**SEMESTER – VI - III BSc IT**

**18BIT61C – Programming in PHP**

**Prepared by Dr.N.Thenmozhi**

**UNIT IV:** Reading & Writing Files – Testing File Attributes – Managing Sessions And Using Session Variables – Destroying A Session. Storing Data in Cookies – Selecting Cookies – Removing Cookies Data – Deleting Cookies – Dealing With Date & Time.

**TEXT BOOKS**

1. Vikram Vaswani, "PHP: A beginners guide", TMH Hill, 1stedition, 2010 (Unit-I to IV).

**REFERENCE BOOKS**

1. Matt Doyle, " Beginning PHP 5.3", Wiley India pvt. Ltd, First edition, 2010.
2. Luke welling and Laura Thomson, "PHP and MySQL Web Development", 5th Edition, 2016.

## Reading & Writing Files

Since PHP is a server side programming language, it allows you to work with files and directories stored on the web server. The following will show how to create, access, and manipulate files on your web server using the PHP file system functions.

### Opening a File with PHP fopen() Function

To work with a file you first need to open the file. The PHP `fopen()` function is used to open a file. The basic syntax of this function can be given with:

```
fopen(filename, mode)
```

The first parameter passed to `fopen()` specifies the name of the file you want to open, and the second parameter specifies in which mode the file should be opened. For example:

```
<?php
    $handle = fopen("data.txt", "r");
?>
```

The file may be opened in one of the following modes:

| Modes | What it does |
|---|---|
| r | Open the file for reading only. |
| r+ | Open the file for reading and writing. |
| w | Open the file for writing only and clears the contents of file. If the file does not exist, PHP will attempt to create it. |
| w+ | Open the file for reading and writing and clears the contents of file. If the file does |

| Modes | What it does |
|---|---|
| | not exist, PHP will attempt to create it. |
| a | Append. Opens the file for writing only. Preserves file content by writing to the end of the file. If the file does not exist, PHP will attempt to create it. |
| a+ | Read/Append. Opens the file for reading and writing. Preserves file content by writing to the end of the file. If the file does not exist, PHP will attempt to create it. |
| x | Open the file for writing only. Return `FALSE` and generates an error if the file already exists. If the file does not exist, PHP will attempt to create it. |
| x+ | Open the file for reading and writing; otherwise it has the same behavior as 'x'. |

If you try to open a file that doesn't exist, PHP will generate a warning message. So, to avoid these error messages you should always implement a simple check whether a file or directory exists or not before trying to access it, with the PHP `file_exists()` function. For example

```
<?php
        $file = "data.txt";
        // Check the existence of file
        if(file_exists($file))
        {
                 // Attempt to open the file
                $handle = fopen($file, "r");
        }
        else
        {
                echo "ERROR: File does not exist.";
        }
?>
```

### Closing a File with PHP `fclose()` Function

Once you've finished working with a file, it needs to be closed. The `fclose()` function is used to close the file, as shown in the following example:

```
<?php
        $file = "data.txt";
         // Check the existence of file
        if(file_exists($file))
        {
                 // Open the file for reading
```

```
        $handle = fopen($file, "r") or die("ERROR: Cannot open the file.");
         /* Some code to be executed */
          // Closing the file handle
         fclose($handle);
        }
        else
        {
                echo "ERROR: File does not exist.";
        }
?>
```

### Reading from Files with PHP fread() Function

PHP has several functions for reading data from a file. You can read from just one character to the entire file with a single operation.

### Reading Fixed Number of Characters

The `fread()` function can be used to read a specified number of characters from a file. The basic syntax of this function can be given with.

```
fread(file handle, length in bytes)
```

This function takes two parameter — A file handle and the number of bytes to read. The following example reads 20 bytes from the "data.txt" file including spaces. Let's suppose the file "data.txt" contains a paragraph of text "The quick brown fox jumps over the lazy dog."

```
<?php
        $file = "data.txt";

        // Check the existence of file
        if(file_exists($file))
        {
                // Open the file for reading
          $handle = fopen($file, "r") or die("ERROR: Cannot open the file.");

        // Read fixed number of bytes from the file
         $content = fread($handle, "20");

        // Closing the file handle
         fclose($handle);

        // Display the file content
        echo $content;
        }
        else
        {
                  echo "ERROR: File does not exist.";
        }
?>
```

The above example will produce the following output:

The quick brown fox

**Reading the Entire Contents of a File**

The `fread()` function can be used in conjugation with the `filesize()` function to read the entire file at once. The `filesize()` function returns the size of the file in bytes.

```
<?php
	$file = "data.txt";

	// Check the existence of file
	if(file_exists($file))
	{
		// Open the file for reading
		$handle = fopen($file, "r") or die("ERROR: Cannot open the file.");

		 // Reading the entire file
		$content = fread($handle, filesize($file));

		 // Closing the file handle
		fclose($handle);

		 // Display the file content
		echo $content;
	}
	else
	{
		  echo "ERROR: File does not exist.";
	}
?>
```

The above example will produce the following output:

The quick brown fox jumps over the lazy dog.

The easiest way to read the entire contents of a file in PHP is with the `readfile()` function. This function allows you to read the contents of a file without needing to open it. The following example will generate the same output as above example:

```
<?php
	$file = "data.txt";

	// Check the existence of file
	if(file_exists($file))
	{
	// Reads and outputs the entire file
		readfile($file) or die("ERROR: Cannot open the file.");
	}
	else
	{
		 echo "ERROR: File does not exist.";
```

```
        }
?>
```
The above example will produce the following output:

```
The quick brown fox jumps over the lazy dog.
```

Another way to read the whole contents of a file without needing to open it is with the `file_get_contents()` function. This function accepts the name and path to a file, and reads the entire file into a string variable. Here's an example:

```
<?php
        $file = "data.txt";

        // Check the existence of file
        if(file_exists($file))
        {
                // Reading the entire file into a string
                $content = file_get_contents($file) or die("ERROR: Cannot open the file.");

                 // Display the file content
                 echo $content;
        }
        else
        {
                 echo "ERROR: File does not exist.";
        }
?>
```

One more method of reading the whole data from a file is the PHP's `file()` function. It does a similar job to `file_get_contents()` function, but it returns the file contents as an array of lines, rather than a single string. Each element of the returned array corresponds to a line in the file.

To process the file data, you need to iterate over the array using a foreach loop. Here's an example, which reads a file into an array and then displays it using the loop:

```
<?php
        $file = "data.txt";

        // Check the existence of file
        if(file_exists($file))
        {
                // Reading the entire file into an array
                 $arr = file($file) or die("ERROR: Cannot open the file.");
                 foreach($arr as $line)
                 {
                         echo $line;
                 }
         }
        else
        {
                echo "ERROR: File does not exist.";
```

```
    }
?>
```

### Writing the Files Using PHP fwrite() Function

Similarly, you can write data to a file or append to an existing file using the PHP `fwrite()` function. The basic syntax of this function can be given with:

```
    fwrite(file handle, string)
```

The `fwrite()` function takes two parameter — A file handle and the string of data that is to be written, as demonstrated in the following example:

```
<?php
    $file = "note.txt";
    // String of data to be written
    $data = "The quick brown fox jumps over the lazy dog.";

    // Open the file for writing
    $handle = fopen($file, "w") or die("ERROR: Cannot open the file.");

    // Write data to the file
    fwrite($handle, $data) or die ("ERROR: Cannot write the file.");

    // Closing the file handle
    fclose($handle);

    echo "Data written to the file successfully.";
?>
```

In the above example, if the "note.txt" file doesn't exist PHP will automatically create it and write the data. But, if the "note.txt" file already exist, PHP will erase the contents of this file, if it has any, before writing the new data, however if you just want to append the file and preserve existing contents just use the mode `a` instead of `w`.

An alternative way is using the `file_put_contents()` function. It is counterpart of `file_get_contents()` function and provides an easy method of writing the data to a file without needing to open it. This function accepts the name and path to a file together with the data to be written to the file. Here's an example:

```
<?php
    $file = "note.txt";

    // String of data to be written
    $data = "The quick brown fox jumps over the lazy dog.";

    // Write data to the file
    file_put_contents($file, $data) or die("ERROR: Cannot write the file.");

    echo "Data written to the file successfully.";
?>
```

If the file specified in the file_put_contents() function already exists, PHP will overwrite it by default. If you would like to preserve the file's contents you can pass the special FILE_APPEND flag as a third parameter to the file_put_contents() function. It will simply append the new data to the file instead of overwriting it. Here's an example:

```
<?php
	$file = "note.txt";

	// String of data to be written
	$data = "The quick brown fox jumps over the lazy dog.";

	// Write data to the file
	file_put_contents($file, $data, FILE_APPEND) or die("ERROR: Cannot write the file.");

	echo "Data written to the file successfully.";
?>
```

### Renaming Files with PHP rename() Function

You can rename a file or directory using the PHP's rename() function, like this:

```
<?php
	$file = "file.txt";

	// Check the existence of file
	if(file_exists($file)
	{
		// Attempt to rename the file
		if(rename($file, "newfile.txt"))
		{
			echo "File renamed successfully.";
		}
		 else
		{
			 echo "ERROR: File cannot be renamed.";
		 }
	}
	else
	{
		 echo "ERROR: File does not exist.";
	}
?>
```

### Removing Files with PHP unlink() Function

You can delete files or directories using the PHP's unlink() function, like this:

```
<?php
	$file = "note.txt";
```

```
        // Check the existence of file
        if(file_exists($file))
        {
                // Attempt to delete the file
                if(unlink($file)){
    echo "File removed successfully.";
  } else{
    echo "ERROR: File cannot be removed.";
  }
} else{
  echo "ERROR: File does not exist.";
}
?>
```

**PHP File system Functions**

The following table provides the overview of some other useful PHP file system functions that can be used for reading and writing the files dynamically.

| Function | Description |
|---|---|
| fgetc() | Reads a single character at a time. |
| fgets() | Reads a single line at a time. |
| fgetcsv() | Reads a line of comma-separated values. |
| filetype() | Returns the type of the file. |
| feof() | Checks whether the end of the file has been reached. |
| is_file() | Checks whether the file is a regular file. |
| is_dir() | Checks whether the file is a directory. |
| is_executable() | Checks whether the file is executable. |
| realpath() | Returns canonicalized absolute pathname. |
| rmdir() | Removes an empty directory. |

**Managing Sessions And Using Session Variables**

Session functions provide a unique identifier to a user, which can then be used to store and acquire information linked to that ID. When a visitor accesses a session enabled page, either a new identifier is allocated or the user is re-associated with one that was already

established in a previous visit. Any variables that have been associated with the session become available to your code through the $_SESSION superglobal.

Session state is usually stored in a temporary file, although you can implement database storage or other server-side storage methods using a function called session_set_save_handler().

**Starting a Session**

To work with a session, you need to explicitly start or resume that session unless you have changed your php.ini configuration file. By default, sessions do not start automatically. If you want to start a session this way, you must find the following line in your php.ini file and change the value from 0 to 1 (and restart the web server):

session.auto_start = 0

By changing the value of session.auto_start to 1, you ensure that a session initiates for every PHP document. If you don't change this setting, you need to call the session_start() function in each script.

After a session is started, you instantly have access to the user's session ID via the session_id() function. The session_id() function enables you to either set or retrieve a session ID. Listing below code starts a session and prints the session ID to the browser.

```
1: <?php
2: session_start();
3: echo "<p>Your session ID is ".session_id()."."</p>";
4: ?>
```

When this script (let's call it session_checkid.php) is run for the first time from a browser, a session ID is generated by the session_start() function call on line 2. If the script is later reloaded or revisited, the same session ID is allocated to the user.

This action assumes that the user has cookies enabled. For example, when run this script the first time, the output is as follows:

Your session ID is 8jou17in51d08e5onsjkbles16.

When reload the page, the output is still

Your session ID is 8jou17in51d08e5onsjkbles16.

Because, it has cookies enabled and the session ID still exists.

Because start_session() attempts to set a cookie when initiating a session for the first time, it is imperative that you call this function before you output anything else at all to the browser. If you do not follow this rule, your session will not be set, and you will likely see warnings on your page.

Sessions remain current as long as the web browser is active. When the user restarts the browser, the cookie is no longer stored. You can change this behaviour by altering the session.cookie_lifetime setting in your php.ini file. The default value is 0, but you can set an expiry period in seconds.

**Working with Session Variables**

Accessing a unique session identifier in each of your PHP documents is only the start of session functionality. When a session is started, you can store any number of variables in the $_SESSION superglobal and then access them on any session-enabled page. The following code will adds two variables into the $_SESSION superglobal: product1 and product2 (lines 3 and 4).

Example : Storing Variables in a Session
```
1: <?php
2: session_start();
3: $_SESSION[‘product1’] = “Sonic Screwdriver”;
4: $_SESSION[‘product2’] = “HAL 2000”;
5: echo “The products have been registered.”;
6: ?>
```

The above code will not become apparent until the user moves to a new page. The following example creates a separate PHP script that accesses the variables stored in the $_SESSION superglobal.

Example : Accessing Stored Session Variables
```
1: <?php
2: session_start();
3: ?>
4: <p>Your chosen products are:</p>
5: <ul>
6: <li><?php echo $_SESSION[‘product1’]; ?></li>
7: <li><?php echo $_SESSION[‘product2’]; ?></li>
8: </ul>
```

The above script does show how to access stored session variables. Behind the scenes, PHP writes information to a temporary file. You can find out where this file is being written on your system by using the session_save_path() function. This function optionally accepts a path to a directory and then writes all session files to it. If you pass it no arguments, it returns a string representing the current directory to which it saves session files. The following prints /tmp:

```
echo session_save_path();
```

A glance at my /tmp directory reveals a number of files with names like the following:

```
sess_fa963e3e49186764b0218e82d050de7b
sess_76cae8ac1231b11afa2c69935c11dd95
sess_bb50771a769c605ab77424d59c784ea0
```

Opening the file that matches the session ID, it was allocated and how the registered variables have been stored:

```
product1|s:17:”Sonic Screwdriver”;product2|s:8:”HAL 2000”;
```

When a value is placed in the $_SESSION superglobal, PHP writes the variable name and value to a file. After you add a variable to the $_SESSION superglobal, you can still change its value at any time during the execution of your script, but the altered value is not reflected in the global setting until you reassign the variable to the $_SESSION superglobal.

**Destroying sessions and Un setting variables**

You can use session_destroy() to end a session, erasing all session variables. The session_destroy() function requires no arguments. You should have an established session for this function to work as expected. The following code fragment resumes a session and abruptly destroys it:

```
session_start();
session_destroy();
```

When you move on to other pages that work with a session, the session you have destroyed will not be available to them, forcing them to initiate new sessions of their own. Any registered variables will be lost.

The session_destroy() function does not instantly destroy registered variables, however. They remain accessible to the script in which session_destroy() is called (until it is reloaded). The following code fragment resumes or initiates a session and registers a variable called test, set to 5. Destroying the session does not destroy the registered variable:

```
session_start();
$_SESSION['test'] = 5;
session_destroy();
echo $_SESSION['test']; // prints 5
```

To remove all registered variables from a session, you simply unset the variable:

```
session_start();
$_SESSION['test'] = 5;
session_destroy();
unset($_SESSION['test']);
echo $_SESSION['test']; // prints nothing (or a notice about an undefined index)
```

**Introducing cookies**

You can use cookies within your PHP scripts to store small bits of information about a user. **A cookie is** a small amount of data stored by the user's browser in compliance with a request from a server or script. A single host can request that up to 20 cookies be stored by a user's browser. Each cookie consists of a name, value, and expiration date, as well as host and path information. The size of an individual cookie is limited to 4KB.

After a cookie is set, only the originating host can read the data, ensuring that the user's privacy is respected. Furthermore, users can configure their browser to notify them upon receipt of all cookies, or even to refuse all cookie requests.

**The Anatomy of a Cookie**

A PHP script that sets a cookie might send headers that look something like this:

```
HTTP/1.1 200 OK
Date: Wed, 18 Jan 2012 10:50:58 GMT
Server: Apache/2.2.21 (Unix) PHP/5.4.0
X-Powered-By: PHP/5.4.0
Set-Cookie: vegetable=artichoke; path=/; domain=yourdomain.com
Connection: close
Content-Type: text/html
```

This Set-Cookie header contains a name/value pair, a path, and a domain. If set, the expiration field provides the date at which the browser should "forget" the value of the cookie. If no expiration date is set, the cookie expires when the user's session expires—that is, when he closes his browser.

The path and domain fields work together: The path is a directory found on the domain, below which the cookie should be sent back to the server. If the path is "/", which is common, that means the cookie can be read by any files below the document root. If the path is "/products/", the cookie can be read only by files within the /products directory of the website.

The domain field represents the Internet domain from which cookie-based communication is allowed. For example, if your domain is www.yourdomain.com and you use www.yourdomain.com as the domain value for the cookie, the cookie will be valid only when browsing the www.domain.com website. This could pose a problem if you send the user to some domain like www2.domain.com or billing.domain.com within the course of his browsing experience, because the original cookie will no longer work. Therefore, it is common simply to begin the value of the domain slot in cookie definitions with a dot, leaving off the host (for example, .domain.com). In this manner, the cookie is valid for all hosts on the domain. The domain cannot be different from the domain from which the cookie was sent; otherwise, the cookie will not function properly, if at all, or the web browser will refuse the cookie in its entirety.

**Cookie Attributes**

1. Every cookie contains a name-value pair, which represents the variable name and corresponding value to be stored in the cookie.

| Attribute | Description | Example |
|---|---|---|
| name=*value* | The cookie name and value | 'email=myself@some.domain.com' |
| expires | The validity of the cookie | 'expires=Friday, 25-Jan-08 |
| domain | The domain associated | 'domain=.thiswebsite.com' with the cookie 23:59:50 IST' |
| path | The domain path associated with the cookie | path=/' |
| secure | If present, a secure HTTP connection is required to read | 'secure' |

Table 1 : Cookie Attributes

2. A cookie's 'expires' attribute defines how long the cookie is valid for. Setting this

attribute's value to a date in the past will usually cause the browser to delete the cookie.
3. A cookie's 'domain' attribute defines the domain name to be associated with the cookie. Only this domain will be able to access the information in the cookie.
4. A cookie's 'path' attribute defines which sections of the domain specified in the 'domain' attribute can access the cookie. Setting this to the server root (/) allows the entire domain access to the information stored in the cookie.
5. A cookie's 'secure' attribute indicates whether a secure HTTP connection is mandatory before the cookie can be accessed.

**Cookie Headers**

Cookies are transmitted between the user's browser and a remote Web site by means of HTTP headers. For example, to set a cookie, a Web site must send the user's browser a 'Set-Cookie:' header containing the necessary attributes. The following example illustrates the headers sent to create two cookies for a domain:

```
Set-Cookie: username=john; path=/; domain=.thiswebsite.com;
expires=Friday, 25-Jan-08 23:59:50 IST
Set-Cookie: location=UK; path=/; domain=.thiswebsite.com;
expires=Friday, 25-Jan-08 23:59:50 IST
```

Similarly, if a particular cookie is valid for a Web site and path, the user's browser automatically includes the cookie information in a 'Cookie:' header when requesting the site URL. Using the preceding example, when the user next visits the thiswebsite.com domain, the browser will automatically include the following header in its request:

```
Cookie: username=john; location=UK
```

**Setting Cookies**

PHP's cookie-manipulation API is very simple: it consists of a single function, setcookie(), which can be used to both set and remove cookies. This function accepts six arguments: the cookie's name and value, its expiry date in UNIX timestamp format, its domain and path, and a Boolean flag for the 'secure' attribute. The function returns true if the cookie header was successfully transmitted to the user's system; however, this does not indicate if the cookie was successfully set or not (if the user's browser is set to reject all cookies, a cookie header might be successfully transmitted but the cookie might not actually be set on the user's system).

Here's an example, which sets a cookie containing the user's e-mail address:

```
<?php
    // set a cookie
    setcookie('email', 'john@somewebsite.com', mktime()+129600, '/');
?>
```

It's possible to set multiple cookies, simply by calling setcookie() once for each cookie. Here's an example, which sets three cookies with different validity periods and paths:

```
<?php
    // set multiple cookies
    setcookie('username', 'whitewhale', mktime()+129600, '/');
    setcookie('email', 'john@somewebsite.com', mktime()+86400, '/');
```

```
        setcookie('role', 'moderator', mktime()+3600, '/admin');
?>
```

**Reading Cookies**

Cookies set for a domain become available in the special $_COOKIE associative array in PHP scripts running on that domain. These cookies may be accessed using standard array notation. Here's an example:

```
<?php
        // read cookie
        if (isset($_COOKIE['email']))
        {
                echo 'Welcome back, ' . $_COOKIE['email'] . '!';
        }
        else
        {
                echo 'Hello, new user!';
        }
?>
```

**Removing Cookies**

To delete a cookie, use setcookie() with its name to set the cookie's expiry date to a value in the past, as shown here:

```
<?php
        // remove cookie
        $ret = setcookie('role', 'moderator', mktime()-1600, '/admin');
        if ($ret)
        {
                echo 'Cookie headers successfully transmitted.';
        }
?>
```

**Accessing Cookies**

If your web browser is configured to store cookies, it keeps the cookie-based information until the expiration date. If the user points the browser at any page that matches the path and domain of the cookie, it resends the cookie to the server. The browser's headers might look something like this:

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.7 (KHTML, like
Gecko) Chrome/16.0.912.75 Safari/535.7

Host: www.yourdomain.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en,pdf
Accept-Charset: iso-8859-1,*,utf-8
```

Cookie: vegetable=artichoke

A PHP script then has access to the cookie in the environment variable HTTP_COOKIE or as part of the $_COOKIE superglobal variable, which you may access three different ways:

```
echo $_SERVER['HTTP_COOKIE']; // will print "vegetable=artichoke"
echo getenv('HTTP_COOKIE'); // will print "vegetable=artichoke"
echo $_COOKIE['vegetable']; // will print "artichoke"
```

**Setting a Cookie with PHP**

You can set a cookie in a PHP script in two ways. First, you can use the header() function to set the Set-Cookie header. The header() function requires a string that is then included in the header section of the server response. Because headers are sent automatically for you, header() must be called before any output at all is sent to the browser:

```
header("Set-Cookie: vegetable=artichoke; expires=Thu, 19-Jan-12 14:39:58 GMT;
path=/; domain=yourdomain.com");
```

This method of setting a cookie requires you to build a function to construct the header string. Although formatting the date as in this example and URL-encoding the name/value pair is not a particularly arduous task, it is a repetitive one because PHP provides a function that does just that: setcookie().

The setcookie() function does what its name suggests—it outputs a Set-Cookie header. For this reason, it should be called before any other content is sent to the browser. The function accepts the cookie name, cookie value, expiration date in UNIX epoch format, path, domain, and integer that should be set to 1 if the cookie is to be sent only over a secure connection. All arguments to this function are optional apart from the first (cookie name) parameter.

**Example : Setting and Printing a Cookie Value**

```
1: <?php
2: setcookie("vegetable", "artichoke", time()+3600, "/", ".yourdomain.com", 0);
3:
4: if (isset($_COOKIE['vegetable'])) {
5: echo "<p>Hello again! You have chosen: ".$_COOKIE['vegetable']."."</p>";
6: } else {
7: echo "<p>Hello, you. This may be your first visit.</p>";
8: }
9: ?>
```

Even though the listing sets the cookie (line 2) when the script is run for the first time, the $_COOKIE['vegetable'] variable is not created at this point. Because a cookie is read only when the browser sends it to the server, you cannot read it until the user revisits a page within this domain.

The cookie name is set to "vegetable" on line 2, and the cookie value to "artichoke". The time() function gets the current timestamp and adds 3600 to it (3,600 seconds in an hour). This total represents the expiration date. The code defines a path of "/", which means that a cookie should be sent for any page within this server environment. The domain

argument is set to ".yourdomain.com" , which means that a cookie will be sent to any server in that group. Finally, the code passes 0 to setcookie(), signaling that cookies can be sent in an unsecure environment.

Passing setcookie() an empty string ("") for string arguments or 0 for integer fields causes these arguments to be skipped. With using a dynamically created expiration time in a cookie, the expiration time is created by adding a certain number of seconds to the current system time of the machine running Apache and PHP. If this system clock is not accurate, the machine may send the cookie at an expiration time that has already passed.

**Deleting a Cookie with PHP**

Officially, to delete a cookie, you call setcookie() with the name argument only:

```
setcookie("vegetable");
```

This approach does not always work well, however, and you should not rely on it. Instead, to delete a cookie, it is safest to set the cookie with a date that you are sure has already expired:

```
setcookie("vegetable", "", time()-60, "/", ".yourdomain.com", 0);
```

Also make sure that you pass setcookie() the same path, domain, and secure parameters as you did when originally setting the cookie.