // everything
error_reporting(E_ERROR | E_PARSE);   // only major problems
error_reporting(E_ALL & ~E_NOTICE);   // everything but notices
```

Discussion

Every error generated has an error type associated with it. For example, if you try to `array_pop()` a string, PHP complains that "This argument needs to be an array," since you can only pop arrays. The error type associated with this message is `E_NOTICE`, a nonfatal runtime problem.

By default, the error reporting level is `E_ALL & ~E_NOTICE`, which means all error types except notices. The `&` is a logical `AND`, and the `~` is a logical `NOT`. However, the *php.ini-recommended* configuration file sets the error reporting level to `E_ALL`, which is all error types.

Error messages flagged as notices are runtime problems that are less serious than warnings. They're not necessarily wrong, but they indicate a potential problem. One example of an `E_NOTICE` is "Undefined variable," which occurs if you try to use a variable without previously assigning it a value:

```
// Generates an E_NOTICE
foreach ($array as $value) {
    $html .= $value;
}

// Doesn't generate any error message
$html = '';
```

```
foreach ($array as $value) {
    $html .= $value;
}
```

In the first case, the first time though the `foreach`, `$html` is undefined. So, when you append to it, PHP lets you know you're appending to an undefined variable. In the second case, the empty string is assigned to `$html` above the loop to avoid the `E_NOTICE`. The previous two code snippets generate identical code because the default value of a variable is the empty string. The `E_NOTICE` can be helpful because, for example, you may have misspelled a variable name:

```
foreach ($array as $value) {
    $hmtl .= $value; // oops! that should be $html
}

$html = ''
foreach ($array as $value) {
    $hmtl .= $value; // oops! that should be $html
}
```

A custom error-handling function can parse errors based on their type and take an appropriate action. A complete list of error types is shown in Table 8-2.

Table 8-2. Error types

Value	Constant	Description	Catchable
1	`E_ERROR`	Nonrecoverable error	No
2	`E_WARNING`	Recoverable error	Yes
4	`E_PARSE`	Parser error	No
8	`E_NOTICE`	Possible error	Yes
16	`E_CORE_ERROR`	Like `E_ERROR` but generated by the PHP core	No
32	`E_CORE_WARNING`	Like `E_WARNING` but generated by the PHP core	No
64	`E_COMPILE_ERROR`	Like `E_ERROR` but generated by the Zend Engine	No
128	`E_COMPILE_WARNING`	Like `E_WARNING` but generated by the Zend Engine	No
256	`E_USER_ERROR`	Like `E_ERROR` but triggered by calling `trigger_error()`	Yes
512	`E_USER_WARNING`	Like `E_WARNING` but triggered by calling `trigger_error()`	Yes
1024	`E_USER_NOTICE`	Like `E_NOTICE` but triggered by calling `trigger_error()`	Yes
2047	`E_ALL`	Everything	n/a

Errors labeled catchable can be processed by the function registered using `set_error_handler()`. The others indicate such a serious problem that they're not safe to be handled by users, and PHP must take care of them.

See Also

Recipe 8.16 shows how to set up a custom error handler; documentation on `error_reporting()` at *http://www.php.net/error-reporting* and `set_error_handler()` at *http://www.php.net/set-error-handler*; for more information about errors, see *http://www.php.net/ref.errorfunc.php*.

8.16 Using a Custom Error Handler

Problem

You want to create a custom error handler that lets you control how PHP reports errors.

Solution

To set up your own error function, use `set_error_handler()`:

```
set_error_handler('pc_error_handler');

function pc_error_handler($errno, $error, $file, $line) {
    $message = "[ERROR][$errno][$error][$file:$line]";
    error_log($message);
}
```

Discussion

A custom error handling function can parse errors based on their type and take the appropriate action. See Table 8-2 in Recipe 8.15 for a list of error types.

Pass `set_error_handler()` the name of a function, and PHP forwards all errors to that function. The error handling function can take up to five parameters. The first parameter is the error type, such as `8` for `E_NOTICE`. The second is the message thrown by the error, such as "Undefined variable: html". The third and fourth arguments are the name of the file and the line number in which PHP detected the error. The final parameter is an array holding all the variables defined in the current scope and their values.

For example, in this code `$html` is appended to without first being assigned an initial value:

```
error_reporting(E_ALL);
set_error_handler('pc_error_handler');

function pc_error_handler($errno, $error, $file, $line, $context) {
    $message = "[ERROR][$errno][$error][$file:$line]";
    print "$message";
```

```
    print_r($context);
}

$form = array('one','two');

foreach ($form as $line) {
    $html .= "<b>$line</b>";
}
```

When the "Undefined variable" error is generated, `pc_error_handler()` prints:

```
[ERROR][8][Undefined variable:  html][err-all.php:16]
```

After the initial error message, `pc_error_handler()` also prints a large array containing all the globals, environment, request, and session variables.

Errors labeled catchable in Table 8-2 can be processed by the function registered using `set_error_handler()`. The others indicate such a serious problem that they're not safe to be handled by users and PHP must take care of them.

See Also

Recipe 8.15 lists the different error types; documentation on `set_error_handler()` at *http://www.php.net/set-error-handler*.

8.17 Logging Errors

Problem

You want to write program errors to a log. These errors can include everything from parser errors and files not being found to bad database queries and dropped connections.

Solution

Use `error_log()` to write to the error log:

```
// LDAP error
if (ldap_errno($ldap)) {
    error_log("LDAP Error #" . ldap_errno($ldap) . ": " . ldap_error($ldap));
}
```

Discussion

Logging errors facilitates debugging. Smart error logging makes it easier to fix bugs. Always log information about what caused the error:

```
$r = mysql_query($sql);
if (! $r) {
```

```
    $error = mysql_error();
    error_log('[DB: query @'.$_SERVER['REQUEST_URI']."][$sql]: $error");
} else {
    // process results
}
```

You're not getting all the debugging help you could be if you simply log that an error occurred without any supporting information:

```
$r = mysql_query($sql);
if (! $r) {
    error_log("bad query");
} else {
    // process result
}
```

Another useful technique is to include the `__FILE__` and `__LINE__` constants in your error messages:

```
error_log('['.__FILE__.']['.__LINE__."]: $error");
```

The `__FILE__` constant is the current filename, and `__LINE__` is the current line number.

See Also

Recipe 8.14 for hiding error messages from users; documentation on `error_log()` at *http://www.php.net/error-log*.

8.18 Eliminating "headers already sent" Errors

Problem

You are trying to send a HTTP header or cookie using `header()` or `setcookie()`, but PHP reports a "headers already sent" error message.

Solution

This error happens when you send nonheader output before calling `header()` or `setcookie()`.

Rewrite your code so any output happens after sending headers:

```
// good
setcookie("name", $name);
print "Hello $name!";

// bad
print "Hello $name!";
setcookie("name", $name);
```

```
// good
<?php setcookie("name",$name); ?>
<html><title>Hello</title>
```

Discussion

An HTTP message has a header and a body, which are sent to the client in that order. Once you begin sending the body, you can't send any more headers. So, if you call `setcookie()` after printing some HTML, PHP can't send the appropriate `Cookie` header.

Also, remove trailing whitespace in any include files. When you include a file with blank lines outside `<?php ?>` tags, the blank lines are sent to the browser. Use `trim()` to remove leading and trailing blank lines from files:

```
$file = '/path/to/file.php';

// backup
copy($file, "$file.bak") or die("Can't copy $file: $php_errormsg);

// read and trim
$contents = trim(join('',file($file)));

// write
$fh = fopen($file, 'w')  or die("Can't open $file for writing: $php_errormsg);
if (-1 == fwrite($fh, $contents)) { die("Can't write to $file: $php_errormsg); }
fclose($fh)              or die("Can't close $file: $php_errormsg);
```

Instead of processing files on a one-by-one basis, it may be more convenient to do so on a directory-by-directory basis. Recipe 19.7 describes how to process all the files in a directory.

If you don't want to worry about blank lines disrupting the sending of headers, turn on output buffering. Output buffering prevents PHP from immediately sending all output to the client. If you buffer your output, you can intermix headers and body text with abandon. However, it may seem to users that your server takes longer to fulfill their requests since they have to wait slightly longer before the browser displays any output.

See Also

Recipe 8.12 discusses output buffering; Recipe 19.7 for processing all files in a directory; documentation on `header()` at *http://www.php.net/header*.

8.19 Logging Debugging Information

Problem

You want to make debugging easier by adding statements to print out variables. But, you want to easily be able to switch back and forth from production and debug modes.

Solution

Put a function that conditionally prints out messages based on a defined constant in a page included using the `auto_prepend_file` configuration setting. Save the following code to *debug.php*:

```
// turn debugging on
define('DEBUG',true);

// generic debugging function
function pc_debug($message) {
    if (defined(DEBUG) && DEBUG) {
        error_log($message);
    }
}
```

Set the `auto_prepend_file` directive in *php.ini*:

```
auto_prepend_file=debug.php
```

Now call `pc_debug()` from your code to print out debugging information:

```
$sql = 'SELECT color, shape, smell FROM vegetables';
pc_debug("[sql: $sql]"); // only printed if DEBUG is true
$r = mysql_query($sql);
```

Discussion

Debugging code is a necessary side-effect of writing code. There are a variety of techniques to help you quickly locate and squash your bugs. Many of these involve including scaffolding that helps ensure the correctness of your code. The more complicated the program, the more scaffolding needed. Fred Brooks, in *The Mythical Man-Month*, guesses that there's "half as much code in scaffolding as there is in product." Proper planning ahead of time allows you to integrate the scaffolding into your programming logic in a clean and efficient fashion. This requires you to think out beforehand what you want to measure and record and how you plan on sorting through the data gathered by your scaffolding.

One technique for sifting through the information is to assign different priority levels to different types of debugging comments. Then the debug function prints information only if it's higher than the current priority level.

```
define('DEBUG',2);

function pc_debug($message, $level = 0) {
    if (defined(DEBUG) && ($level > DEBUG) {
        error_log($message);
    }
}

$sql = 'SELECT color, shape, smell FROM vegetables';
pc_debug("[sql: $sql]", 1); // not printed, since 1 < 2
pc_debug("[sql: $sql]", 3); // printed, since 3 > 2
```

Another technique is to write wrapper functions to include additional information to help with performance tuning, such as the time it takes to execute a database query.

```
function getmicrotime(){
    $mtime = microtime();
    $mtime = explode(' ',$mtime);
    return ($mtime[1] + $mtime[0]);
}

function db_query($sql) {
    if (defined(DEBUG) && DEBUG) {
         // start timing the query if DEBUG is on
         $DEBUG_STRING = "[sql: $sql]<br>\n";
         $starttime = getmicrotime();
    }

    $r = mysql_query($sql);

    if (! $r) {
        $error = mysql_error();
        error_log('[DB: query @'.$_SERVER['REQUEST_URI']."][$sql]: $error");
    } elseif (defined(DEBUG) && DEBUG) {
        // the query didn't fail and DEBUG is turned on, so finish timing it
        $endtime = getmicrotime();
        $elapsedtime = $endtime - $starttime;
        $DEBUG_STRING .= "[time: $elapsedtime]<br>\n";
        error_log($DEBUG_STRING);
    }

    return $r;
}
```

Here, instead of just printing out the SQL to the error log, you also record the number of seconds it takes MySQL to perform the request. This lets you see if certain queries are taking too long.

The `getmicrotime()` function converts the output of `microtime()` into a format that allows you to easily perform addition and subtraction upon the numbers.

See Also

Documentation on `define()` at *http://www.php.net/define*, `defined()` at *http://www.php.net/defined*, and `error_log()` at *http://www.php.net/error-log*; *The Mythical Man-Month*, by Frederick P. Brooks (Addison-Wesley).

8.20 Reading Environment Variables

Problem

You want to get the value of an environment variable.

Solution

Read the value from the `$_ENV` superglobal array:

```
$name = $_ENV['USER'];
```

Discussion

Environment variables are named values associated with a process. For instance, in Unix, you can check the value of `$_ENV['HOME']` to find the home directory of a user:

```
print $_ENV['HOME']; // user's home directory
/home/adam
```

Early versions of PHP automatically created PHP variables for all environment variables by default. As of 4.1.0, *php.ini-recommended* disables this because of speed considerations; however *php.ini-dist* continues to enable environment variable loading for backward compatibility.

The `$_ENV` array is created only if the value of the `variables_order` configuration directive contains `E`. If `$_ENV` isn't available, use `getenv()` to retrieve an environment variable:

```
$path = getenv('PATH');
```

The `getenv()` function isn't available if you're running PHP as an ISAPI module.

See Also

Recipe 8.21 on setting environment variables; documentation on `getenv()` at *http://www.php.net/getenv*; information on environment variables in PHP at *http://www.php.net/reserved.variables.php#reserved.variables.environment*.

8.21 Setting Environment Variables

Problem

You want to set an environment variable in a script or in your server configuration. Setting environment variables in your server configuration on a host-by-host basis allows you to configure virtual hosts differently.

Solution

To set an environment variable in a script, use `putenv()`:

```
putenv('ORACLE_SID=ORACLE'); // configure oci extension
```

To set an environment variable in your Apache *httpd.conf* file, use `SetEnv`:

```
SetEnv DATABASE_PASSWORD password
```

Discussion

An advantage of setting variables in *httpd.conf* is that you can set more restrictive read permissions on it than on your PHP scripts. Since PHP files need to be readable by the web-server process, this generally allows other users on the system to view them. By storing passwords in *httpd.conf*, you can avoid placing a password in a publicly available file. Also, if you have multiple hostnames that map to the same document root, you can configure your scripts to behave differently based on the hostnames.

For example, you could have *members.example.com* and *guests.example.com*. The members version requires authentication and allows users additional access. The guests version provides a restricted set of options, but without authentication:

```
$version = $_ENV['SITE_VERSION'];

// redirect to http://guest.example.com, if user fails to sign in correctly
if ('members' == $version) {
    if (!authenticate_user($_REQUEST['username'], $_REQUEST['password'])) {
        header('Location: http://guest.example.com/');
        exit;
    }
}

include_once "${version}_header"; // load custom header
```

See Also

Recipe 8.20 on getting the values of environment variables; documentation on `putenv()` at *http://www.php.net/putenv*; information on setting environment variables in Apache at *http://httpd.apache.org/docs/mod/mod_env.html*.

8.22 Reading Configuration Variables

Problem

You want to get the value of a PHP configuration setting.

Solution

Use `ini_get()`:

```
// find out the include path:
$include_path = ini_get('include_path');
```

Discussion

To get all configuration variable values in one step, call `ini_get_all()`. It returns the variables in an associative array, and each array element is itself an associative array. The second array has three elements: a global value for the setting, a local value, and an access code:

```
// put all configuration variables in an associative array
$vars = ini_get_all();
print_r($vars['include_path']);
Array
(
    [global_value] => .:/usr/local/lib/php/
    [local_value] => .:/usr/local/lib/php/
    [access] => 7
)
```

The `global_value` is the value set from the *php.ini* file; the `local_value` is adjusted to account for any changes made in the web server's configuration file, any relevant *.htaccess* files, and the current script. The value of `access` is a numeric constant representing the places where this value can be altered. Table 8-3 explains the values for `access`. Note that the name `access` is a little misleading in this respect, as the setting's value can always be checked, but not adjusted.

Table 8-3. Access values

Value	PHP constant	Meaning
1	PHP_INI_USER	Any script, using ini_set()
2	PHP_INI_PERDIR	Directory level, using *.htaccess*
4	PHP_INI_SYSTEM	System level, using *php.ini* or *httpd.conf*
7	PHP_INI_ALL	Everywhere: scripts, directories, and the system

A value of 6 means the setting can be changed in both the directory and system level, as 2 + 4 = 6. In practice, there are no variables modifiable only in PHP_INI_USER or PHP_INI_PERDIR, and all variables are modifiable in PHP_INI_SYSTEM, so everything has a value of 4, 6, or 7.

You can also get variables belonging to a specific extension by passing the extension name to ini_get_all():

```
// return just the session module specific variables
$session = ini_get_all('session');
```

By convention, the variables for an extension are prefixed with the extension name and a period. So, all the session variables begin with session. and all the Java variables begin with java., for example.

Since ini_get() returns the current value for a configuration directive, if you want to check the original value from the *php.ini* file, use get_cfg_var():

```
$original = get_cfg_var('sendmail_from'); // have we changed our address?
```

The value returned by get_cfg_var() is the same as what appears in the global_value element of the array returned by ini_get_all().

See Also

Recipe 8.23 on setting configuration variables; documentation on ini_get() at *http://www.php.net/ini-get*, ini_get_all() at *http://www.php.net/ini-get-all*, and get_cfg_var() at *http://www.php.net/get-cfg-var*; a complete list of configuration variables and when they can be modified at *http://www.php.net/function.ini-set.php*.

8.23 Setting Configuration Variables

Problem

You want to change the value of a PHP configuration setting.

Solution

Use `ini_set()`:

```
// add a directory to the include path
ini_set('include_path', ini_get('include_path') . ':/home/fezzik/php');
```

Discussion

Configuration variables are not permanently changed by `ini_set()`. The new value lasts only for the duration of the request in which `ini_set()` is called. To make a persistent modification, alter the values stored in the *php.ini* file.

It isn't meaningful to alter certain variables, such as `asp_tags` or `register_globals` because by the time you call `ini_set()` to modify the setting, it's too late to change the behavior the setting affects. If a variable can't be changed, `ini_set()` returns `false`.

However, it is useful to alter configuration variables in certain pages. For example, if you're running a script from the command line, set `html_errors` to `off`.

To reset a variable back to its original setting, use `ini_restore()`:

```
ini_restore('sendmail_from'); // go back to the default value
```

See Also

Recipe 8.22 on getting values of configuration variables; documentation on `ini_set()` at *http://www.php.net/ini-set* and `ini_restore()` at *http://www.php.net/ini-restore*.

8.24 Communicating Within Apache

Problem

You want to communicate from PHP to other parts of the Apache request process. This includes setting variables in the *access_log*.

Solution

Use `apache_note()`:

```
// get value
$session = apache_note('session');

// set value
apache_note('session', $session);
```

Discussion

When Apache processes a request from a client, it goes through a series of steps; PHP plays only one part in the entire chain. Apache also remaps URLs, authenticates users, logs requests, and more. While processing a request, each handler has access to a set of key/value pairs called the *notes table*. The `apache_note()` function provides access to the notes table to retrieve information set by handlers earlier on in the process and leave information for handlers later on.

For example, if you use the session module to track users and preserve variables across requests, you can integrate this with your log file analysis so you can determine the average number of page views per user. Use `apache_note()` in combination with the logging module to write the session ID directly to the *access_log* for each request:

```
// retrieve the session ID and add it to Apache's notes table
apache_note('session_id', session_id());
```

Then, modify your *httpd.conf* file to add this string to your `LogFormat`:

```
%{session_id}n
```

The trailing `n` tells Apache to use a variable stored in its notes table by another module.

If PHP is built with the `--enable-memory-limit` configuration option, it stores the peak memory usage of each request in a note called `mod_php_memory_usage`. Add the memory usage information to a `LogFormat` with:

```
%{mod_php_memory_usage}n
```

See Also

Documentation on `apache_note()` at *http://www.php.net/apache-note*; information on logging in Apache at *http://httpd.apache.org/docs/mod/mod_log_config.html*.

8.25 Profiling Code

Problem

You have a block of code and you want to profile it to see how long each statement takes to execute.

Solution

Use the PEAR Benchmark module:

```
require 'Benchmark/Timer.php';

$timer =& new Benchmark_Timer(true);
```

```
$timer->start();
// some setup code here
$timer->setMarker('setup');
// some more code executed here
$timer->setMarker('middle');
// even yet still more code here
$timer->setmarker('done');
// and a last bit of code here
$timer->stop();

$timer->display();
```

Discussion

Calling `setMarker()` records the time. The `display()` method prints out a list of markers, the time they were set, and the elapsed time from the previous marker:

```
-------------------------------------------------------------
marker    time index            ex time               perct
-------------------------------------------------------------
Start     1029433375.42507400   -                       0.00%
-------------------------------------------------------------
setup     1029433375.42554800   0.00047397613525391    29.77%
-------------------------------------------------------------
middle    1029433375.42568700   0.00013899803161621     8.73%
-------------------------------------------------------------
done      1029433375.42582000   0.00013303756713867     8.36%
-------------------------------------------------------------
Stop      1029433375.42666600   0.00084602832794189    53.14%
-------------------------------------------------------------
total     -                     0.0015920400619507    100.00%
-------------------------------------------------------------
```

The Benchmark module also includes the `Benchmark_Iterate` class, which can be used to time many executions of a single function:

```
require 'Benchmark/Iterate.php';

$timer =& new Benchmark_Iterate;

// a sample function to time
function use_preg($ar) {
    for ($i = 0, $j = count($ar); $i < $j; $i++) {
        if (preg_match('/gouda/',$ar[$i])) {
            // it's gouda
        }
    }
}

// another sample function to time
function use_equals($ar) {
    for ($i = 0, $j = count($ar); $i < $j; $i++) {
        if ('gouda' == $ar[$i]) {
            // it's gouda
```

```
        }
    }
}

// run use_preg() 1000 times
$timer->run(1000,'use_preg',
               array('gouda','swiss','gruyere','muenster','whiz'));
$results = $timer->get();
print "Mean execution time for use_preg(): $results[mean]\n";

// run use_equals() 1000 times
$timer->run(1000,'use_equals',
               array('gouda','swiss','gruyere','muenster','whiz'));
$results = $timer->get();
print "Mean execution time for use_equals(): $results[mean]\n";
```

The `Benchmark_Iterate::get()` method returns an associative array. The `mean` element of this array holds the mean execution time for each iteration of the function. The `iterations` element holds the number of iterations. The execution time of each iteration of the function is stored in an array element with an integer key. For example, the time of the first iteration is in `$results[1]`, and the time of the 37th iteration is in `$results[37]`.

To automatically record the elapsed execution time after every line of PHP code, use the `declare` construct and the `ticks` directive:

```
function profile($display = false) {
    static $times;

    switch ($display) {
    case false:
        // add the current time to the list of recorded times
        $times[] = microtime();
        break;
    case true:
        // return elapsed times in microseconds
        $start = array_shift($times);

        $start_mt = explode(' ', $start);
        $start_total = doubleval($start_mt[0]) + $start_mt[1];

        foreach ($times as $stop) {
            $stop_mt = explode(' ', $stop);
            $stop_total = doubleval($stop_mt[0]) + $stop_mt[1];
            $elapsed[] = $stop_total - $start_total;
        }

        unset($times);
        return $elapsed;
        break;
    }
}
```

```
// register tick handler
register_tick_function('profile');

// clock the start time
profile();

// execute code, recording time for every statement execution
declare (ticks = 1) {
    foreach ($_SERVER['argv'] as $arg) {
        print strlen($arg);
    }
}

// print out elapsed times
$i = 0;
foreach (profile(true) as $time) {
    $i++;
    print "Line $i: $time\n";
}
```

The `ticks` directive allows you to execute a function on a repeatable basis for a block of code. The number assigned to `ticks` is how many statements go by before the functions that are registered using `register_tick_function()` are executed.

In the previous example, we register a single function and have the `profile()` function execute for every statement inside the `declare` block. If there are two elements in `$_SERVER['argv']`, `profile()` is executed four times: once for each time through the `foreach` loop, and once each time the `print strlen($arg)` line is executed.

You can also set things up to call two functions every three statements:

```
register_tick_function('profile');
register_tick_function('backup');

declare (ticks = 3) {
    // code...
}
```

You can also pass additional parameters into the registered functions, which can be object methods instead of regular functions:

```
// pass "parameter" into profile()
register_tick_function('profile', 'parameter');

// call $car->drive();
$car = new Vehicle;
register_tick_function(array($car, 'drive'));
```

If you want to execute an object method, pass the object and the name of the method in encapsulated within an array. This lets the `register_tick_function()` know you're referring to an object instead of a function.

Call `unregister_tick_function()` to remove a function from the list of tick functions:

```
unregister_tick_function('profile');
```

See Also

http://pear.php.net/package-info.php?package=Benchmark for information on the PEAR Benchmark class; documentation on `register_tick_function()` at *http://www.php.net/register-tick-function*, `unregister_tick_function()` at *http://www.php.net/unregister-tick-function, and declare at http://www.php.net/declare*.

8.26 Program: Website Account (De)activator

When users sign up for your web site, it's helpful to know that they've provided you with a correct email address. To validate the email address they provide, send an email to the address they supply when they sign up. If they don't visit a special URL included in the email after a few days, deactivate their account.

This system has three parts. The first is the *notify-user.php* program that sends an email to a new user and asks them to visit a verification URL, shown in Example 8-4. The second, shown in Example 8-5, is the *verify-user.php* page that handles the verification URL and marks users as valid. The third is the *delete-user.php* program that deactivates accounts of users who don't visit the verification URL after a certain amount of time. This program is shown in Example 8-6.

Here's the SQL to create the table that user information is stored in:

```
CREATE TABLE users (
 email VARCHAR(255) NOT NULL,
 created_on DATETIME NOT NULL,
 verify_string VARCHAR(16) NOT NULL,
 verified TINYINT UNSIGNED
);
```

You probably want to store more information than this about your users, but this is all that's needed to verify them. When creating a user's account, save information to the `users` table, and send the user an email telling them how to verify their account. The code in Example 8-4 assumes that user's email address is stored in the variable `$email`.

Example 8-4. notify-user.php

```
// generate verify_string
$verify_string = '';
for ($i = 0; $i < 16; $i++) {
    $verify_string .= chr(mt_rand(32,126));
}

// insert user into database
if (! mysql_query("INSERT INTO users (email,created_on,verify_string,verified)
    VALUES ('".addslashes($email)."',NOW(),'".addslashes($verify_string)."',0)")) {
    error_log("Can't insert user: ".mysql_error());
    exit;
}
```

Example 8-4. notify-user.php (continued)

```
$verify_string = urlencode($verify_string);
$safe_email = urlencode($email);

$verify_url = "http://www.example.com/verify.php";

$mail_body=<<<_MAIL_
To $email:

Please click on the following link to verify your account creation:

$verify_url?email=$safe_email&verify_string=$verify_string

If you do not verify your account in the next seven days, it will be
deleted.
_MAIL_;

mail($email,"User Verification",$mail_body);
```

The verification page users go to when they follow the link in the email message updates the `users` table if the proper information has been provided, as shown in Example 8-5.

Example 8-5. verify-user.php

```
$safe_email = addslashes($_REQUEST['email']);
$safe_verify_string = addslashes($_REQUEST['verify_string']);

if ($r = mysql_query("UPDATE users SET verified = 1 WHERE email
    LIKE '$safe_email' AND
    verify_string = '$safe_verify_string' AND verified = 0")) {
    if (mysql_affected_rows() == 1) {
        print "Thank you, your account is verified.";
    } else {
        print "Sorry, you could not be verified.";
    }
} else {
    print "Please try again later due to a database error.";
}
```

The user's verification status is updated only if the email address and verify string provided match a row in the database that has not already been verified. The last step is the short program that deletes unverified users after the appropriate interval, as shown in Example 8-6.

Example 8-6. delete-user.php

```
$window = 7; // in days

if ($r = mysql_query("DELETE FROM users WHERE verified = 0 AND
    created_on < DATE_SUB(NOW(),INTERVAL $window DAY)")) {
    if ($deleted_users = mysql_affected_rows()) {
```

Example 8-6. delete-user.php (continued)

```
        print "Deactivated $deleted_users users.\n";
    }
} else {
    print "Can't delete users: ".mysql_error();
}
```

Run this program once a day to scrub the `users` table of users that haven't been verified. If you want to change how long users have to verify themselves, adjust the value of `$window`, and update the text of the email message sent to users to reflect the new value.

8.27 Program: Abusive User Checker

Shared memory's speed makes it an ideal way to store data different web server processes need to access frequently when a file or database would be too slow. Example 8-7 shows the `pc_Web_Abuse_Check` class, which uses shared memory to track accesses to web pages in order to cut off users that abuse a site by bombarding it with requests.

Example 8-7. pc_Web_Abuse_Check class

```
class pc_Web_Abuse_Check {
  var $sem_key;
  var $shm_key;
  var $shm_size;
  var $recalc_seconds;
  var $pageview_threshold;
  var $sem;
  var $shm;
  var $data;
  var $exclude;
  var $block_message;

  function pc_Web_Abuse_Check() {
    $this->sem_key = 5000;
    $this->shm_key = 5001;
    $this->shm_size = 16000;
    $this->recalc_seconds = 60;
    $this->pageview_threshold = 30;

    $this->exclude['/ok-to-bombard.html'] = 1;
    $this->block_message =<<<END
<html>
<head><title>403 Forbidden</title></head>
<body>
<h1>Forbidden</h1>
You have been blocked from retrieving pages from this site due to
abusive repetitive activity from your account. If you believe this
is an error, please contact
```

Example 8-7. pc_Web_Abuse_Check class (continued)

```
<a href="mailto:webmaster@example.com?subject=Site+Abuse">webmaster@example.com</a>.
</body>
</html>
END;
   }

  function get_lock() {
    $this->sem = sem_get($this->sem_key,1,0600);
    if (sem_acquire($this->sem)) {
      $this->shm = shm_attach($this->shm_key,$this->shm_size,0600);
      $this->data = shm_get_var($this->shm,'data');
    } else {
      error_log("Can't acquire semaphore $this->sem_key");
    }
  }

  function release_lock() {
    if (isset($this->data)) {
      shm_put_var($this->shm,'data',$this->data);
    }
    shm_detach($this->shm);
    sem_release($this->sem);
  }

  function check_abuse($user) {
    $this->get_lock();
    if ($this->data['abusive_users'][$user]) {
      // if user is on the list release the semaphore & memory
      $this->release_lock();
      //  serve the "you are blocked" page
      header('HTTP/1.0 403 Forbidden');
      print $this->block_message;
      return true;
    } else {
     // mark this user looking at a page at this time
     $now = time();
     if (! $this->exclude[$_SERVER['PHP_SELF']]) {
       $this->data['user_traffic'][$user]++;
     }
     // (sometimes) tote up the list and add bad people
     if (! $this->data['traffic_start']) {
       $this->data['traffic_start'] = $now;
     } else {
       if (($now - $this->data['traffic_start']) > $this->recalc_seconds) {
         while (list($k,$v) = each($this->data['user_traffic'])) {
           if ($v > $this->pageview_threshold) {
             $this->data['abusive_users'][$k] = $v;
             // log the user's addition to the abusive user list
             error_log("Abuse: [$k] (from ".$_SERVER['REMOTE_ADDR'].')');
           }
         }
         $this->data['traffic_start'] = $now;
```

Example 8-7. pc_Web_Abuse_Check class (continued)

```
            $this->data['user_traffic'] = array();
          }
        }
        $this->release_lock();
      }
      return false;
    }
}
```

To use this class, call its `check_abuse()` method at the top of a page, passing it the username of a logged in user:

```
// get_logged_in_user_name() is a function that finds out if a user is logged in
if ($user = get_logged_in_user_name()) {
    $abuse = new pc_Web_Abuse_Check();
    if ($abuse->check_abuse($user)) {
        exit;
    }
}
```

The `check_abuse()` method secures exclusive access to the shared memory segment in which information about users and traffic is stored with the `get_lock()` method. If the current user is already on the list of abusive users, it releases its lock on the shared memory, prints out an error page to the user, and returns `true`. The error page is defined in the class's constructor.

If the user isn't on the abusive user list, and the current page (stored in `$_SERVER['PHP_SELF']`) isn't on a list of pages to exclude from abuse checking, the count of pages that the user has looked at is incremented. The list of pages to exclude is also defined in the constructor. By calling `check_abuse()` at the top of every page and putting pages that don't count as potentially abusive in the `$exclude` array, you ensure that an abusive user will see the error page even when retrieving a page that doesn't count towards the abuse threshold. This makes your site behave more consistently.

The next section of `check_abuse()` is responsible for adding users to the abusive users list. If more than `$this->recalc_seconds` have passed since the last time it added users to the abusive users list, it looks at each user's pageview count and if any are over `$this->pageview_threshold`, they are added to the abusive users list, and a message is put in the error log. The code that sets `$this->data['traffic_start']` if it's not already set is executed only the very first time `check_abuse()` is called. After adding any new abusive users, `check_abuse()` resets the count of users and pageviews and starts a new interval until the next time the abusive users list is updated. After releasing its lock on the shared memory segment, it returns `false`.

All the information `check_abuse()` needs for its calculations, such as the abusive user list, recent pageview counts for users, and the last time abusive users were calculated, is stored inside a single associative array, `$data`. This makes reading the values from and writing the values to shared memory easier than if the information

was stored in separate variables, because only one call to `shm_get_var()` and `shm_put_var()` are necessary.

The `pc_Web_Abuse_Check` class blocks abusive users, but it doesn't provide any reporting capabilities or a way to add or remove specific users from the list. Example 8-8 shows the *abuse-manage.php* program, which lets you manage the abusive user data.

Example 8-8. abuse-manage.php

```
// the pc_Web_Abuse_Check class is defined in abuse-check.php
require 'abuse-check.php';

$abuse = new pc_Web_Abuse_Check();
$now = time();

// process commands, if any
$abuse->get_lock();
switch ($_REQUEST['cmd']) {
    case 'clear':
      $abuse->data['traffic_start'] = 0;
      $abuse->data['abusive_users'] = array();
      $abuse->data['user_traffic'] = array();
      break;
    case 'add':
      $abuse->data['abusive_users'][$_REQUEST['user']] = 'web @ '.strftime('%c',$now);
      break;
    case 'remove':
      $abuse->data['abusive_users'][$_REQUEST['user']] = 0;
      break;
}
$abuse->release_lock();

// now the relevant info is in $abuse->data

print 'It is now <b>'.strftime('%c',$now).'</b><br>';
print 'Current interval started at <b>'.strftime('%c',$abuse->data['traffic_start']);
print '</b> ('.($now - $abuse->data['traffic_start']).' seconds ago).<p>';

print 'Traffic in the current interval:<br>';
if (count($abuse->data['user_traffic'])) {
  print '<table border="1"><tr><th>User</th><th>Pages</th></tr>';
  while (list($user,$pages) = each($abuse->data['user_traffic'])) {
    print "<tr><td>$user</td><td>$pages</td></tr>";
  }
  print "</table>";
} else {
  print "<i>No traffic.</i>";
}
print '<p>Abusive Users:';

if ($abuse->data['abusive_users']) {
  print '<table border="1"><tr><th>User</th><th>Pages</th></tr>';
  while (list($user,$pages) = each($abuse->data['abusive_users'])) {
```

Example 8-8. abuse-manage.php (continued)

```
    if (0 === $pages) {
      $pages = 'Removed';
      $remove_command = '';
    } else {
      $remove_command =
         "<a href=\"$_SERVER[PHP_SELF]?cmd=remove&user=".urlencode($user)."\">remove</a>";
    }
    print "<tr><td>$user</td><td>$pages</td><td>$remove_command</td></tr>";
  }
  print '</table>';
} else {
  print "<i>No abusive users.</i>";
}

print<<<END
<form method="post" action="$_SERVER[PHP_SELF]">
<input type="hidden" name="cmd" value="add">
Add this user to the abusive users list:
<input type="text" name="user" value="">
<br>
<input type="submit" value="Add User">
</form>
<hr>
<form method="post" action="$_SERVER[PHP_SELF]">
<input type="hidden" name="cmd" value="clear">
<input type="submit" value="Clear the abusive users list">
END;
```

Example 8-8 prints out information about current user page view counts and the current abusive user list, as shown in Figure 8-1. It also lets you add or remove specific users from the list and clear the whole list.

When it removes users from the abusive users list, instead of:

```
unset($abuse->data['abusive_users'][$_REQUEST['user']])
```

it sets the following to 0:

```
$abuse->data['abusive_users'][$_REQUEST['user']]
```

This still causes `check_abuse()` to return `false`, but it allows the page to explicitly note that the user was on the abusive users list but was removed. This is helpful to know in case a user that was removed starts causing trouble again.

When a user is added to the abusive users list, instead of recording a pageview count, the script records the time the user was added. This is helpful in tracking down who or why the user was manually added to the list.

If you deploy `pc_Web_Abuse_Check` and this maintenance page on your server, make sure that the maintenance page is protected by a password or otherwise inaccessible to the general public. Obviously, this code isn't very helpful if abusive users can remove themselves from the list of abusive users.

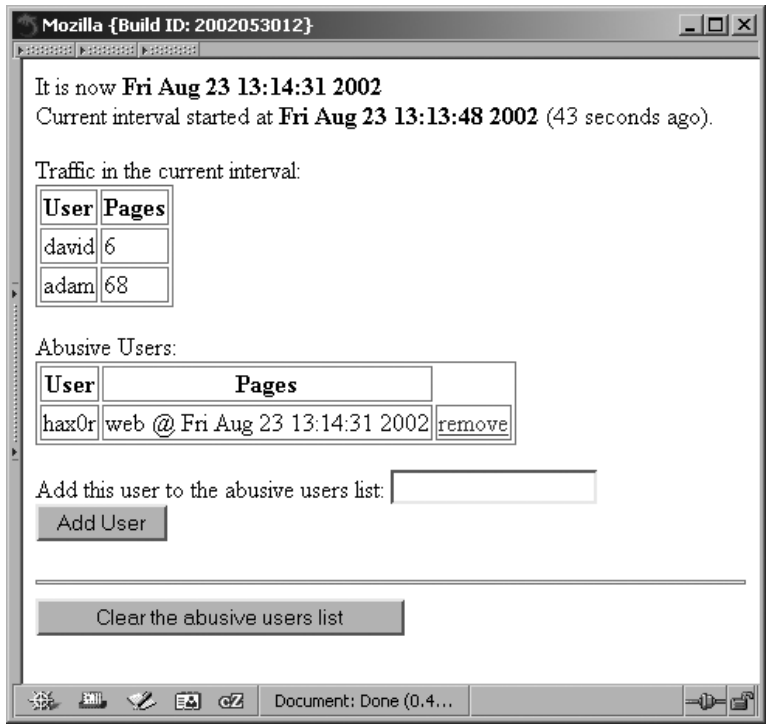

Figure 8-1. Abusive users