setcookie

(PHP 4, PHP 5)

setcookie — Send a cookie

Report a bug

Description

bool **setcookie** (string *$name* [, string *$value* [, int *$expire* = *0* [, string *$path* [, string *$domain* [, bool *$secure* = *false* [, bool *$httponly* = *false*]]]]])

setcookie() defines a cookie to be sent along with the rest of the HTTP headers. Like other headers, cookies must be sent ***before*** any output from your script (this is a protocol restriction). This requires that you place calls to this function prior to any output, including ***<html>*** and ***<head>*** tags as well as any whitespace.

Once the cookies have been set, they can be accessed on the next page load with the *$_COOKIE* or *$HTTP_COOKIE_VARS* arrays. Note, superglobals such as *$_COOKIE* became available in PHP 4.1.0. Cookie values also exist in *$_REQUEST*.

Report a bug

Parameters

All the arguments except the ***name*** argument are optional. You may also replace an argument with an empty string (***""***) in order to skip that argument. Because the ***expire*** argument is integer, it cannot be skipped with an empty string, use a zero (***0***) instead.

» RFC 6265 provides the normative reference on how each **setcookie()** parameter is interpreted.

name

The name of the cookie.

value

The value of the cookie. This value is stored on the clients computer; do not store sensitive information. Assuming the ***name*** is ***'cookiename'***, this value is retrieved through *$_COOKIE['cookiename']*

expire

The time the cookie expires. This is a Unix timestamp so is in number of seconds since the epoch. In other words, you'll most likely set this with the time() function plus the number of seconds before you want it to expire. Or you might use mktime(). ***time()+60*60*24*30*** will set the cookie to expire in 30 days. If set to 0, or omitted, the cookie will expire at the end of the session (when the

browser closes).

Note:

You may notice the ***expire*** parameter takes on a Unix timestamp, as opposed to the date format ***Wdy, DD-Mon-YYYY HH:MM:SS GMT***, this is because PHP does this conversion internally.

path

The path on the server in which the cookie will be available on. If set to ***'/'***, the cookie will be available within the entire ***domain***. If set to ***'/foo/'***, the cookie will only be available within the ***/foo/*** directory and all sub-directories such as ***/foo/bar/*** of ***domain***. The default value is the current directory that the cookie is being set in.

domain

The domain that the cookie is available to. Setting the domain to ***'www.example.com'*** will make the cookie available in the ***www*** subdomain and higher subdomains. Cookies available to a lower domain, such as ***'example.com'*** will be available to higher subdomains, such as ***'www.example.com'***. Older browsers still implementing the deprecated » RFC 2109 may require a leading ***.*** to match all subdomains.

secure

Indicates that the cookie should only be transmitted over a secure HTTPS connection from the client. When set to **TRUE**, the cookie will only be set if a secure connection exists. On the server-side, it's on the programmer to send this kind of cookie only on secure connection (e.g. with respect to *$_SERVER["HTTPS"]*).

httponly

When **TRUE** the cookie will be made accessible only through the HTTP protocol. This means that the cookie won't be accessible by scripting languages, such as JavaScript. It has been suggested that this setting can effectively help to reduce identity theft through XSS attacks (although it is not supported by all browsers), but that claim is often disputed. Added in PHP 5.2.0. **TRUE** or **FALSE**

Return Values

If output exists prior to calling this function, **setcookie()** will fail and return **FALSE**. If **setcookie()** successfully runs, it will return **TRUE**. This does not indicate whether the user accepted the cookie.

Report a bug

Examples

Some examples follow how to send cookies:

Example #1 setcookie() send example

```
<?php
$value = 'something from somewhere';

setcookie("TestCookie", $value);
setcookie("TestCookie", $value, time()+3600);  /* expire in 1 hour */
setcookie("TestCookie", $value, time()+3600, "/
~rasmus/", "example.com", 1);
?>
```

Note that the value portion of the cookie will automatically be urlencoded when you send the cookie, and when it is received, it is automatically decoded and assigned to a variable by the same name as the cookie name. If you don't want this, you can use setrawcookie() instead if you are using PHP 5. To see the contents of our test cookie in a script, simply use one of the following examples:

```
<?php
// Print an individual cookie
echo $_COOKIE["TestCookie"];
echo $HTTP_COOKIE_VARS["TestCookie"];

// Another way to debug/test is to view all cookies
print_r($_COOKIE);
?>
```

Example #2 setcookie() delete example

When deleting a cookie you should assure that the expiration date is in the past, to trigger the removal mechanism in your browser. Examples follow how to delete cookies sent in previous example:

```
<?php
// set the expiration date to one hour ago
setcookie ("TestCookie", "", time() - 3600);
setcookie ("TestCookie", "", time() - 3600, "/
~rasmus/", "example.com", 1);
?>
```

Example #3 setcookie() and arrays

You may also set array cookies by using array notation in the cookie name. This has the effect of setting as many cookies as you have array elements, but when the cookie is received by your script, the values are all placed in an array with the cookie's name:

```php
<?php
// set the cookies
setcookie("cookie[three]", "cookiethree");
setcookie("cookie[two]", "cookietwo");
setcookie("cookie[one]", "cookieone");

// after the page reloads, print them out
if (isset($_COOKIE['cookie'])) {
    foreach ($_COOKIE['cookie'] as $name => $value) {
        $name = htmlspecialchars($name);
        $value = htmlspecialchars($value);
        echo "$name : $value <br />\n";
    }
}
?>
```

The above example will output:

```
three : cookiethree
two : cookietwo
one : cookieone
```

Report a bug

Changelog

Version	Description
5.2.0	The ***httponly*** parameter was added.

Report a bug

Notes

Note:

You can use output buffering to send output prior to the call of this function, with the overhead of all of your output to the browser being buffered in the server until you send it. You can do this by calling ob_start() and ob_end_flush () in your script, or setting the ***output_buffering*** configuration directive on in

your *php.ini* or server configuration files.

Note:

If the PHP directive register_globals is set to ***on*** then cookie values will also be made into variables. In our examples below, *$TestCookie* will exist. It's recommended to use *$_COOKIE*.

Common Pitfalls:

- Cookies will not become visible until the next loading of a page that the cookie should be visible for. To test if a cookie was successfully set, check for the cookie on a next loading page before the cookie expires. Expire time is set via the ***expire*** parameter. A nice way to debug the existence of cookies is by simply calling ***print_r($_COOKIE);***.
- Cookies must be deleted with the same parameters as they were set with. If the value argument is an empty string, or **FALSE**, and all other arguments match a previous call to setcookie, then the cookie with the specified name will be deleted from the remote client. This is internally achieved by setting value to 'deleted' and expiration time to one year in past.
- Because setting a cookie with a value of **FALSE** will try to delete the cookie, you should not use boolean values. Instead, use ***0*** for **FALSE** and ***1*** for **TRUE**.
- Cookies names can be set as array names and will be available to your PHP scripts as arrays but separate cookies are stored on the user's system. Consider explode() to set one cookie with multiple names and values. It is not recommended to use serialize() for this purpose, because it can result in security holes.

Multiple calls to **setcookie()** are performed in the order called.

See Also

- header() - Send a raw HTTP header
- setrawcookie() - Send a cookie without urlencoding the cookie value
- cookies section
- » RFC 6265
- » RFC 2109