



**Simple Debugging in PHP**  
**by Ray Borduin**

## Simple Debugging in PHP

Many hosts turn off error reporting on the page for security reasons. Errors can sometimes contain code, so it is generally recommended to turn off errors once your site is live and only leave it on when debugging locally or when you are actively debugging problems.

As a result of this setting being turned off, php pages can come up blank when there is an error on the page or you will see a 500 error message, which is a general error message that doesn't give any real details about the error.

In order to enable you to debug, to try to view the errors since they may include line numbers which you can use to really home in on the error and identify the cause. You can do this by either changing the default settings in the php.ini file, or by adding the following to the top of the .php file:

```
<?php
error_reporting(E_ALL);
ini_set('display_errors',1);
?>
```

If that doesn't work you may want to find your error log files or talk to your hosting provider about enabling error handling on your server in your php.ini file while you are developing; then have it turned off once your site is live.

The most important part of enabling error handling is to get line numbers to identify where the error occurs. Without line numbers, finding an error is like finding a needle in a haystack - you have to be meticulous and spend a lot of time to be successful.

Once you have a file name and line number you may need someone that understands php to help from there; however there are a few common errors you might be able to fix with very limited experience.

```
// output already started
```

```
// file not found
```

Sometimes more advanced debugging is necessary. Most debugging can be done with three or four simple php functions:

### **phpinfo() function:**

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

Adding this to a page will create a table with all of the configuration information about your php installation and settings. It will include the location of the php.ini file and list all of the installed components and their versions and settings.

### **echo() function:**

```
<?php
echo($variable);
?>
```

The echo function can be used to write the value of a string variable to the page. It will write the value and continue processing the page. This should be generally used inside the html body to prevent the "output already started" message described previously.

### **die() function:**

```
<?php
die($variable);
?>
```

The die function works like the echo function, but it stops processing the page after writing the value. This means you can use it above the html tag without having to worry about it causing errors later since the script stops running as soon as the die statement executes.

For instance, if you are trying to determine whether a page is opening at all or if you have permissions problems or a bad php install preventing the page from working you can add code to the first line of the page:

```
<?php
die("no permissions or php install issues");
?>
```

If you see this text when you visit the page you can be sure that php is installed and that there are no permissions problems preventing the page from being viewed.

### **var\_dump() function:**

```
<?php
var_dump($array);
die();
?>
```

The var\_dump function is a lot like the echo function, but it can be used for complex data structures and not just string variables. With var\_dump you can output all of the data in an array at once instead of echo() for single values.

There is no var\_dump equivalent to the die function, so to stop the code from processing you can just add a die statement below the var\_dump as in my example to stop the code from continuing. This would usually be done when your var\_dump is above the <html> tag and isn't usually necessary when in the page body.

You can use this to write recordset rows or other complex data. When using WebAssist extensions and code, we often store key errors and data in the session, so adding the following to your page will often give you more information about an email, upload, or other transaction that isn't working properly and may be influential in debugging and fixing the problem.

```
<?php  
var_dump($_SESSION);  
?>
```

Any information you can find out about a problem can help you understand the issue and eventually fix it. I don't recommend taking stabs at fixing errors you don't understand in the first place. Do Internet searches on errors you find to see what could be causing them and try to fully understand your problem before working on a solution.

If you are unable to find a line number, adding die statements progressively lower on the page can help you find the error line through the process of elimination. You can start by adding this code to the first line of the page:

```
<?php  
die("here");  
?>
```

Then view the page to see the word "here" then progressively move the die code down the page, maybe 10 or 100 lines at a time (depending on the number of lines). Once you don't see the text, you know the error is in between those lines. Keep repeating this until you find the exact line of code with the error. You can then analyze that line by adding var\_dump or die statements above it to see the values being referenced and fully understand the problem you are looking at.