Cookies in PHP

The internet is a stateless one. That is , web pages don't store data. So when the user log in next time to the same web page, it is like entering the first time entry. I.e the webpage is reinitialized and is displayed anew. But when a user moves from one page of a web site to another page of the same web site it must remember who the user is. For example when a webpage allows its user to customize his webpage to his taste like changing the background color, font size , arranging the tools in a different way, it must remember all this, so that each time when a user login he must have the modified customized page. He should not redo it again and again each time. These details are stored as cookies in user system and whenever he login to that particular site,it will be supplied to the server and after user details are verified the customized page will be displayed.

Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies.
There are three steps involved in identifying returning users:

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

This chapter will teach you how to set cookies, how to access them and how to delete them.

Setting Cookies with PHP:

PHP provided **setcookie()** function to set a cookie. This function has six arguments and only the first argument is compulsory all the others are optional. It should be called before <html> tag. For each cookie this function has to be called separately.

```
setcookie(name [, value[, expire[, path[, domain,[ security]]]]]);
```

Here is the detail of all the arguments:

- **Name -** This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
- **Value -**This sets the value of the named variable and is the content that you actually want to store.
- **Expiry -**This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
- **Path -**This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain -**This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- **Security -**This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

Following example will create two cookies **name** and **age**

Example 1: set_cookie_eg.php

```
<html>
   <head>
      <title>
         Setting a cookie
      </title>
   </head>
<?php
setcookie("name","radha");
setcookie("age","20");
?>
   <body>
      <h1>
         Setting cookie example
      </h1>
      The cookie was set
    </body>
</html>
```

Figure 1

Accessing Cookies with PHP

PHP provides many ways to access cookies. Simplest way is to use either $_COOKIE or $HTTP_COOKIE_VARS variables. Following example will access all the cookies set in above example. You can use **isset()** function to check if a cookie is set or not.

Example 2: read_cookie_eg.php

```
<?php
if (isset($_COOKIE['name']))
{
echo 'Welcome back, ' . $_COOKIE['name'] . ' your age is '.$_COOKIE['age'].'!';
}
 else
 {
  echo 'Hello, new user!';
}
?>
```

The output is given in figure 2.

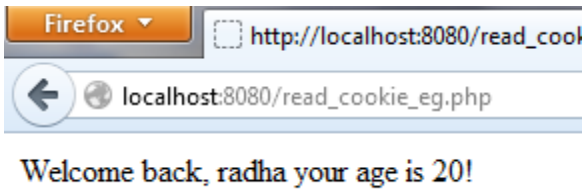

Figure 2

Deleting Cookie with PHP

Officially, to delete a cookie you should call setcookie() with the name argument only but this does not always work well, however, and should not be relied on.

It is safest to set the cookie with a date that has already expired: In the example program the cookie created in example 2 will be deleted in example 3:

Example 3: del_cookie_eg.php

```
<?php
  setcookie( "name", "", time()- 10);
  setcookie( "age", "", time()- 10);
?>
<html>
<head>
<title>Deleting Cookies with PHP</title>
</head>
<body>
<?php echo "Deleted Cookies" ?>
</body>
</html>
```

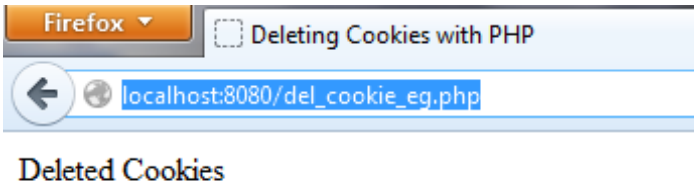

Figure 3

once the cookie is deleted now run the read_cookie_eg.php file again to read the cookie data. Now the output is in figure 4 which shows the cookie is deleted.

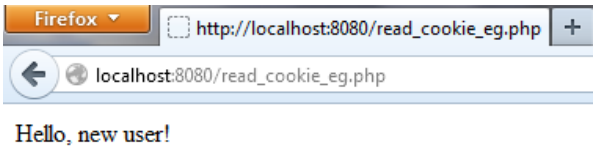

Figure 4

Following example 4 will create two cookies **name** and **age** these cookies will be expired after one hour.

Example 4: phpdelcookie2.php

```
<?php
  setcookie("name", "radha", time()+3600, "/","", 0);
  setcookie("age", "20", time()+3600, "/", "",  0);
?>
<html>
<head>
<title>Setting Cookies with PHP</title>
</head>
<body>
<?php echo "Set Cookies"?>
</body>
</html>
```

The example below sets a cookie with an expiration date of thirty days from now, by setting the expiration to time()+60*60*24*30

```
<?php
setcookie("message","this will be set for 30 days",time()+60*60*24*30);
?>
```

An alternative way to make data accessible across the various pages of an entire website is to use a PHP Session.

Session

The most significant differences between the two are that cookies are stored on the client, while the session data is stored on the server. As a result, sessions are more secure than cookies (no information is being sent back and forth between the client and the server) and sessions work even when the user has disabled cookies in their browser. Cookies, on the other hand, can be used to track information even from one session to another by setting it's time( ) parameter.

A session creates a file in a temporary directory on the server where registered session variables and their values are stored. This data will be available to all pages on the site during that visit.

The location of the temporary file is determined by a setting in the **php.ini** file called

session.save_path. Before using any session variable make sure you have setup this path.

When a session is started following things happen:

- PHP first creates a unique identifier for that particular session which is a random string of 32 hexadecimal numbers such as 3c7foj34c3jj973hjkop2fc937e3443.
- A cookie called **PHPSESSID** is automatically sent to the user's computer to store unique session identification string.
- A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier prefixed by sess_ iesess_3c7foj34c3jj973hjkop2fc937e3443.

When a PHP script wants to retrieve the value from a session variable, PHP automatically gets the unique session identifier string from the PHPSESSID cookie and then looks in its temporary directory for the file bearing that name and a validation can be done by comparing both values.

A session ends when the user loses the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

Starting a PHP Session:
A PHP session is easily started by making a call to the **session_start()** function. This function first checks if a session is already started and if none is started then it starts one. It is recommended to put the call to **session_start()** at the beginning of the page.

Session variables are stored in associative array called **$_SESSION[]**. These variables can be accessed during lifetime of a session.

A session can be started with the function
session_start();

To store data in the session, use the $_SESSION array. For example
example : phpsessionstart.php

```
<?php
session_start();
$_SESSION['username']="radha";
$_SESSION['password']="secret";
?>
<html>
   <body>
   to read your username and password, <a href="phpsession2.php">click here</a>
   </body>
   </html>
```

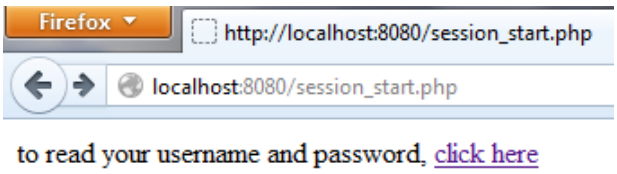

Figure 5

Example : phpsession2.php

```
<?php
session_start();
if(isset($_SESSION["username"]) && isset($_SESSION["password"]))
{
   echo "welcome ".$_SESSION["username"]." your password is ". $_SESSION["password"];

}
?>
```

The output of the above program is given in figure 6

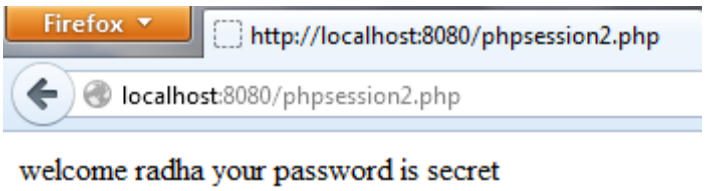

Figure 6

The following example starts a session then register a variable called **counter** that is incremented each time the page is visited during the session.

Make use of **isset()** function to check if session variable is already set or not.
Put this code in a session_count.php file and load this file many times to see the result:

```
<?php
  session_start();
  if( isset( $_SESSION['counter'] ) )
  {
    $_SESSION['counter'] += 1;
  }
  else
  {
    $_SESSION['counter'] = 1;
  }
  $msg = "You have visited this page ".  $_SESSION['counter'];
  $msg .= "in this session.";
?>
<html>
<head>
<title>Setting up a PHP session</title>
</head>
<body>
<?php  echo ( $msg ); ?>
</body>
</html>
```

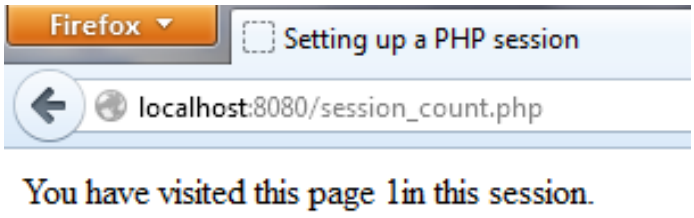

Figure 7

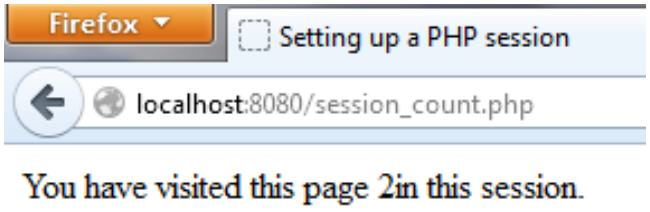

Figure 8

Destroying a PHP Session:

A PHP session can be destroyed by **session_destroy()** function. This function does not need any argument and a single call can destroy all the session variables. If you want to destroy a single session variable then you can use **unset()** function to unset a session variable.
Here is the example to unset a single variable:

```
<?php
   unset($_SESSION['userName']);
?>
```

Here is the call which will destroy all the session variables:

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

Turning on Auto Session:

You don't need to call start_session() function to start a session when a user visits your site if you can set **session.auto_start** variable to 1 in **php.ini** file.

Sessions without cookies:

There may be a case when a user does not allow to store cookies on their machine. So there is another method to send session ID to the browser.

Alternatively, you can use the constant SID which is defined if the session started. If the client did not send an appropriate session cookie, it has the form session_name=session_id. Otherwise, it expands to an empty string. Thus, you can embed it unconditionally into URLs.

The following example demonstrates how to register a variable, and how to link correctly to another page using SID.

```
<?php
   session_start();

   if (isset($_SESSION['counter'])) {
      $_SESSION['counter'] = 1;
   } else {
      $_SESSION['counter']++;
   }
?>
   $msg = "You have visited this page ".  $_SESSION['counter'];
   $msg .= "in this session.";
   echo ( $msg );
<p>
To continue  click following link <br />
<a  href="nextpage.php?<?php echo htmlspecialchars(SID); >">
</p>
```

The **htmlspecialchars()** may be used when printing the SID in order to prevent XSS related attacks.

PHP Session Array- Method 1:

```
<?php
session_start();
 $_SESSION['userName']='radha';
 $_SESSION['dept']='csc';
 $_SESSION['LoggedIn']=true;
 $_SESSION['desig']='Assoc.Prof';
 $_SESSION['gender']='female';
 var_dump($_SESSION);
?>
```

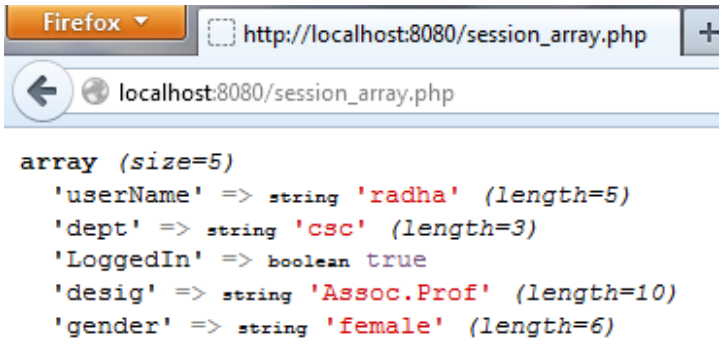

Figure 9

If you store all of the data like this then it will mix with various other kind of data. So, it can be better stored as:

PHP Session Array- Method 2:

```
<html>
   <title> session array</title>
   <h1> use of session variables as array</h1>
</html>
<?php
session_start();
$user=array();
 $user['userName']='radha';
 $user['dept']='csc';
 $user['LoggedIn']=true;
 $user['desig']='Assoc.Prof';
 $user['gender']='female';
$_SESSION['user']=$user;
 var_dump($_SESSION['user']);
?>
```

This is same as:

Figure 10

PHP Session Array- Method 3:

```
<?php
$_SESSION['user']['userName']  = 'radha';
$_SESSION['user']['loggedIn']  = true;
$_SESSION['user']['ranking']   = 5;
var_dump($_SESSION['user']);
//unset($_SESSION['user']);
?>
```

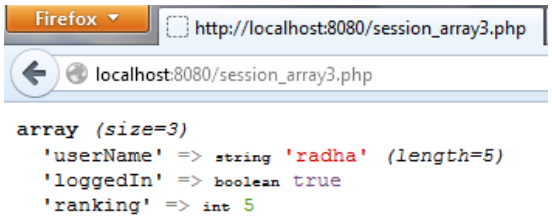

Figure 11

All Logged-In user data, which are organized inside array, are in user key of a session variable, so it is easy to maintain.

```
<?php
// session_remote.php
// record remote client name as session variable
session_start();
$_SESSION['verif_remote_agent'] = $_SERVER['HTTP_USER_AGENT'];
echo 'Session started. Your session ID is: ' . session_id();

// now try check-session.php
?>
```

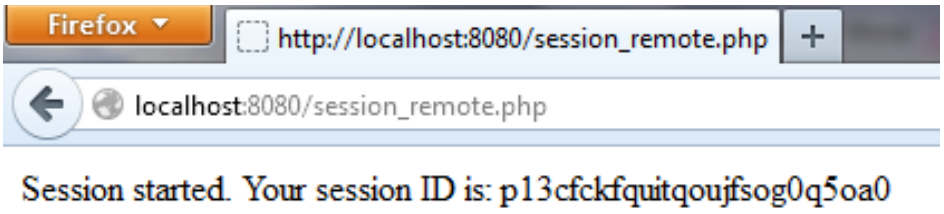

Figure 12

```
<?php
// session_remote_check.php
// also check remote client name
session_start();
if (!isset($_SESSION['verif_remote_agent']) || $_SESSION['verif_remote_agent'] !=
$_SERVER['HTTP_USER_AGENT']) {
  die('Session check failed.');
} else {
  echo 'Session verified.' . session_id();
}
?>
```

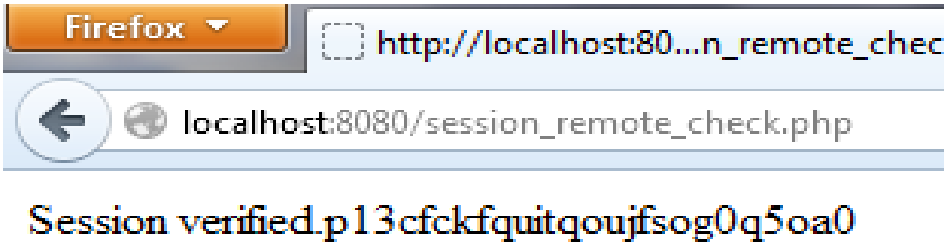

Figure 13