PHP ADVANCED CONCEPTS

USING COOKIES:

Before the advent of sessions, there were cookies. Cookies are files that get written to a temporary file on a user’s computer by a web application. Cookies store information that can be read by the online application, thus authenticating a user as unique. By allowing a web application to identify whether a user is unique, the application can then perform login scripts and other functionality.

The problem with cookies is that because they are stored on a user’s computer, they have developed a bad rap as being highly insecure. And because of possible insecurities with cookies, users have begun to turn them off in their browser security settings; in fact, users often do not accept cookies.

Cookies themselves are not bad or insecure if used correctly by a developer. However, since users have the ability to turn them off (and since the actual cookie must be stored on the user’s computer), most good developers have migrated their code to sessions.

Setting Cookies:

To be able to use cookies and store values in them, you must first set a cookie on a user’s computer. You can use plenty of parameters to take full advantage of a cookie, including the expiration time, path of use, name, value, and so on. By using the different parameters, you can customize the way the cookie works for you. The way to set a cookie is by using the function setcookie(), which has the following prototype:

bool setcookie (string name [, string value [, int expire [, string path [, string domain [, bool secure]]]]])

The following table lists the parameters available to you when creating a cookie using setcookie().

Parameter	Description
name	The name to set the cookie variable to and hence the name to access it with
value	The value of the current cookie
expire	When a cookie will expire (in the form of a Unix timestamp)
path	The directory where the cookie will be available for use
domain	The domain at which the cookie will be available
secure	Whether a cookie can be read on a non-SSL enable script

The Code:

```
<?php
//Let's say that the correct login is based on these global user and pass values.
//In the real world, this would be taken from the database most likely.
$GLOBALS['username'] = "test";
$GLOBALS['password'] = "test";
//Here is an example to set a cookie based on a correct login.
function validatelogin ($username, $password){
//Check for a valid match.
if (strcmp ($username, $GLOBALS['username']) == 0➥
&& strcmp ($password, $GLOBALS['password']) == 0){
//If you have a valid match, then you set the cookies.
//This will set two cookies, one named cookie_user set to $cookieuser,
//and another set to cookie_pass, which contains the value of $password
//When storing passwords, it is a good idea to use something like md5() to
//encrypt the stored cookie.
setcookie ("cookie_user", $username, time()+60*60*24*30);
setcookie ("cookie_pass", md5 ($password), time()+60*60*24*30);
return true;
} else {
return false;
}
}
//You call the validatelogin() script.
if (validatelogin ("test", "test")){
echo "Successfully logged in.";
} else {
echo "Sorry, invalid login.";
}
?>
```

In the above program the prototype for md5() is as follows:

string md5 (string str [, bool raw_output])

Reading Cookies:

Cookies can indeed be read—and quite easily. By using the $_COOKIE Superglobal, you can have full access to your cookie for reading and writing to it from your script. The following script allows you to determine if you are properly logged in using a function that returns a true value upon proper validation of login.

The Code:

```
<?php
//Let's say the correct login is based on these global user and pass values.
//In the real world, this would be taken from the database most likely.
$GLOBALS['username'] = "test";
$GLOBALS['password'] = "test";
//Let's assume you already have a valid set of cookies in place.
setcookie ("cookie_user", "test", time()+60*60*24*30);
setcookie ("cookie_pass", md5 ("test"), time()+60*60*24*30);
//Here is an example to set a cookie based on a correct login.
function validatelogin (){
//Check for a valid match.
if (strcmp ($_COOKIE['cookie_user'], $GLOBALS['username']) == 0
&& strcmp ($_COOKIE['cookie_pass'], md5 ($GLOBALS['password'])) == 0){
return true;
} else {
return false;
}
}
//You call the validatelogin() script.
```

```
if (validatelogin ()){
echo "Successfully logged in.";
} else {
echo "Sorry, invalid login.";
}
?>
```

As you can see, using a set of cookies is rather simple; you can simply access them via the $_COOKIE Superglobal. In this case, you compare the (currently) global username and password against the cookies that have been set. If a match is acquired, the unique user is logged in, and the script will remember him until the cookie is expired or until the user physically removes the cookies from their collection.

Deleting Cookies:

Removing cookies is also a simple task. You should note that cookies will disappear by themselves if you have set them up to do so. Cookies that have not been assigned a time to die will simply be removed when the browser window closes. Sometimes, however, a user will want to be able to clear the cookies on a site. Such functionality typically goes by the name of “logout” and is a staple of a well-programmed user interface. The following code allows a user to log out.

The Code:

```
<?php
//Let's assume you already have a valid set of cookies in place.
setcookie ("cookie_user", "test", time()+60*60*24*30);
setcookie ("cookie_pass", md5 ("test"), time()+60*60*24*30);
//Here is a function that will kill the cookies and hence "log out."
function logout (){
//To remove a cookie, you simply set the value of the cookie to blank.
setcookie ("cookie_user", "", time()+60*60*24*30);
setcookie ("cookie_pass", "", time()+60*60*24*30);
}
//You call the logout script.
logout();
//You can no longer access the cookies.
echo $_COOKIE['cookie_user'] . "<br />";
echo "You have successfully logged out.";
?>
```

As you can see, removing cookies is as easy as setting them and leaving the value blank. It is important to remember that when removing the cookies, the parameters passed to the setcookie() function must be identical to the parameters that were passed to it initially. If the parameter list varies from the original, PHP will assume you are trying to remove a different cookie, and the removal will not take place. Once a cookie has been removed, your scripts will no longer have access to it, and the physical cookie itself will have been deleted from your collection.

Writing and Using a Cookie Class:

Cookies should be as easy to use as sessions are. To cut down on some of the more underused functionality that cookies are capable of and make them nice and easy to manage, you can use the following class, which can manage a cookie with the greatest of ease by making instances of a *cookieclass*.

The Code:

```
<?php
//A class to manage a very simple cookie set.
class cookieclass {
private $cookiename;
private $cookievalue;
private $cookieexpiry;
//A function to construct the class.
public function __construct (){
$num_args = func_num_args();
if($num_args > 0){
$args = func_get_args();
$this->cookiename = $args[0];
$this->cookievalue = $args[1];
$this->cookieexpiry = $args[2];
$this->cookieset();
}
}
//The function to actually set a cookie.
public function cookieset (){
try {
if ($this->cookiename != "" && $this->cookievalue != "" && $this->cookieexpiry != ""){
setcookie ($this->cookiename, $this->cookievalue, time() + $this->cookieexpiry);
} else {
throw new exception ("Sorry, you must assign a name and expiry date for the cookie.");
}
} catch (exception $e){
echo $e->getmessage();
}
}
//A function to change the value of the cookie.
public function change ($newvalue){
$_COOKIE[$this->cookiename] = $newvalue;
}
//A function to retrieve the current value of the cookie
public function getvalue (){
return $_COOKIE[$this->cookiename];
}
//A function to remove the cookie.
public function remove (){
$this->change ("");
}
}
//Create a cookie.
$mycookie = new cookieclass ("cookieid","1","60");
```

```
echo $mycookie->getvalue() . "<br />"; //Echoes 1.
$mycookie->change ("Hello World!");
echo $mycookie->getvalue() . "<br />"; //Echoes Hello World!
//Now, you kill off the cookie.
$mycookie->remove();
echo $mycookie->getvalue(); //Outputs nothing as the cookie is dead.
?>
```

USING HTTP HEADERS:

HTTP headers are slightly finicky but rather powerful sets of functionality. The most important aspect to remember about headers is that they can be called only before any output has been written to the web page. If you attempt to call a header after output has been sent to the page, you will generate an error; hence, your script will fail on you.

That being said, the functionality of headers is rather powerful. You can use them to control everything, including setting the current page location, finding out what file format is being displayed, and managing all aspects of the browser cache. In the following examples, you will learn how to use the *header()* function in a variety of ways. The ***header()*** function's prototype is as follows:

void header (string string [, bool replace [, int http_response_code]])

Redirecting to a Different Location:

One of the more common uses for HTTP headers is redirecting a script. By using headers inside processing scripts, you can force the browser to return to any page you want. We prefer to use headers to control exception handling within process scripts. The following script makes sure that all input coming from a form is not blank.

The Code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<title>Sample 12.5</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<form action="sample12_5.php" method="post">
Name: <input type="text" name="yourname" maxlength="150" /><br />
<input type="submit" value="Submit" style="margin-top: 10px;" />
</form>
</body>
</html>
```

The form in the previous block of code will then call the processing statement as follows:

```
<?php
//You will assume that this scripts main focus is to validate against a blank entry.
if (trim ($_POST['yourname']) == ""){
header ("Location: sample12_5.html");
exit;
}
```

```
//If you have a value, then it would do something with said value. Like, say, output it.
echo $_POST['yourname'];
?>
```

The header() function is rather nice in that it will redirect you automatically to the appropriate file (providing it exists) without a single hiccup in the processing. You will simply find yourself at the appropriate page. You can even use the header() function with the Location parameter to send you to a page not currently on the server on which the script is located. As such, this functionality can be rather effective even as a simple page redirection script.

Sending Content Types Other Than HTML:

Naturally, sometimes you will want to use the header() function to output a type of file format that may not be an actual web page. Thankfully, the header function is more than versatile enough to take care of this issue. To make the most out of this function, you can effectively output other file types by simply declaring the content type you want to output.

This functionality can be handy in circumstances where you want to deploy a document to a user or perhaps even output a dynamic image. You can use the following script to output a JPG image to the user.

The Code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<title>Sample 12.6</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<div align="center">
<img src="sample12_6.php" alt="" title="" style="border: none;" />
</div>
</body>
</html>
<?php
//The location of the image.
$path = "images/winter.jpg";
try {
if (is_file ($path)){
if ($file = fopen($path, 'rb')) {
while(!feof($file) and (connection_status()==0)) {
$f .= fread($file, 1024*8);
}
fclose($file);
}
//Use the header function to output an image of .jpg.
header ("Content-type: image/jpeg");
print $f;
} else {
throw new exception ("Sorry, file path is not valid.");
}
} catch (exception $e){
//Create a dynamic error message.
$animage = imagecreate (500, 500);
$red = imagecolorallocate ($animage, 255, 0, 0);
$white = imagecolorallocate ($animage, 255, 255, 255);
imagefilledrectangle ($animage, 0, 0, 500, 500, $white);
imagestring ($animage, 4, ((500 - (strlen($e->getmessage()) ➡
* imagefontwidth(4))) / 2), 5, $e->getmessage(), $red);
imagejpeg ($animage);
header ("Content-type: image/jpeg");
imagedestroy ($animage);
}
?>
```

The following table lists the Common File Format Content Types:

Content Type	Application
application/pdf	Adobe Portable Document Format (PDF) types
application/msword	Microsoft Word documents
application/excel	Microsoft Excel documents
image/gif	GIF Images
image/png	PNG Images
application/octet-stream	Zip files
text/plain	Plain text (text files)

Forcing File "Save As" Downloads:

Because web browsers can output many different file types directly onto the screen, the default when you use headers to output a wide variety of file types is to make them automatically appear on the screen. What if you would rather have the file appear as a download, though? You can use the header() function to force a Save As dialog box to appear for the user to accept a download. The following example uses largely the same code as the previous example but instead forces the user to download the file.

The Code:

```
<?php
//The location of the image.
$path = "images/winter.jpg";
try {
if (is_file ($path)){
if ($file = fopen($path, 'rb')) {
while(!feof($file) and (connection_status()==0)) {
$f .= fread($file, 1024*8);
}
fclose($file);
}
//Use the header function to output an image of .jpg.
$outputname = "myimage";
header ("Content-type: image/jpeg");
//This will force a download.
header("Content-disposition: attachment; filename=".$outputname.".jpg");
print $f;
} else {
throw new exception ("Sorry, file path is not valid.");
}
} catch (exception $e){
echo $e->getmessage();
}
?>
```

The key point in this code is showing content-disposition in the header. By making content-disposition an attachment value, the browser will force a download rather than display the file inline. By using this, you can force the download to appear with any particular filename you prefer and also with pretty much any file extension. By using content-type, you force the browser to output a file of the requested type.

USING SESSIONS:

Because cookies are getting less and less trusted, a means had to be created to allow user authentication without having to store physical files on a remote computer. As a solution, sessions came onto the scene. Considered the best solution for user authentication that allows for script control, sessions store their files on the actual server.

Implementing Sessions:

Sessions are handled much like cookies but with a major difference. While cookies are pretty much declared as global members of the site, a session state must be enabled to use them effectively. While in the session state, sessions can be accessed just like cookies, in a global sense, and can be manipulated, added to, or removed with relative ease.

Setting sessions requires less overhead than creating cookies. Instead of having to completely define how and where a cookie will be in use, with sessions you control most of that through the PHP configuration file. You use sessions in PHP 5 using the $_SESSION Superglobal. You can assign and access a session using the Superglobal, provided the script that is doing the work is within the session state. The following example creates a session state, sets a session, and then outputs the session value.

The Code

```
<?php
//First, create a session states.
session_start();
$GLOBALS['user'] = "test";
$GLOBALS['pass'] = "test";
//Now, here is a function that will log you in.
function login ($username, $password){
if (strcmp ($username, $GLOBALS['user']) == 0 && strcmp ($password, $GLOBALS['pass']) == 0){
$_SESSION['user'] = $username;
$_SESSION['pass'] = md5 ($password);
return true;
} else {
return false;
}
}
//Function to logout.
function logout (){
unset ($_SESSION['user']);
unset ($_SESSION['pass']);
session_destroy();
}
//Now, you can login.
if (login("test", "test")){
//And output our sessions with the greatest of ease.
echo "Successfully logged in with user: " . $_SESSION['user']. " and pass: " . $_SESSION['pass'];
} else {
echo "Could not login.";
}
//Now, you logout.
logout();
//And hence cannot use our sessions anymore.
if (isset ($_SESSION['user'])){
echo $_SESSION['user']; //Outputs nothing.
}
?>
```

The code works quite simply. You create a session state using the session_start() function and then use and access these session values using the $_SESSION Superglobal. Using the Superglobal, you can then add to, remove, or modify the session values. You can use the sessions anywhere the session state is enabled, which means the session_start() function needs to be called at the beginning of every page where you want session access. When you have finished with the sessions, you can simply use the unset() function on the session values and finish off the session state using the session_destroy() function. The prototypes for these session-related functions are as follows:

```
bool session_start ( void )
bool session_destroy ( void )
```

Storing Simple Data Types in Sessions:

Up until PHP 5, short of using a bit of serialization (which is somewhat inconvenient at best), sessions have really been useful only for passing simple data types around. Sessions handle simple data types, and they handle them well. Like any PHP variable, however, the data type of a current session is based upon what was last assigned to it and can be changed quite easily. The following example passes three values by session: an integer, a string, and a floating-point value.

The Code:

```
<?php
//First, create a session states.
session_start();
(int) $_SESSION['integer_value'] = "115";
(string) $_SESSION['string_value'] = "Hello World";
```

```
(float) $_SESSION['float_value'] = "1.07";
//This function exists for the sole purpose of showing how sessions can be called
//from anywhere within the scope of the session state.
function outputsessions (){
echo $_SESSION['integer_value'] . "<br />"; //Outputs 115.
echo $_SESSION['string_value'] . "<br />"; //Outputs Hello World.
echo $_SESSION['float_value'] . "<br />"; //Outputs 1.07.
}
//Then you can call the function from here:
outputsessions();
?>
```

Storing Complex Data Types in Sessions:

One of the major improvements to PHP 5 is the ability to store complex data types within a session. In the past, code that tracked information such as shopping carts had to be stored within temporary database tables and such, which was incredibly clunky and not space efficient.

Fortunately, PHP now allows you to store objects within sessions. Using this technique, you can easily store large quantities of data within a single object (such as a shopping cart object), use the functionality within the session for these purposes, and then pass the data along to other pages. The following example shows how to pass an object and then access the object from a session.

The Code:

```
<?php
//First, create a session states.
session_start();
//A class that does not do too much.
class myclass {
protected $myvalue;
public function setmyvalue ($newvalue){
$this->myvalue = $newvalue;
}
public function getmyvalue (){
return $this->myvalue;
}
}
$_SESSION['myclass_value'] = new myclass ();
//This function exists for the sole purpose of showing how sessions can be called
//from anywhere within the scope of the session state.
function outputsessions (){
$_SESSION['myclass_value']->setmyvalue ("Hello World");
echo $_SESSION['myclass_value']->getmyvalue ();
}
//Then you can call the function from here:
outputsessions();
?>
```

As you can see, the ability to use and set an object through a session variable is now just as simple as doing so with regular data types. This ability will prove to be quite effective in future applications, as web developers can now use the system memory to perform certain functionality rather than wasting space within a database or text/Extensible Markup Language (XML) file.

Detecting Browsers:

To determine the browser version of the user who is currently viewing your site in PHP, several algorithms are at your disposal. The most useful and easiest to implement is the $_SERVER Superglobal. By grabbing the contents of $_SERVER['HTTP_USER_AGENT'], you can retrieve a fairly conclusive string offering of the system that is currently accessing your website. Once you have the string in hand, it is a simple matter of using regular expressions to break down the different parts of the string into something usable. The other way to detect a browser in PHP is through the get_browser() function. Sadly, using this method is not nearly as reliable and involves quite a bit more server configuration.

The following example uses $_SERVER, which should work on just about any PHP 5 platform.

The Code:

```
<?php
//A class to determine a browser and platform type
class browser {
//Our private variables.
private $browseragent;
private $browserversion;
private $browserplatform;
//A function to set the browser agent.
private function setagent($newagent) {
$this->browseragent = $newagent;
}
//A function to set the browser version.
private function setversion($newversion) {
$this->browserversion = $newversion;
}
//A function to set the browser platform.
private function setplatform($newplatform) {
$this->browserplatform = $newplatform;
}
//A function to determine what browser and version you are using.
private function determinebrowser () {
if (ereg('MSIE ([0-9].[0-9]{1,2})',$_SERVER['HTTP_USER_AGENT'],$version)) {
$this->setversion($version[1]);
$this->setagent("IE");
} else if (ereg( 'Opera ([0-9].[0-9]{1,2})',  $_SERVER['HTTP_USER_AGENT'],$version)) {
$this->setversion($version[1]);
$this->setagent("OPERA");
} else if (ereg( 'Mozilla/([0-9].[0-9]{1,2})',  $_SERVER['HTTP_USER_AGENT'],$version)) {
$this->setversion($version[1]);
$this->setagent("MOZILLA");
} else {
```

```
$this->setversion("0");
$this->setagent("OTHER");
}
}
//A function to determine the platform you are on.
private function determineplatform () {
if (strstr ($_SERVER['HTTP_USER_AGENT'],"Win")) {
$this->setplatform("Win");
} else if (strstr ($_SERVER['HTTP_USER_AGENT'],"Mac")) {
$this->setplatform("Mac");
} else if (strstr ($_SERVER['HTTP_USER_AGENT'],"Linux")) {
$this->setplatform("Linux");
} else if (strstr ($_SERVER['HTTP_USER_AGENT'],"Unix")) {
$this->setplatform("Unix");
} else {
$this->setplatform("Other");
}
}
//A function to return the current browser.
public function getbrowser (){
$this->determinebrowser ();
return $this->browseragent . " " . $this->browserversion;
}
//A function to return the current platform.
public function getplatform (){
$this->determineplatform ();
return $this->browserplatform;
}
}
//Now, you simply create a new instance of the browser class.
$mybrowser = new browser ();
//And then you can determine out current browser and platform status.
echo "Browser: " . $mybrowser->getbrowser() . "<br />";
echo "Platform: " . $mybrowser->getplatform() . "<br />";
//The bare bones output looks as such:
echo $_SERVER['HTTP_USER_AGENT'];
?>
```

AUTHENTICATING USERS:

No matter what type of online application you are building, if you need to keep sections of it private, you will at some point need to create a way of authenticating your users so that you know you have a valid user accessing the site. You can handle authentication in a variety of ways, but the two most common methods for securing a file or set of files is through HTTP based authentication and through cookie authentication. Neither is technically superior to the other, and they both have their own uses. Both can be set up dynamically, and both will stop users in their tracks should they not meet the authenticated demands.

Setting Up HTTP-Based Authentication:

HTTP-based authentication can be a true challenge from a scripting point of view. The interesting part about it is that most server interfaces (such as Cpanel or Ensim) can create HTTP based authentication on the fly. In this case, we have written a class to do this for you.

We are not the biggest fans of HTTP-based authentication because the login mechanism is largely the same. You can set a few variables to customize it slightly, but in the end, it is the same pop-up window asking for your username and password. That being said, this class lets you handle the authentication on the fly.

For this code to work properly, you must first set up a file called .htaccess and ensure that you set the proper path to it when calling the class. You must also have a proper password file prepared. Keep in mind that the .htaccess file must also be read and write enabled.

The Code:

```
<?php
//Class to create and maintain http authorization.
class httpauth {
protected $filepath;
protected $passpath;
//A function to construct the class
public function __construct (){
$num_args = func_num_args();
if($num_args > 0){
$args = func_get_args();
$this->filepath = $args[0];
//Check the validity of the file path.
try {
if (is_file ($this->filepath)){
//Validate that the file is named .htaccess
try {
$expfilename = explode ("/", $this->filepath);
if ($expfilename[count($expfilename) - 1] != ".htaccess"){
throw new exception ("Sorry, file must be named .htaccess.");
} else {
try {
//Make sure the file is writable.
if (!is_writable ($this->filepath)){
throw new exception ("File must be writable.");
}
} catch (exception $e){
echo $e->getmessage();
}
}
} catch (exception $e){
echo $e->getmessage();
}
} else {
```

```
throw new exception ("Sorry, file does not exist.");
}
} catch (exception $e){
echo $e->getmessage();
}
//Now, check the validity of the password file.
$this->passpath = $args[1];
try {
if (is_file ($this->passpath)){
//Make sure the file is writable.
try {
if (!is_writable ($this->passpath)){
throw new exception ("Password file must be writable.");
}
} catch (exception $e){
echo $e->getmessage();
}
} else {
throw new exception ("Sorry, password file does not exist.");
}
} catch (exception $e){
echo $e->getmessage();
}
}
}
//Function to add a user to the password file
public function adduser ($user, $pass) {
//Make sure a given user does not already exist.
try {
if ($file = fopen ($this->passpath, "r")){
$proceed = true;
//Run through the file.
while ($input = fgets ($file, 200)){
$exp = explode (":", $input);
//If this user already exists, then you stop right here.
if ($user == $exp[0]){
$proceed = false;
}
}
fclose ($file);
} else {
throw new exception ("Sorry, could not open the  password file for reading.");
}
} catch (exception $e) {
echo $e->getmessage();
```

```
}
try {
//If you are good to go, then write to the file.
if ($proceed){
try {
//Open the password file for appending.
if ($file = fopen ($this->passpath, "a")){
//And then append a new username and password.
fputs($file, $user . ":" . crypt ($pass) . "\n");
fclose($file);
} else {
throw new exception ("Error opening the password file for appending");
}
} catch (exception $e) {
echo $e->getmessage();
}
} else {
throw new exception ("Sorry, this username already exists.");
}
} catch (exception $e){
echo $e->getmessage();
}
}
//Function to add http authorization
public function addauth ($areaname = "Protected Zone") {
//Now, protect the directory.
try {
if ($file = fopen ($this->filepath, "w+")){
fputs($file, "Order allow, deny\n");
fputs($file, "Allow from all\n");
fputs($file, "AuthType Basic\n");
fputs($file, "AuthUserFile " . $this->passpath . "\n\n");
fputs($file, "AuthName \"" . $areaname . "\"\n");
fputs($file, "require valid-user\n");
fclose($file);
} else {
throw new exception ("Sorry, could not open htaccess file for writing.");
}
} catch (exception $e) {
echo $e->getmessage();
}
}
//Function to remove a user from the password listing.
public function removeuser ($user) {
//Run through the current file and get all of the usernames and passwords.
```

```
$userarray = array ();
$passarray = array ();
$arrcounter = 0;
try {
if ($file = fopen ($this->passpath, "r")){
//Run through the file.
while ($input = fgets ($file, 200)){
$exp = explode (":", $input);
//If this user already exists, then you stop right here.
if ($user != $exp[0]){
//Then add to the list.
$userarray[$arrcounter] = $exp[0];
$passarray[$arrcounter] = $exp[1];
$arrcounter++;
}
}
fclose ($file);
} else {
throw new exception ("Sorry, could not open the  password file for reading.");
}
} catch (exception $e) {
echo $e->getmessage();
}
//Then go through the file again and write back all the logins in the array.
try {
if ($file = fopen ($this->passpath, "w")){
//Run through the file.
for ($i = 0; $i < count ($userarray); $i++){
if ($userarray[$i] != "" && $passarray[$i] != ""){
fputs ($file, $userarray[$i] . ":" . $passarray[$i] . "\n");
}
}
fclose ($file);
} else {
throw new exception ("Sorry, could not open the  password file for writing.");
}
} catch (exception $e) {
echo $e->getmessage();
}
}
//Function to change the password of a user.
public function changepass ($user, $newpass){
try {
if ($newpass == ""){
throw new exception ("Sorry, you must supply a new password");
```

```
} else {
$userarray = array ();
$passarray = array ();
$arrcounter = 0;
try {
if ($file = fopen ($this->passpath, "r")){
//Run through the file.
while ($input = fgets ($file, 200)){
$exp = explode (":", $input);
//If you don't have a match you to the array.
if ($user != $exp[0]){
//Then add to the list.
$userarray[$arrcounter] = $exp[0];
$passarray[$arrcounter] = $exp[1];
$arrcounter++;
} else {
//Else you change the pass.
$userarray[$arrcounter] = $exp[0];
$passarray[$arrcounter] = crypt ($newpass);
$arrcounter++;
}
}
fclose ($file);
} else {
throw new exception ("Sorry, could not open the password file for reading.");
}
} catch (exception $e) {
echo $e->getmessage();
}
//Then go through the file again and write back all the logins in the array.
try {
if ($file = fopen ($this->passpath, "w")){
//Run through the file
for ($i = 0; $i < count ($userarray); $i++){
if ($userarray[$i] != "" && $passarray[$i] != ""){
fputs ($file, $userarray[$i] . ":" . $passarray[$i] . "\n");
}
}
fclose ($file);
} else {
throw new exception ("Sorry, could not open the password file for writing.");
}
} catch (exception $e) {
echo $e->getmessage();
}
```

```
}
} catch (exception $e){
echo $e->getmessage();
}
}
//Function to kill the authorization.
public function removeauth () {
unlink ($this->filepath);
}
}
//Set this path to your password file
$passpath = "/home/ensbabin/public_html/php5recipes/chapter12/code/htpasswd";
//Set this path to the folder you want to protect.
$toprotect = "/home/ensbabin/public_html/php5recipes/chapter12/code/foldertoprotect/.htaccess";
//Create a new instance of an httpauth.
$myhttp = new httpauth ($toprotect, $passpath);
//Add user.
$myhttp->adduser ("test", "test");
//Protect a directory.
$myhttp->addauth ("My Protected Zone");
//Add another user.
$myhttp->adduser ("phpauth", "sample");
//Change a user's password.
$myhttp->changepass ("phpauth", "testing");
//Remove a user.
$myhttp->removeuser ("phpauth");
//Remove the protection entirely.
$myhttp->removeauth ();
?>
```

Basically, to set up authentication, you must first set up a username and password that can access the authentication. You can perform this action in this particular script by using the adduser() method. Once you have set up a user, you can then set up authentication on a particular directory using the addauth() method. Any users you have added to the password file can have access to the protected directory.

This class also comes with a few bells and whistles such as the ability to change the password for a user, remove a user entirely, or remove the authentication, but the functionality for the methods speaks for itself.

Setting Up Cookie Authentication:

Managing user authentication through cookies or sessions is a little harder than using HTTP based authentication, but it can ultimately be more flexible and rewarding. Some of the nice features of cookie-based authentication are being able to set your own error messages, being able to control what happens upon login, and being allowed to make your login form blend seamlessly into your application (rather than being forced to use the pop-up boxes of the HTTP-based variety).

Two schools of thought exist on the whole cookie vs. sessions issue; the advantages of sessions being kept on the server side and working on any platform outweigh the cookie method's advantage of being slightly more flexible. By using sessions you will know that your script should work on pretty much any platform and will be a reliable, secure way of handling authentication. You can use the following example as a login system.

The Code:

```
<?php
session_start();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<title>Sample 12.17</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<?php
//Normally your username and pass would be stored in a database.
//For this example you will assume that you have already retrieved them.
$GLOBALS['user'] = "test";
$GLOBALS['pass'] = "test";
//Now, check if you have a valid submission.
if (isset ($_POST['user']) && isset ($_POST['pass'])){
//Then check to see if you have a match.
if (strcmp ($_POST['user'], $GLOBALS['user']) == 0  && strcmp ($_POST['pass'], $GLOBALS['pass']) == 0){
//If you have a valid match, then set the sessions.
$_SESSION['user'] = $_POST['user'];
$_SESSION['pass'] = $_POST['pass'];
} else {
?><div align="center"><p style="color: #FF0000;">Sorry, you have entered an incorrect login.</p></div>
<?php
}
}
//Check if you need to logout.
if ($_POST['logout'] == "yes"){
unset ($_SESSION['user']);
unset ($_SESSION['pass']);
session_destroy();
}
//You then use this function on every page to check for a valid login at all
times.
function checkcookies () {
if (strcmp ($_SESSION['user'], $GLOBALS['user']) == 0  && strcmp ($_SESSION['pass'], $GLOBALS['pass'])
== 0){
return true;
} else {
return false;
}          }
?>
</head>
```

```
<body>
<div align="center">
<?php
//Check if you have a valid login.
if (checkcookies()){
?>
<p>Congratulations, you are logged in!</p>
<form action="sample12_17.html" method="post" style="margin: 0px;">
<input type="hidden" name="logout" value="yes" />
<input type="submit" value="Logout" />
</form>
<?php
//Or else present a login form.
} else {
?>
<form action="sample12_17.html" method="post" style="margin: 0px;">
<div style="width: 500px; margin-bottom: 10px;">
<div style="width: 35%; float: left; text-align: left;">
Username:
</div>
<div style="width: 64%; float: right; text-align: left;">
<input type="text" name="user" maxlength="25" />
</div>
<br style="clear: both;" />
</div>
<div style="width: 500px; margin-bottom: 10px;">
<div style="width: 35%; float: left; text-align: left;">
Password:
</div>
<div style="width: 64%; float: right; text-align: left;">
<input type="password" name="pass" maxlength="25" />
</div>
<br style="clear: both;" />
</div>
<div style="width: 500px; text-align: left;"><input type="submit" value="Login" /></div>
</form>
<?php
}
?>
</div>
</body>
</html>
```

Basically, you are running the entire login algorithm from this one script. If the script detects that you have submitted a username and password, it will then check for a valid match and set proper sessions upon the match. If the system detects that the sessions are already in place and are proper (as handled by the checkcookies() function), it does not display the login form and instead displays a means to log out. The logout algorithm is handled in mostly the same way. If the script detects a logout field is in place, it then goes through the algorithm to kill off the session variables.

USING ENVIRONMENT AND CONFIGURATION VARIABLES:

PHP provides a means to use and verify the configuration settings and environment variables relative to the server space the script is occupying. Having access to this feature set can come in handy on many occasions. By having access to environment variables, you can customize your scripts to work optimally on the platform that is available. By having access to the configuration variables of PHP, you can customize the PHP environment your script is working in for special occurrences.

A common use of the environment variables in PHP is for dynamic imaging. While Windows systems commonly store their fonts in one folder, Linux-based systems keep theirs in another. By using PHP's environment variables to determine the current operating system, you can make your code slightly more portable.

Using configuration variables can also come in quite handy, particularly with file upload scripts. The base PHP installation leaves only enough processing time to upload files that are generally 2MB or smaller in size. By manipulating the PHP configuration files temporarily, you can increase the limit enough to allow a script to process much larger files.

Reading Environment and Configuration Variables:

PHP 5 makes reading environment and configuration variables easy. The $_ENV Superglobal is the PHP method for reading system's environment variables and has an argument set that is based upon the current environment, that is available to it. Because of its relative flexibility, there is no real set argument list, as it is generated based on the current server environment.

You can use the ***phpinfo()*** function to determine the current environment variables, and you can retrieve them using the ***getenv()*** function, which needs to be supplied a valid environment variable name.

Reading configuration variables, on the other hand, takes place through two functions, ini_get() and ***ini_get_all()***. The function ***ini_get()*** will retrieve the value of a specified configuration variable, and the function ***ini_get_all()*** will retrieve an array filled with the entire selection of configuration variables that are available. The following example shows how to retrieve both environment and configuration variables.

The Code:

```
<?php
//Here is an example of retrieving an environmental variable or two.
echo $_ENV['ProgramFiles'] . "<br />"; //Outputs C:\Program Files.
echo $_ENV['COMPUTERNAME'] . "<br />"; //Outputs BABINZ-CODEZ.
echo getenv("COMPUTERNAME") . "<br />"; //Also Outputs BABINZ-CODEZ.
//Now, let's look at reading configuration variables.
echo ini_get ("post_max_size") . "<br />"; //Outputs 8MB.
//And you can output the entire listing with this function.
print_r (ini_get_all());
?>
```

Setting Environment and Configuration Variables:

Setting environment and configuration variables is just as easy as it is to get them. While working with environment variables, you merely need to assign a new value to the $_ENV Superglobal to process a temporary change. The change will be in effect for the script's duration.

The same applies for configuration variables but with a different approach. To set a configuration variable, you have to use the PHP function ***ini_set()***, which will allow you to set a configuration variable for the script's duration. Once the script finishes executing, the configuration variable will return to its original state. The prototype for *ini_set()* is as follows:

string ini_set (string varname, string newvalue)

The Code:

```
<?php
//Setting an environment variable in php is as easy as assigning it.
echo $_ENV['COMPUTERNAME'] . "<br />"; // Echoes BABINZ-CODEZ.
$_ENV['COMPUTERNAME'] = "Hello World!";
echo $_ENV['COMPUTERNAME'] . "<br />"; //Echoes the new COMPUTERNAME.
//Of course the change is relevant only for the current script.
//Setting a configuration variable is the same in that it is in effect only for
//the duration of the script.
echo ini_get ('post_max_size'); //Echoes 8MB.
//Then you set it to 200M for the duration of the script.
ini_set('post_max_size','200M');
//Any files that are to be uploaded in this script will be OK up to 200M.
?>
```

WORKING WITH DATE AND TIME:

Dates and times are quite unlike most other sorts of data you are likely to encounter when programming with PHP 5. Arrays, whether they are indexed arrays or associative arrays, are nicely structured.

Fortunately, PHP has a number of functions to help you keep dates and times simple for computers. Now we will show you how to use some of these functions to perform tasks such as the following:

- Obtaining the current date and time
- Accounting for time zones
- Converting between different date and time formats
- Performing calculations involving dates and times (the earlier "quick question" being one example)
- Determining whether a given date and time representation is a valid one

The PHP 5 date and time function library makes up a core part of the language. The library is included by default, and no special compilation or configuration directives are required to use these functions: they are available in any working PHP installation. No external dependencies such as shared libraries need to be installed on the server in addition to PHP. Also, this library does not define any special constants or resource types.

The table below lists the PHP 5 Date/Time Functions:

Function	Description
checkdate()	Validates set of Gregorian year, month, and day values (for example, 2005, 3, 17).
date_sunrise()	Returns time of sunrise for a given day and location (new in PHP 5).
date_sunset()	Returns time of sunset for a given day and location (new in PHP 5).
date()	Formats a local date/time, given a Unix timestamp and a formatting string.

getdate()	Given a Unix timestamp, returns an associative array containing date and time Information (defaults to current time).
gettimeofday()	Returns an associative array containing information about the current system time. In PHP 5.1, it is possible for this function to return a float as well.
gmdate()	Formats a GMT/UTC date/time. Uses the same formatting characters as the date() function.
gmmktime()	Converts a set of GMT date/time values into a Unix timestamp (analogous to mktime()).
gmstrftime()	Formats a GMT/UTC date/time according to locale settings (similar to strftime() except the time used is GMT/UTC).
idate()	Formats a local time/date value as an integer. Uses many of the same formatting characters as the date() function
localtime()	Given a Unix timestamp, returns an array of date/time values.
microtime()	Returns a string representation of the current Unix timestamp with microseconds.
mktime()	Converts a set of local date/time values into a Unix timestamp.
strftime()	Given a timestamp and a formatting string, returns a representation of a local date/time according to locale settings.
strptime()	Given a date/time string generated with strftime() and the formatting string used to generate it, returns a Unix timestamp.
strtotime()	Converts an English textual date/time description into a Unix timestamp.
time()	Returns the current system date and time as a Unix timestamp.

Most of these functions depend on the concept of a Unix timestamp. Simply, a Unix timestamp reflects the time elapsed since the beginning of what is known as the Unix epoch, that is, midnight on January 1, 1970, GMT. Usually this is expressed in seconds, although sometimes milliseconds or microseconds are used.

Displaying Dates and Times:

The three functions that you most need to be familiar with for displaying human-readable dates in English are ***time()***, ***mktime()***, and ***date()***. time() and mktime() provide ways to represent date/time values that are locale independent and easy for computers to use.

Each of these functions returns a time expressed as seconds since the Unix epoch. The difference between time() and mktime() is that time() does not take any arguments and always returns a value corresponding to the current system time, whereas mktime() retrieves a Unix timestamp for an arbitrary date and time. This latter function takes from zero to seven arguments; when you call it without any arguments, it acts just like time() and uses the current system date and time.

Displaying Human-Readable Dates and Times:

To obtain a timestamp for the current system date and time, it is necessary only to call the time() function, as shown here:

```
<?php
echo time();
?>
```

In a web browser, this produces output such as the following:

```
1110638611
```

This is not helpful for users, who are likely to be expecting something more along the lines of May 23, 2005, 12:25 p.m. For obtaining a human-readable date and time, PHP provides the date() function. When called with a single argument (a formatting string), this function returns a string representation of the current date and/or time. The optional second argument is a timestamp. The following example shows a few ways you can use various formatting strings with date().

The Code:

```php
<?php
$time = time();
$formats = array(
'U',
'r',
'c',
'l, F jS, Y, g:i A',
'H:i:s D d M y',
'm/j/y g:i:s O (T)'
);
foreach($formats as $format)
echo "<p><b>$format</b>: " . date($format, $time) . "</p>\n";
?>
```

Here is the output of this script as it might be viewed in a browser:

```
U: 1110643578

r: Sun, 13 Mar 2005 02:06:18 +1000

c: 2005-03-13T02:06:18+10:00

l, F jS, Y, g:i A: Sunday, March 13th, 2005, 2:06 AM

H:i:s D d M y: 02:06:18 Sun 13 Mar 05

m/j/y g:i:s a O (T): 03/13/05 2:06:18 am +1000 (E. Australia Standard Time)
```

Following is the list of formatting Characters for the date() Function:

Character	**Description**
Month	
F	Full name of the month
M	Three-letter abbreviation for the month.
m	Numeric representation for the month, with leading zero (two digits)
n	Numeric representation for the month (no leading zero)
Day	
d	Day of the month, with leading zeros (two digits).
j	Day of the month (no leading zeros).
S	Ordinal suffix for the day of the month, two characters (st, nd, th); most commonly used in combination with j.
l	Full name of the day of the week.
D	A textual representation of a day, three letters.
w	Numeric representation of the day of the week (0 = Sunday, 6 = Saturday).
Year	
y	Two-digit year.
Y	Four-digit year.
Hour	
h	Hour in 12-hour format, with leading zero (two digits).
g	Hour in 12-hour format (no leading zero).
H	Hour in 24-hour format, with leading zero (two digits).
G	Hour in 24-hour format (no leading zero).
a	am/pm (lowercase).

A	AM/PM (uppercase).
O (uppercase)	String representation of the difference in hours between local time and GMT/UTC (for example, +1000, –0500).
Minute	
i	Minute, with leading zero (two digits).
j	Minute (no leading zero).
Second	
s	Second, with leading zero (two digits).
Z	Integer representation of the difference in seconds between local time and GMT/UTC (for example, 36000 for GMT+1000 and –18000 for GMT–0500).
Complete Date and Time	
c	ISO-8601 format (YYYY-MM-DDTHH:MM:SS±HHMM, for example, 2005-03-14T19:38:08+10:00).
r	RFC-2822 format WWW, DD MMM YYYY HH:MM:SS ±HHMM, for example, Mon, 14 Mar 2005 19:38:08 +1000).
U	Seconds since the Unix epoch. Calling date('U') with no timestamp argument produces the same output as the time() function.

Displaying Arbitrary Dates and Times:

Displaying arbitrary dates and times requires passing the Unix timestamp for the desired date and time to the date() function as its second parameter. As already mentioned, you can obtain this with mktime(), which is pretty straightforward to use. In this example, you can get the Unix timestamp for 6:30 p.m. on August 10, 1997, and test the result by passing it to the date() function.

The Code:

```
<?php
$timestamp = mktime(18, 30, 0, 8, 10, 1997);
echo date('r (T)', $timestamp);
?>
```

The output from this on our server is as follows:

```
Sun, 10 Aug 1997 18:30:00 +1000 (E. Australia Standard Time)
```