MAINTAINING STATES IN PHP

Today – Useful PHP

1. Passing Variables Between Scripts
2. Cookies
3. Sessions
4. Redirection
5. Making a Web Mail system

HTTP is stateless

- A fundamental characteristic of Web is:
  - The stateless interaction between browsers and web servers
  - Each HTTP request sent to a web server is independent of any other request.

- Applications that require complex user interaction can't be implemented as a series of unrelated, stateless web pages.

- An often-cited example is a shopping cart in which items are added to the cart while searching or browsing an on-line store. The state of the shopping cart (the selected items) needs to be stored somewhere to be displayed when the user visits the order page.

HTTP is stateless

- There are three ways to build an application that keeps state:
  - Variables can be passed between scripts as query string appended with the URL.
  - Variables can be stored in the browser at client-side as cookies and then can be included with each request.
  - variables can be stored on the server as session variables

1. Passing Variables Between Scripts

- The simplest way is to add the variables to the url:

  www.seecs.edu.pk/myscript.php?**variable=value**

- You can chain these variables using an ampersand.

  myscript.php?**variable1=value1&variable2=value2&…**

- As per normal you can access these variables easily. You just need to use $_GET array, and access variables through their names. E.g

  $var1 = $_GET[“variable1”];

URL encoding

- However this can lead to problems as you can't have certain characters in url's – spaces for example, more ampersands, colons and so on.

- To deal with this php has the **urlencode()** function.This converts all those problem characters into their url friendly counterparts. E.g.

```
<?
   $str = urlencode("script.php?name=T. J.&lastname=O'Reilly");
   print "<A HREF=$str>link</A>";
?>
```

```
<A HREF=script.php%3Fname%3DT.+J.%26lastname%3DO%27Reilly>link</A>
```

2. Cookies

- Setting and playing around with cookies can be a fun and useful way to save data on a user's hard drive.
- It can successfully store valuable information which may be helpful the next time they come to the site.
- Its fairly simple to set up, and even easier to read. To use it, you have to remember some guidelines…

Guidelines

1. You have to put the cookie code before you print out any other HTML in your script.

2. The cookie will not be evident on the page until its refreshed, or the user visits the page again (It is sent **with** the current page data)

- Here's the code to set a variable:

```
<?
   setcookie (“loginName", “Jimbo");
?>
```

$$\underbrace{\text{"loginName"}}_{\boxed{\text{VARIABLE NAME}}} \qquad \underbrace{\text{"Jimbo"}}_{\boxed{\text{VALUE}}}$$

Cookie Expiration

- Now, the next time someone visits this page, or any other PHP page that cookie variable will be available.

- However by default this cookie will expire when the user turns his browser off.

- To extend the time to expire, set in seconds as the next field. For example:

```
<?
    setcookie ("loginName", "jimbo", time()+3600);
?>
```

EXPIRES IN 1 HOUR

Time Conversion table

1 minute	-	60s
1 hour	-	3600s
1 day	-	86400s
1 week	-	604800s
1 fortnight	-	1209600s
1 month	-	2419200s
3 month	-	7257600s
1 year	-	29030400s

Reading Cookie Information

- The cookies for the web domain your page is in will be **automatically** loaded into PHP.

- You can get at them via two arrays:

```
$HTTP_COOKIE_VARS["loginName"];
```

or

```
$_COOKIE["loginName"];
```

- So to display the cookie data on screen all you need is:

```
<?
    print $_COOKIE["loginName"]
?>
```

Practical Cookies : User Prefs

User_prefs.php

```
<?php
if(!$_COOKIE["body_color"])
{
setcookie("body_color", "#000000", time()+3600);
setcookie("text_color", "#FFFF00", time()+3600);
}
?>
<HTML>
<BODY BGCOLOR=<?php echo $_COOKIE["body_color"]?> text=<?php
   echo $_COOKIE["text_color"] ?> >
Hello buddy
</BODY>
<HTML>
```

Multiple Cookies

- It is not a problem to have multiple cookies - save it, here is a code example:

```
<?
    setcookie (“loginName", “jimbo");
    setcookie (“password", “bosh");
    setcookie (“hits", “3");

    print $_COOKIE[“loginName”]."<BR>”;
    print $_COOKIE[“password”]."<BR>”;
    print $_COOKIE[“hits”]."<BR>”;
?>
```

Deleting Cookies – Reading

- There are two ways of deleting cookies. The traditional way

```
<?
    setcookie ("cookie", "", time()-86400);
?>
```

Or simply by setting the cookie as nothing:

```
<?
    setcookie ("cookie");
?>
```

Don’t use multiple cookies

- As such it is viewed as bad coding to use more than one cookie, and so people tend to store all variables they need in ONE cookie.
- This is easy in PHP because of the **explode()** and **implode()** commands.

Exploding Cookies

- As I said before you can also use implode and explode.

```
$info[0] = "Jimbo";
$info[1] = "bosh";
$cookie = implode($info, "-");
setcookie ("myCookie", $cookie, time()+86400);
```

- And you can take them out as follows

```
$cookie = $_COOKIE['myCookie'];
$info = explode($cookie, "-");
```

- Of course you need to remember that element 0 of the info array is the username and element 1 is the password. But this way you can build up huge cookies.

Problems with Cookies

- Not only are cookies painful to code.
- It may seem a suprisingly low statistic, but Cookies are about **30% unreliable on the web** right now and it's getting worse.
- More and more web browsers are starting to come with security and privacy settings and people browsing the net these days are starting to frown upon Cookies because they store information on their local computer that they do not want stored there.

PHP Maintaining state

- We now have two ways of maintaining state – of keeping variables common between scripts.
  - Adding variables to the url
  - Storing variables in cookies

- Neither are satisfactory. One is incredibly clumsy the other out of synch. Your cookie is always one step behind because you send it out with each page.

- So whats the answer? **Sessions!**

The Session Solution

- PHP has a great set of functions that can achieve the same results of Cookies and more without storing information on the user's computer.
- PHP Sessions store the information on the web server in special files.
- These files are connected to the user's web browser **via the server** and a special ID called a "**Session ID**".
- *This is nearly 99% flawless in operation and it is virtually invisible to the user.*

Session Start

- The correct way to start a session is using the **session_start()** command.
- We must include this statement at the start of every script of our site that we want to be able to use session variables in.

```
<?
     session_start();
     print "We have started our session:";
?>
```

- This is essential and an easy thing to forget.

A common error

- Just like Cookies you MUST call the session_start() function before anything is output to your web browser. This is absoultey important because you will get some ugly errors by PHP that will say something like this:

```
<?
   echo "This is incorrect and will cause an error:";
   session_start();
?>
```

Generates the error:

Warning: Cannot send session cookie - headers already sent by (output started at session_header_error/session_error.php:2) in session_header_error/session_error.php on line 3

Assigning Variables

```
<?
   // start the session
   session_start();
   print "Registering a session";

   // Get the user's input from the form for example
   $data = $_POST['data'];

   // Create a new Session variable. You can skip this step. A
   variable can be automatically registered when you assign it a
   value.
   session_register('name');

   // way of putting data into the variable. If variable 'name' is
   not already registered, then it will be automatically registered
   and assigned a value here.
   $_SESSION['name'] = $data;

?>
```

```
   Welcome to my website <? print $_SESSION['name'] ?><BR>
   This is an example of receiving a data variable from an HTML form
   and putting it in the session.
```

Sessions on Multiple Pages

- The first thing you **MUST** do on each page you want to access a session variable is to start the session.
- That may not sound right to you because "We already started the session on the last page."
- That's true, but we need to keep the "connection" going between our session because they do not have persistent connections like MySQL does.

A Multiple Page Session

```
<?
   // start the session
   session_start();
   print "In this script we use session variables";
   print "that we created in the previous script<br>";

   // display the session variable
   print "Hi there $_SESSION['name'] everything is working
   fine! <br>";


?>
```

Unregistering Session Variables

- PHP is really well designed.
- With PHP Sessions, we have the ability to simply remove a single session variable without dumping our entire session and rebuilding it. The function is called **session_unregister()**
- Here's how we unregister a single session variables and leave the rest intact.

```
session_unregister('name');
```

Destroying a Whole Session

- Why might it be necessary to destroy a session when the session will get destroyed when the user closes their browser?

- Well, Imagine that you had a session you were using to determine if the user was logged into your site based upon a username and password - anytime you have a login feature, to make the users feel better, **you should have a logout feature as well.**

- That's where **session_destroy()** may be useful – *it will delete the session files and clears any trace of that session.*

Practical Sessions : Hit Counter

- What we're about to do here is:
  - start your session
  - register a variable called "count“
  - assign a value of 1 to it on the first page.
  - Then, we're going to increment the counter as we go through the website.
  - We’re also going to provide a reset page

Hit Counter – counter page

hit_counter.php

```
<?
   session_start();
   if (!$_SESSION['count']) // or if(isset($_SESSION["count"]))
        session_register('count');
   if($_SESSION['count'] == 0)
        $_SESSION['count'] = 1;
   else
        $_SESSION['count']++;
?>

You've visited <?=$_SESSION['count']?> pages so far!<br>
<a href="hit_counter.php">Increment Your Counter!</a><br>
<a href="reset.php">Reset Your Counter!</a><br>
```

Hit Counter – Reset Page

reset_counter.php

```
<?
   session_start();
   session_register('count');
   $_SESSION['count'] = 1;
?>
You've visited <?=$_SESSION['count']?> pages so far!<br>
<a href="hit_counter.php">Increment Your Counter!</a><br>
<a href="reset_counter.php">Reset Your Counter!</a><br>
```

Pretty easy!?

Viewing Your Session ID

- Every Session has a unique Session ID. A session ID looks like some chatting guru collapsed on the keyboard.

- There's a function in PHP called **session_id()** that allows you to display the current session ID or utilize it however you need.

```
<?
   session_start();
   echo "Your session ID is <B>". session_id() ."</B>";
?>
```

- This will simply display something like:

Your session ID is **Bd315d2ed59dfa1c2d0fb0b0339c758d**

Practical Sessions : User Prefs

User_prefs.php

```
<?
   session_start();
   if((!$_SESSION["body_color"])||(!$_SESSION["text_color"])) {
        $_SESSION["body_colour"] = "#000000";
        $_SESSION["text_colour"] = "#FFFFFF";
   }
?>
<HTML>
<BODY BGCOLOR=<?=$_SESSION["body_colour"]?>
   TEXT=<?=$_SESSION["body_colour"] ?> >
…
```

IE6 Session Problem

- When you click your back button to make changes in the form, you have to click the REFRESH button on that page to get the information that you posted back into the form.

- This only works about 50% of the time. The other 50% the users information is lost

- This can be horrific for users… but there is a simple solution. Enter this right below the **session_start()** of each script:

```
header("Cache-control: private");
```

Discussion of Prefs

- Now this is all great at the moment but we do have a problem – a session automatically closes when a user shuts his web browser.

- If that person has spent hours setting all their user preferences and they disappear when the browser is closed you aren't going to get many repeat users.

- So while sessions maintain state over a visit we need someway of storing data between visits….

- One solution is to store such valuable user preferences in the database for repeated users. And when these users come again to visit your site, simply fetch these values from database and assign them to session variables.

Redirection

- Just like Cookies and Sessions, you MUST call the header() function before anything is output to your web browser. Otherwise you will get a famous error message i.e. Headers already sent etc.

```
<?php
   header("Location: http://www.example.com/");
?>

<?php
   header("Location: myApp/login.php");
?>
```

3. Web Mail Systems

- Its easy to send emails in php too.

- **Mail()** function uses SMTP (Simple Mail Transfer Protocol) to send emails automatically from inside your scripts.

- To receive and process mail PHP can use the IMAP protocols (we won't go into this).

- PHP comes with the IMAP library and this can be used for POP and NNTP (news) connections.

Sending a mail…

```
<?

  $email   = “falak.nawaz@seecs.edu.pk";
  $title   = “More SPAM!”;
  $message = “This is my first\n PHP mail message”;
  $from    = "From: falak@msn.com\n";

  mail($email, $title, $message, $from);

?>
```

LOONEY TUNES
"That's all Folks!"
A WARNER BROS. CARTOON