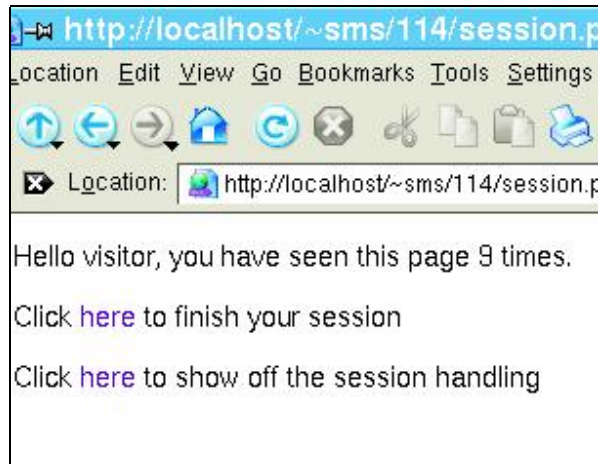


PHP Sessions

HTTP is what is known as a stateless protocol, which makes designing easy-to-use web apps a nightmare - unless you use sessions



We've all done this: wanting to post a reply on a forum, or to read our email using a Web-based front end, and having to log into the Web site before we can do anything else. Once that's out of the way, we don't need to do it again, until either a certain amount of time has gone by without any more page requests, or we restart the browser. This is a really clever trick, because HTTP is stateless; that is, once a Web server has sent a page it retains no further memory of you. So, how's it done?

All that's required is some means to keep 'state', such as your username available between page requests. The simplest way to do this involves putting your data into hidden form fields, but this isn't always practical. The next and most obvious method involves storing all the information as cookies on the client's computer, but this has security implications.

The most secure way would be to give each session a unique ID and send this ID backwards and forwards between the browser and the server, allowing the server to use the ID to look up state as required. This is how PHP implements sessions, where the ID is sent either encoded into a URL or as a cookie, with the latter being used in preference.

When we use the header function to redirect the Web browser to another page, we can't use POST variables. But if we want to pass along the information in the POST variables we can store them in SESSION variables. When we navigate to the new page they will be available.

The first thing that you MUST do is call the `session_start()` function before anything is output to your Web browser. By anything, this could be HTML, blank lines or spaces outside of PHP tags, echo or print statements, or include files that do any of the these things. This is important because you will get some ugly errors that will say something like this:

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

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

The above error was generated by this code:

```
<?php
echo "Look at this nasty error below:<br />";
session_start();
?>
```

Why did this happen? Because I called the echo "Look at this nasty error below:
"; before I called the **session_start()**; function. If the lines were swapped around I would not have received this error. The correct way to start a session is like this:

```
<?php
session_start();
echo "Look at this nasty error below:";
?>
```

The output of this would have simply been: "Look at this nasty error below:" and no errors would have been generated.

Microsoft Internet Explorer Sessions Fix

There's a problem with IE6 when you are using sessions and you post to a form, 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 user's information is lost and they have to type it over again. Not a good thing if you are trying to get the person to enter their billing information to process an order. They might just get irritated and leave. So, here's a solution for that. Enter this right below the session_start() of each script (yes it still must be before anything is output to the browser).

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

Now your users can hit the back button and change information all they want!

So, now that we have all of that out of the way, let's move on to starting sessions, registering session variables and assigning values to those variables.

Starting Your Session and Assigning Variables

The first thing we need to do as you probably figured out is to start your session. This is as simple as typing the following at the top of your scripts:

```
<?php
session_start();
header("Cache-control: private"); // IE 6 Fix.
?>
```

Now we have told PHP to start a session and begin storing information about the user. So, let's say that we had an input form from a page that asks the user their name. This form will post to a PHP script that will get the post information and register it as a session variable that we can use throughout our website until the user leaves the site or until we unregister that variable. Here's the form:

```
<FORM METHOD="POST" ACTION="page2.php">
Enter your Name: <input type="text" name="name">
<input type="SUBMIT" value="Submit">
</FORM>
```

It is just an HTML form and it will post to a script called "**page2.php**" and we'll look at that now.

```
<?php
// start the session
session_start();
header("Cache-control: private"); //IE 6 Fix

echo "<strong>Step 2 - Register Session </strong><br />";

// Get the user's input from the form
$name = $_POST['name'];

// Register session key with the value
$_SESSION['name'] = $name;

// Display the session information:
?>

Welcome to my website <strong><? echo $_SESSION['name']; ?></strong>!<br />
Let's see what happens on the <a href="page3.php">next page.</a><br /><br />
```

First, we've started the session and used the header cache control. Then we created a \$name variable and assigned the value of \$_POST['name'] to it. Then we assigned \$name to a newly created session key identified as 'name'.

You could do it this way instead:

```
// Register the input with the value
$_SESSION['name'] = $_POST['name'];
```

The next code block we will look at is where we test the session variable that was just registered and assigned a value to.

```
// Display the session information:
?>
```

```
Welcome to my website <strong><? echo $_SESSION['name']; ?></strong>!<br />
Let's see what happens on the <a href="page3.php">next page.</a><br /><br />
```

If everything worked properly and you typed "Bob" in the form on the first page, you'll see:

Welcome to my website Bob!

Using Sessions on Multiple Pages

The reason we register sessions is to avoid reading cookies and querying databases for information about the user on each page we need that information on. So, let's see how that is done.

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 you may be thinking "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. Start your session on each page.

Here's the next script called "page3.php":

```
<?php
// start the session
session_start();
header("Cache-control: private"); //IE 6 Fix
?>
<strong>Step 3 - Test Session Part II </strong><br />
Hey <strong><? echo $_SESSION['name']; ?></strong> Everything is still
working!<br /><br />
<strong>Pick an option:</strong><br />
Let's delete this session value now. <a href="page4.php">Click Here.</a><br
/>
Let's destroy this session. <a href="page5.php">Click Here.</a><br /><br />
```

We started the session, called the session variable and echoed "Hey Bob Everything is still working!" in the user's browser. Now you can see how the session variables can follow be used on multiple pages.

You may also notice on this script that there are a couple of hyperlinks to two more scripts. The first one "Let's delete this session value now" will do just that.

Unregistering Session Variables

With PHP Sessions, we have the ability to simply remove a single session variable without dumping our entire session and rebuilding it. We can simply do this by assigning a blank value or a 'false' Boolean to the session key we want to get rid of. Here's how to unregister a single session variable and leave the rest intact.

```
<?php
// start the session
session_start();
header("Cache-control: private"); //IE 6 Fix

unset($_SESSION['name']);
// or $_SESSION['name'] = '';
// or $_SESSION['name'] = FALSE;
echo "<strong>Step 4 - Unregister Session </strong><br />";
if($_SESSION['name']){
    echo "The session is still registered.<br /><br />";
} else {
    echo "Ok, the session is no longer registered! <br />";
    echo "<a href=\"pagel.php\"><< Go Back Step 1</a><br /><br />";
}
?>
```

Any time we deal with sessions, we have to run the **session_start()** function first. As you see in this code, we have done just that. Let's talk about the line in this example which removes a session value.

```
unset($_SESSION['name']);
```

We just cleared out that session variable called "name" and it is no longer available to us until we set it again.

To ensure that everything worked, we can use an IF statement to determine if the value of `$_SESSION['name']` is set, true or not false. If it is, we will display "The session is still registered." to the browser. Here it is:

```
if($_SESSION['name']){
    echo "The session is still registered.<br /><br />";
} else {
    echo "Ok, the session is no longer registered! <br />";
    echo "<a href=\"pagel.php\"><< Go Back Step 1</a><br /><br />";
}
```

If the session is registered "name" then tell the user 'The session is still registered' otherwise tell the user 'Ok, the session is no longer registered' and give them a hyperlink back to Step 1."

We should only see "Ok, the session is no longer registered!" and a hyperlink back to step 1.

Destroying a Session

You may be wondering why it is necessary to destroy a session when the session will get destroyed when the user closes their browser. Well, imagine that you had a session registered called "access_granted" and you were using that 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 this cool function called **session_destroy()** comes in handy. **session_destroy()** will completely demolish your session (no, the computer won't blow up or self destruct) but it just deletes the session files and clears any trace of that session.

NOTE: If you are using the `$_SESSION` superglobal array like the above examples, you must clear the array values first before running `session_destroy`.

Here's how we use **session_destroy()**:

```
<?php
// start the session
session_start();
header("Cache-control: private"); //IE 6 Fix
$_SESSION = array();
session_destroy();
echo "<strong>Step 5 - Destroy This Session </strong><br />";
if($_SESSION['name']){
    echo "The session is still active";
} else {
    echo "Ok, the session is no longer active! <br />";
    echo "<a href=\"page1.php\"><< Go Back Step 1</a>";
}
?>
```

You must still run the **session_start()** function so that PHP knows which session to destroy. To destroy a session, do the following:

```
<?php
session_start();
$_SESSION = array();
session_destroy();
?>
```

Let's go ahead and use this practically with a session hit counter to display how many pages the user has clicked on during their visit to our site.

Practical Use of Sessions: Page Hit Counter

What you do is start your session, register a variable called "count" and 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. Here's the code for the script.

```

<?php
// start the session
session_start();
header("Cache-control: private"); //IE 6 Fix
if(!$SESSION['count']){
    $SESSION['count'] = 1;
} else {
    $SESSION['count']++;
}
?>
You have visited <? echo $SESSION['count']; ?> pages so far!<br /><br />
<a href="page1.php">Increment Your Counter!</a><br />
<a href="page2.php">Reset Your Counter!</a><br /><br />

```

We've started our session, registered "count" as a variable and then did some checking on the count variable before we assigned any values to it. The reason I did this is because let's say that this was included on our index page. I didn't want my users counter to reset to **1** when I browse 10 pages deep into my website and when they come back to the index page to start digging in a different section. Let's summarize the check that we performed below:

"If the session variable count has no value or is equal to **0**, set it to a value of **1**, otherwise increment it each time this code is called."

Using the `$SESSION['count']++`; we are simply saying "Add 1 to the current value of "count".

Next, we've given the user a page to go and reset their counter. This is nearly the same code except we aren't doing the error checking above because we want to reset their counter to 1 when they access that page, thus resetting their counter.

```

<?php
// start the session
session_start();
header("Cache-control: private"); //IE 6 Fix
$SESSION['count'] = 1;
?>
You have visited <? echo $SESSION['count']; ?> pages so far!<br /><br />
<a href="page1.php">Increment Your Counter!</a><br />
<a href="page2.php">Reset Your Counter!</a><br /><br />

```

We could combine these two scripts easily by just echoing the appropriate values we assigned in the other script if the user actually entered his/her name in the text box of the form. You can gather and set variables and assign values on multiple scripts throughout your website to the same session.

Viewing Your Session Variables and Values

```
<?php
session_start();
header("Cache-control: private"); //IE 6 Fix
echo "Sessions: <pre>";
print_r($_SESSION);
echo "</pre>";
?>
```

This script is pretty straight forward and will give you all the information you need to know about what's in your session's scope.

Viewing Your Session ID

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

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

This will simply display something like:

Your session ID is **bd315d2ed59dfa1c2d0fb0b0339c758d**