Using cookies with PHP

COOKIE CUTTER

Cookies are very much maligned and misunderstood but they can be useful tools in the hands of Web developers. David Tansley explains exactly what cookies are and how you can make use of them with PHP

A few months ago the ITV program Pop Idol captivated the British nation and the public was invited to vote on who they thought was the best singer. Votes could be registered by either ringing a designated phone number or via the ITV Web site. On the day of the final, when my two kids tried to vote more than once via the Web, shouts were heard exclaiming that: "It won't let me vote more than once". "Ah," I said, "their Web site is either overloaded or you have been cookied."

So what are cookies?

Cookies are very small text files that are sent from a Web server to your browser. The browser will then store them, usually as a cookie file. Cookies do not harm your computer; they are a way of storing general information about you or keeping track on what you are doing on a Web site. Cookies will only store information that you give the Web site, so be careful – if you don't want personal information to be stored, then don't give it in the first place!

Let's look at how a cookie might be used. Suppose you visited an online record store. If you decided to purchase a couple of records, a cookie would be used to keep track of your choice and how much you are spending. This cookie will provide you with a unique customer number that your browser will use to identify you to the Web site. When you make a purchase, this cookie is read and your purchase is added to a database, with your (cookie) number as the key that identifies you. When you wish to check out your transactions a database will be displayed. The Web server knows that these are your transactions because it will have used the cookie to identify and keep track of you.

Now you've got to remember, that the World Wide Web is a stateless transaction. By that we mean once you've loaded a Web page, that's it, the connection is broken. The Web server has no idea who you are, which is why cookies are so important – they keep track of your transactions/movements within that Web site. Cookies are also used for login screens and personalising Web pages like at Yahoo. The downside of cookies is that some Web servers now use them to load up lots of unwanted advertising banners based on a previous transaction you may have made. Thankfully, you can disable the use of cookies or let the browser tell you when a cookie is being sent via your browser configuration.

Cookie ingredients

Any Web-enabled script or programming language can utilise cookies. In this article we will demonstrate cookie handling using the Web scripting language PHP. PHP is a server-side language, which means it resides on the Web server itself. PHP has for a while supported sessions – a much better and more robust alternative to using cookies – but for this article we will stick with plain cookie handling. The principals that we are going to show you can be used in any Web language of your choice.

The structure of a cookie is as follows:

- *Value*: The actual (data) contents of the cookie.
- *Expiration*: The length of time a cookie is valid for.
- *Path*: Which directory the cookie is valid for. A single slash means all public Web directories on the Web server.
- *Domain*: The domain name the cookie is valid for. Please note you cannot make up a domain on your cookie, as security will prevent it from being sent successfully. If you do not know your domain then play it safe, leave it blank and it will default to your domain. If you specify a domain name – and you need to if you're on the Web – you can use a sort of wildcard (a dot at the beginning of the domain name) to match all other domains that belong to you. For example, suppose you belong to *www.example.com*, by specifying *.example.com* that also would be valid for

www1.example.com and *www2.example.com*, and so on.

- *Security*: If this is set to '1' then a secure connection (SSL) can read the cookie. Leaving the Security part blank will default to non-secure.

When setting a cookie not all the above are mandatory: if the Domain and Security are left blank, PHP will assume it is not a secure cookie and the domain will be the current one you belong to (if any). The expiration time is determined by the number of seconds since 01/01/1970. Don't worry, there's no need to get a calculator out. By using PHP's time function, all you need to do is give it the time in seconds when you want the cookie to expire. So, *time()+3600* will compute one hour from the current time (the cookie is sent), and *time()+86400* will compute 24 hours from the current time, get it? If you only want the cookie valid till the browser closes down then leave the expiration part blank.

When a cookie is initially sent from the Web server to the browser, you cannot then read that cookie until the client revisits. Beware of this; it is the most common mistake when learning cookies. Another common mistake is trying to send content to the browser before setting a cookie – this is a big no, no, as it won't get sent in a million years. Always send your cookie before outputting any content to your browser. (By content we mean any information/pictures that are displayed on the browser.)

You may have already guessed how a Web server could stop you from registering multiple Pop Idol votes at a time: by simply setting a cookie with say a expiration time of six hours. The cookie would be set when you initially vote, then when you try and vote again, the Web server would check to see if a cookie is present. If it is, then they must have already voted, viola!

Making a cookie

Now we know what the cookie's ingredients are let's get our hands dirty and bake one. To set a cookie with PHP the *setcookie* function is used. The format for this is:

```
setcookie(cookie_name, value, expire time,
path, domain, secure flag);
```

To set the expiration time to 12 hours, this would be 43200, worked out as follows:

```
(3600 = 1 hour, thus 3600 * 12hours=43200)
```

The code in Listing 1 is a simple script that sends a cookie to a browser with the contents of "Yum Yum, I love cookies", the cookie name is "cookie_test", and the expiration time is 12 hours.

All browsers have the options to either accept cookies automatically or prompt you before accepting. When testing cookie handling it is always best to set the browser to prompt you before accepting a cookie, as in Figure 1. This way you are absolutely sure that you are getting the cookie whilst testing cookie handling.

When the browser is pointed to the script in Listing 1 and the cookie is loaded, you can see the actual contents and structure of the cookie by selecting cookie details, as in Figure 2. Notice that the (value) contents part has been URL-encoded. When we next read back the cookie PHP will take care of the URL de-coding for us. The cookie will be stored in your HOME directory structure inside cookie.txt.

Reading the cookie

Now the cookie has been sent to the browser, the next time the browser revisits we can read it. How do we know which cookie to read, after all the browser will probably have quite a few cookies stored? Well for one, you can only read cookies that belong to your domain. Secondly you may have noticed the name we gave the cookie was "cookie_test". This is how we will pick the cookie up.

Before we try to read the cookie, it is best to first make sure the cookie is present. With PHP this is accomplished with the *isset* function, which tests to see if the object is defined. If the cookie is defined then we then display it to the browser, if the cookie is not defined then we can throw up a nice error message instead. Listing 2 does just that. Notice the use of braces on both sides of the else part. Figure 3 shows the output of the script to the browser after successfully reading the previous cookie that was sent in Listing 1. As a side note you can also read cookies by looking at the CGI environment variable HTTP_COOKIE or the PHP environment $HTTP_COOKIE_VAR.

Listing 1: Simple code to send a cookie with PHP

```
<?php
setcookie("cookie_test","Yum Yum, I love cookies",time()+43200,"/");
?>
```

Listing 2. Displaying the cookie if it is present

```
<?php
# showing a cookie
if (isset($cookie_test))
 {
  echo "Yum Yum, I've got your cookie, the contents are: $cookie_test";
 } else {
echo "No cookie found...sorry";
}
?>
```

Listing 3. Deleting the previous sent cookie

```
<?php
# deleting a cookie use only one method!

# delete. With contents of cookie removed
setcookie("cookie_test","",time()+43200,"/");

# delete. With a time that has expired
setcookie("cookie_test","",time()-43200,"/");

?>
```

Info:

PHP homepage
http://www.php.net
Konqueror homepage
http://www.konqueror.org
The Unofficial Cookie FAQ
http://www.cookiecentral.com/faq

Deleting the Cookie

By specifying an expiration time, the cookie will go stale (i.e. unusable) when that time has been breached. However, you may want to delete a cookie before the expiration has been reached. For example, suppose a user joins a club. To save them having to sign in all time you set a cookie that then gets read when the user visits the club. If the cookie is present and the cookie content passes your validation then the user bypasses the club sign-in. Now if the user leaves the club, we might as well take that privilege away from the user, so we need to delete that cookie.

Deleting cookies brings us back to actually setting cookies. When setting cookies, it is always a good idea to think about setting a realistic expiration time when initially setting the cookie, this can save you a lot of hassle in maintaining your cookies, after all we all like low maintenance, don't we. To delete a cookie all you need to do is resend the cookie with the same parameters excluding the value part. Another way of deleting them is to set a cookie as above but with an expiration time that has already expired, so for example suppose you set a cookie with the expiration set to say 24 hours, *time()+86400*. If, after a couple of hours, you decide to delete the cookie, just replace the plus sign with a minus, like so: *time()-86400*. I personally prefer this method, as it is a sure cookie deletion scenario. Listing 3 shows both methods of deleting the cookie sent previously

Listing 4. A simple cookie-based counter script

```
<?php
# test to see if cookie set ?
if (isset($counter)) {
# yes, then add one to counter
$counter++;
} else {
# no, initialise counter
$counter=0;
}
# either way set the cookie !
setcookie("counter",$counter,"", "/");
echo "Example Cookie and Counter Page
";
echo "Counter:[ $counter ]";
?>
```

Simple cookie-based counter

Cookies can be used for many tasks, so let's look at how a cookie can be used as a simple counter. The script in Listing 4 uses cookies to continuously count up when the browser page is refreshed, by sending cookies with the accumulated number. Here's how it works. First a check is made to see if a cookie is present, if it is then the user must have already refreshed/visited the page, so one is added to the variable $counter, using the piece of code $counter++. If a cookie is not present, the user must have just loaded the Web page for the first time, so we set the counter to zero. The next task is to set the cookie. The time expiration is left blank, so the cookie will expire (go stale) when the browser closes down. The cookie is called counter, the value of the cookie is the current value of the variable $counter. Finally the browser outputs a message with that value. If the user has just loaded the page, it will show 0, otherwise it will display the current count based on how many refreshes the user has clicked on. Notice that nothing is outputted to the browser before the cookie is set.

Before we finish with PHP examples here's one final tip: do not leave a space between the "<?php" and the start of line. PHP will interpret that as content to the browser and your cookie will not work.

Conclusion

Cookies are a great way of saving state when a user visits a Web site. They are used in validation, shopping carts, personal greetings, in fact if a Web site knows you, you can bet they are using cookies from information you gave previously in a form. In this month's article we have shown the basics of cookies – how to set, read and delete them – and demonstrated the purposes of cookies. As you can see, cookies are great – when they're not being used to bombard us with targeted advertisements, at least.

URL encoding:

All data streams sent to the browser are URL-encoded by changing the following:

- All spaces are converted to +
- All special characters are converted to their 2 digit HEX number preceded by a %, ie: a (quote) " becomes %22.
- All key/value pairs are separated by &