### [PHP: Debugging in PHP - Manual](http://php.net/manual/en/debugger.php)

php.net/manual/en/debugger.php

[Dịch trang này](https://translate.google.com.vn/translate?hl=vi&sl=en&u=http://php.net/manual/en/debugger.php&prev=search)

I am a firm believer in the Firephp **debugger**. It works with Firefox and Firebug to allow you to see the value of any string, array, or object. The best part of it is that ...

### [How to Debug in PHP - Treehouse Blog](http://blog.teamtreehouse.com/how-to-debug-in-php)

blog.teamtreehouse.com/how-to-debug-in-php

[Dịch trang này](https://translate.google.com.vn/translate?hl=vi&sl=en&u=http://blog.teamtreehouse.com/how-to-debug-in-php&prev=search)

7 thg 8, 2008 - This article breaks down the fundamentals of **debugging** in **PHP**, helps you understand**PHP's** error messages and introduces you to some ...

### [eclipse - How do you debug PHP scripts? - Stack Overflow](http://stackoverflow.com/questions/888/how-do-you-debug-php-scripts)

stackoverflow.com/questions/888/how-do-you-debug-php-scripts

[Dịch trang này](https://translate.google.com.vn/translate?hl=vi&sl=en&u=http://stackoverflow.com/questions/888/how-do-you-debug-php-scripts&prev=search)

Please edit the question to limit it to a specific problem with enough detail to identify an adequate answer. Avoid asking multiple distinct ...

### [Debugging techniques for PHP programmers - IBM](https://www.ibm.com/developerworks/library/os-debug/)

https://www.ibm.com › Learn › Open source

[Dịch trang này](https://translate.google.com.vn/translate?hl=vi&sl=en&u=https://www.ibm.com/developerworks/library/os-debug/&prev=search)

29 thg 11, 2005 - Explore various methods for **debugging PHP** applications, including turning on error reporting in Apache and PHP, and by placing strategic ...

### [Debugging PHP Source Code in the NetBeans IDE for PHP Editor](https://netbeans.org/kb/docs/php/debugging.html)

https://netbeans.org/kb/docs/php/debugging.html

[Dịch trang này](https://translate.google.com.vn/translate?hl=vi&sl=en&u=https://netbeans.org/kb/docs/php/debugging.html&prev=search)

Getting Ready. To successfully **debug PHP** applications in the NetBeans IDE for PHP, you need to have the PHP engine, the Apache local web server, and the ...

### [PHP Console - Chrome Web Store](https://chrome.google.com/webstore/detail/php-console/nfhmhhlpfleoednkpnnnkolmclajemef)

https://chrome.google.com/.../php.../nfhmhhlpfleoednkpnnnkolmclaje...

[Dịch trang này](https://translate.google.com.vn/translate?hl=vi&sl=en&u=https://chrome.google.com/webstore/detail/php-console/nfhmhhlpfleoednkpnnnkolmclajemef&prev=search)

Xếp hạng: 4,5 - ‎291 phiếu bầu - ‎Miễn phí - ‎Chrome

4 thg 11, 2016 - Display **PHP** errors & vars dumps in Google Chrome console and ... in Address bar ✓**PHP** errors, exceptions and **debug** data in JavaScript ...

### [php-debug - Atom](https://atom.io/packages/php-debug)

https://atom.io/packages/php-debug

[Dịch trang này](https://translate.google.com.vn/translate?hl=vi&sl=en&u=https://atom.io/packages/php-debug&prev=search)

Atom PHP Debugging Package. **Debug PHP** code using the Xdebug PHP Extension. Features. Add Breakpoints; Step through debugging (Over, In, Out); Stack ...

### [PhpStorm 2016.3 Help :: Debugging with a PHP Web Application ...](https://www.jetbrains.com/help/phpstorm/2016.3/debugging-with-a-php-web-application-debug-configuration.html)

https://www.jetbrains.com/.../phpstorm/.../debugging-with-a-php-web-...

[Dịch trang này](https://translate.google.com.vn/translate?hl=vi&sl=en&u=https://www.jetbrains.com/help/phpstorm/2016.3/debugging-with-a-php-web-application-debug-configuration.html&prev=search)

**Debugging** with a **PHP** Web Application **Debug** Configuration. In this section: Introduction; Preparing the **debugging** engine; Setting breakpoints; Creating a ...

### [Debug PHP with Firebug and FirePHP - SitePoint](https://www.sitepoint.com/debug-php-firebug-firephp/)

https://www.sitepoint.com/debug-php-firebug-firephp/

[Dịch trang này](https://translate.google.com.vn/translate?hl=vi&sl=en&u=https://www.sitepoint.com/debug-php-firebug-firephp/&prev=search)

9 thg 2, 2010 - The little 'bug is a fantastically useful HTML/CSS/JavaScript/Ajax debugger. But did you know it can also be used to **debug PHP**? Yes, thanks to ...

### [GitHub - felixfbecker/vscode-php-debug: PHP Debug Adapter for ...](https://github.com/felixfbecker/vscode-php-debug)

https://github.com/felixfbecker/vscode-php-debug

[Dịch trang này](https://translate.google.com.vn/translate?hl=vi&sl=en&u=https://github.com/felixfbecker/vscode-php-debug&prev=search)

**PHP Debug** Adapter for Visual Studio Code ⛔. Contribute to vscode-**php**-**debug** development by creating an account on GitHub.

### [Debug và Profiling PHP bằng Xdebug - Techmaster](https://techmaster.vn/posts/5816/hoc-lap-trinh-web-bang-ngon-ngu-php-2)

https://techmaster.vn › Blog

18 thg 3, 2014 - **PHP** là một ngôn ngữ phổ biến nhất cho các nhà lập trình web, nhưng lại gặp sự chỉ trích rất lớn rằng nó thiếu một chương trình **debug** phù ...

### [Xdebug - Debugger and Profiler Tool for PHP](https://xdebug.org/)

https://xdebug.org/

[Dịch trang này](https://translate.google.com.vn/translate?hl=vi&sl=en&u=https://xdebug.org/&prev=search)

A **PHP** extension for powerful **debugging**. It supports stack and function traces, profiling information and memory allocation and script execution analysis.

### [phpdbg | php debugger](http://phpdbg.com/)

phpdbg.com/

[Dịch trang này](https://translate.google.com.vn/translate?hl=vi&sl=en&u=http://phpdbg.com/&prev=search)

The **debugging** platform for PHP5.4+ ... The power of **PHP** available to every command issued ! ... phpdbg can act as a **debugging** server, Java client included.

### [Hướng dẫn debug PHP project - Viblo | Free service for technical ...](https://viblo.asia/nguyen.the.linh/posts/MJykjQbJePB)

https://viblo.asia/nguyen.the.linh/posts/MJykjQbJePB

24 thg 9, 2016 - Hôm nay mình xin trình bày với các bạn cách **debug** các dự án sử dụng **PHP** và IDE PHPSTORM. Với Sublime text các bạn có thể tham khảo ...

### [Debugging a PHP Web Page - Eclipse](http://www.eclipse.org/pdt/help/html/debugging_a_php_web_page.htm)

www.eclipse.org/pdt/help/html/debugging\_a\_php\_web\_page.htm

[Dịch trang này](https://translate.google.com.vn/translate?hl=vi&sl=en&u=http://www.eclipse.org/pdt/help/html/debugging_a_php_web_page.htm&prev=search)

**Debugging** a **PHP** Web Page. This procedure describes how to **debug** whole applications, projects, files or collections of files that are already on the server.

### [PHP Debugger - NuSphere](http://www.nusphere.com/products/php_debugger.htm)

www.nusphere.com/products/php\_debugger.htm

[Dịch trang này](https://translate.google.com.vn/translate?hl=vi&sl=en&u=http://www.nusphere.com/products/php_debugger.htm&prev=search)

PhpED's world famous DBG **Debugger** provides a powerful and easy way to simplify **PHP debugging**because it gives you complete visibility and control over ...

### [Beginner PHP Tutorial - Part 8 - How to Debug PHP - YouTube](https://www.youtube.com/watch?v=uCwO4b2Ceuo)

[**▶ 5:09**](https://www.youtube.com/watch?v=uCwO4b2Ceuo)

https://www.youtube.com/watch?v=uCwO4b2Ceuo

3 thg 12, 2013 - Tải lên bởi Simon Sez IT

Get my free 2+ hour **PHP** programming course. Includes 16 videos to help you learn the essentials of **PHP** ...

### [How to debug PHP application - Support - Cloud9 Community](https://community.c9.io/t/how-to-debug-php-application/1212)

https://community.c9.io/t/how-to-debug-php-application/1212

[Dịch trang này](https://translate.google.com.vn/translate?hl=vi&sl=en&u=https://community.c9.io/t/how-to-debug-php-application/1212&prev=search)

13 thg 1, 2016 - Hi, How do I **debug** a **PHP** app? I press RUN and it loads apache and everything works, but nothing works in the **debugger** itself. Tried to put ...

### [How to Debug PHP with Vim and XDebug on Linux | Box Blog](https://www.box.com/blog/how-to-debug-php-with-vim-and-xdebug-on-linux/)

https://www.box.com/.../how-to-debug-php-with-vim-and-xdebug-on-...

[Dịch trang này](https://translate.google.com.vn/translate?hl=vi&sl=en&u=https://www.box.com/blog/how-to-debug-php-with-vim-and-xdebug-on-linux/&prev=search)

19 thg 6, 2007 - Here is the scenario: You have multiple developers logged into a Linux server which is running Apache and **PHP** using Vim to write **PHP** code.

### [Debugging PHP in Zend Studio](http://files.zend.com/help/Zend-Studio/content/debugging_php_in_zend_studio.htm)

files.zend.com/help/Zend-Studio/.../debugging\_php\_in\_zend\_studio.ht...

[Dịch trang này](https://translate.google.com.vn/translate?hl=vi&sl=en&u=http://files.zend.com/help/Zend-Studio/content/debugging_php_in_zend_studio.htm&prev=search)

This tutorial will teach you how to **debug** files and applications to gain maximum efficiency and accuracy from your **PHP** code. Zend Studio's **Debugging** feature ...

content 1

Nobody enjoys the process of debugging their code. If you want to build killer web apps though, it’s vital that you understand the process thoroughly.

This article breaks down the fundamentals of debugging in PHP, helps you understand PHP’s error messages and introduces you to some useful tools to help make the process a little less painful.

## **Doing your Ground Work**

It is important that you configure PHP correctly and write your code in such a way that it produces meaningful errors at the right time. For example, it is generally good practice to turn on a verbose level of error reporting on your development platform. This probably isn’t such a great idea, however, on your production server(s). In a live environment you neither want to confuse a genuine user or give malicious users too much information about the inner-workings of your site.

So, with that in mind lets talk about the all too common “I’m getting no error message” issue. This is normally caused by a syntax error on a platform where the developer has not done their ground work properly. First, you should turn display\_errors on. This can be done either in your php.ini file or at the head of your code like this:

<?php

ini\_set('display\_errors', 'On');

*Tip: In these code examples I omit the closing (?>) PHP tag. It is* [*generally considered good practice*](http://framework.zend.com/manual/en/coding-standard.php-file-formatting.html#coding-standard.php-file-formatting.general) *to do so in files which contain only PHP code in order to avoid accidental injection of white space and the all too common “headers already sent” error.*

Next, you will need to set an error reporting level. As default PHP 4 and 5 do not show PHP notices which can be important in debugging your code (more on that shortly). Notices are generated by PHP whether they are displayed or not, so deploying code with twenty notices being generated has an impact upon the overhead of your site. So, to ensure notices are displayed, set your error reporting level either in your php.ini or amend your runtime code to look like this:

<?php

ini\_set('display\_errors', 'On');

error\_reporting(E\_ALL);

*Tip: E\_ALL is a constant so don’t make the mistake of enclosing it in quotation marks.*

With PHP 5 it’s also a good idea to turn on the E\_STRICT level of error reporting. E\_STRICT is useful for ensuring you’re coding using the best possible standards. For example E\_STRICT helps by warning you that you’re using a deprecated function. Here’s how to enable it at runtime:

<?php

ini\_set('display\_errors', 'On');

error\_reporting(E\_ALL | E\_STRICT);

It is also worth mentioning that on your development platform it is often a good idea to make these changes in your php.ini file rather than at the runtime. This is because if you experience a syntax error with these options set in your code and not in the php.ini you may, depending on your set up, be presented with a blank page. Likewise, it is worth noting that if you’re setting these values in your code, a conditional statement might be a good idea to avoid these settings accidentally being deployed to a live environment.

## **What Type of Error am I Looking at?**

As with most languages, PHP’s errors may appear somewhat esoteric, but there are in fact only four key types of error that you need to remember:

### **1. Syntax Errors**

Syntactical errors or parse errors are generally caused by a typo in your code. For example a missing semicolon, quotation mark, brace or parentheses. When you encounter a syntax error you will receive an error similar to this:

**Parse error:** syntax error, unexpected T\_ECHO in /Document/Root/example.php on line 6

In this instance it is important that you check the line above the line quoted in the error (in this case line 5) because while PHP has encountered something unexpected on line 6, it is common that it is a typo on the line above causing the error. Here’s an example:

<?php

ini\_set('display\_errors', 'On');

error\_reporting(E\_ALL);

$sSiteName = “Treehouse Blog”

echo $sSiteName;

In this example I have omitted the semi-colon from line 5, however, PHP has reported an error occurred on line 6. Looking one line above you can spot and rectify the problem.

*Tip: In this example I am using* [*Hungarian Notation*](http://en.wikipedia.org/wiki/Hungarian_notation)*. Adopting this coding standard can aid with debugging code while working collaboratively or on a piece of code you wrote some time ago. The leading letter denoting the variable type means that determining a variable type is very quick and simple. This can aid in spotting irregularities which can also help highlight any potential logic errors.*

### **2. Warnings**

Warnings aren’t deal breakers like syntax errors. PHP can cope with a warning, however, it knows that you probably made a mistake somewhere and is notifying you about it. Warnings often appear for the following reasons:

1. Headers already sent. Try checking for white space at the head of your code or in files you’re including.
2. You’re passing an incorrect number of parameters to a function.
3. Incorrect path names when including files.

### **3. Notices**

Notices aren’t going to halt the execution of your code either, but they can be very important in tracking down a pesky bug. Often you’ll find that code that’s working perfectly happily in a production environment starts throwing out notices when you set error\_reporting to E\_ALL.

A common notice you’ll encounter during development is:

>**Notice:** Undefined index: FullName in /Document/Root/views/userdetails.phtml on line 55

This information can be extremely useful in debugging your application. Say you’ve done a simple database query and pulled a row of user data from a table. For presentation in your view you’ve assigned the details to an array called $aUserDetails. However, when you echo $aUserDetails[‘FirstName’] on line 55 there’s no output and PHP throws the notice above. In this instance the notice you receive can really help.

PHP has helpfully told us that the FirstName key is undefined so we know that this isn’t a case of the database record being NULL. However, perhaps we should check our SQL statement to ensure we’ve actually retrieved the user’s first name from the database. In this case the notice has helped us rule out a potential issue which has in turn steered us towards the likely source of our problem. Without the notice our likely first stop would have been the database record, followed by tracing back through our logic to eventually find our omission in the SQL.

### **4. Fatal Errors**

Fatal Errors sound the most painful of the four but are in fact often the easiest to resolve. What it means, in short, is that PHP understands what you’ve asked it to do but can’t carry out the request. Your syntax is correct, you’re speaking its language but PHP doesn’t have what it needs to comply. The most common fatal error is an undefined class or function and the error generated normally points straight to the root of the problem:

**Fatal error:** Call to undefined function create() in /Document/Root/example.php on line 23

### **Using var\_dump() to Aid Your Debugging**

var\_dump() is a native PHP function which displays structured, humanly readable, information about one (or more) expressions. This is particularly useful when dealing with arrays and objects as var\_dump() displays their structure recursively giving you the best possible picture of what’s going on. Here’s an example of how to use var\_dump() in context:

Below I have created an array of scores achieved by users but one value in my array is subtly distinct from the others, var\_dump() can help us discover that distinction.

<?php ini\_set('display\_errors', 'On'); error\_reporting(E\_ALL); $aUserScores = array('Ben' => 7,'Linda' => 4,'Tony' => 5,'Alice' => '9'); echo '<pre>'; var\_dump($aUserScores); echo '</pre>';

*Tip: Wrap var\_dump() in <pre> tags to aid readability.*

The output from var\_dump() will look like this:

array(4) { ["Ben"]=> int(7) ["Linda"]=> int(4) ["Tony"]=> int(5) ["Alice"]=> string(1) "9" }

As you can see var\_dump tells us that $aUserScores is an array with four key/value pairs. Ben, Linda, and Tony all have their values (or scores) stored as integers. However, Alice is showing up as a string of one character in length.

If we return to my code, we can see that I have mistakenly wrapped Alice’s score of 9 in quotation marks causing PHP to interpret it as a string. Now, this mistake won’t have a massively adverse effect, however, it does demonstrate the power of var\_dump() in helping us get better visibility of our arrays and objects.

While this is a very basic example of how var\_dump() functions it can similarly be used to inspect large multi-dimensional arrays or objects. It is particularly useful in discovering if you have the correct data returned from a database query or when exploring a JSON response from say, Twitter:

<?php

ini\_set('display\_errors', 'On');

error\_reporting(E\_ALL);

$sJsonUrl = ‘http://search.twitter.com/trends.json’;

$sJson = file\_get\_contents($sJsonUrl,0,NULL,NULL);

$oTrends = json\_decode($sJson);

echo ‘<pre>’;

var\_dump($oTrends);

echo ‘</pre>’;

## **Useful Tools to Consider when Debugging**

Finally, I want to point out a couple of useful tools that I’ve used to help me in the debugging process. I won’t go into detail about installing and configuring these extensions and add-ons, but I wanted to mention them because they can really make our lives easier.

### **Xdebug**

[Xdebug](http://www.xdebug.org/) is a PHP extension that aims to lend a helping hand in the process of debugging your applications. Xdebug offers features like:

* Automatic stack trace upon error
* Function call logging
* Display features such as enhanced var\_dump() output and code coverage information.

Xdebug is highly configurable, and adaptable to a variety of situations. For example, stack traces (which are extremely useful for monitoring what your application is doing and when) can be configured to four different levels of detail. This means that you can adjust the sensitivity of Xdebug’s output helping you to get granular information about your app’s activity.

Stack traces show you where errors occur, allow you to trace function calls and detail the originating line numbers of these events. All of which is fantastic information for debugging your code.

*Tip: As default Xdebug limits var\_dump() output to three levels of recursion. You may want to change this in your xdebug.ini file by setting the xdebug.var\_display\_max\_depth to equal a number that makes sense for your needs.*

Check out [Xdebug’s installation guide](http://www.xdebug.org/docs/install) to get started.

### **FirePHP**

For all you [FireBug](https://addons.mozilla.org/en-US/firefox/addon/1843) fans out there, [FirePHP](http://www.firephp.org/) is a really useful little PHP library and Firefox add-on that can really help with AJAX development.

Essentially FirePHP enables you to log debug information to the Firebug console using a simple method call like so:

<?php

$sSql = 'SELECT \* FROM tbl';

FB::log('SQL query: ' . $sSql);

In an instance where I’m making an AJAX search request, for example, it might be useful to pass back the SQL string my code is constructing in order that I can ensure my code is behaving correctly. All data logged to the Firebug console is sent via response headers and therefore doesn’t effect the page being rendered by the browser.

*Warning: As with all debug information, this kind of data shouldn’t be for public consumption. The downside of having to add the FirePHP method calls into your PHP is that before you go live you will either have to strip all these calls out or set up an environment based conditional statement which establishes whether or not to include the debug code.*

You can install the Firefox add-on at [FirePHP’s website](http://www.firephp.org/) and also [grab the PHP libs](http://www.firephp.org/HQ/) there too. Oh, and don’t forget if you haven’t already installed FireBug, [you’ll need that too](https://addons.mozilla.org/en-US/firefox/addon/1843).

## **In Conclusion…**

Hopefully during the course of this article you have learned how to do your ground work by preparing PHP for the debugging process; recognise and deal with the four key PHP error types and use var\_dump() to your advantage. Likewise, I hope that you will find Xdebug and FirePHP useful and that they will make your life easier during your development cycle.

As I’ve already mentioned, and I really can’t say this enough, always remember to remove or suppress your debug output when you put your sites into production after all there’s nothing worse than all your users being able to read about your errors in excruciating detail.

Got a great debugging tip to share? Do you use a great little PHP extension that makes your bug trapping life easier? Please tell us about them in comments below!

content 2

# Debugging techniques for PHP programmers

Using print statements, error reporting, and the PHPeclipse plug-in



Tyler Anderson

Published on November 29, 2005

[Facebook](http://www.facebook.com/sharer.php?u=http%3A%2F%2Fwww.ibm.com%2Fdeveloperworks%2Flibrary%2Fos-debug%2Findex.html&t=Debugging%20techniques%20for%20PHP%20programmers)[Twitter](https://twitter.com/intent/tweet?url=http%3A%2F%2Fibm.co%2F2iXNBOm&text=Debugging%20techniques%20for%20PHP%20programmers&via=developerWorks)[Linked In](http://www.linkedin.com/shareArticle?mini=true&url=http%3A%2F%2Fwww.ibm.com%2Fdeveloperworks%2Flibrary%2Fos-debug%2Findex.html&title=Debugging%20techniques%20for%20PHP%20programmers)[Google+](https://plus.google.com/share?url=http%3A%2F%2Fwww.ibm.com%2Fdeveloperworks%2Flibrary%2Fos-debug%2Findex.html&t=Debugging%20techniques%20for%20PHP%20programmers)E-mail this page

Comments

[5](https://www.ibm.com/developerworks/library/os-debug/#icomments)

There are many PHP debugging techniques that can save you countless hours when coding. An effective but basic debugging technique is to simply turn on error reporting. Another slightly more advanced technique involves using print statements, which can help pinpoint more elusive bugs by displaying what is actually going onto the screen. PHPeclipse is an Eclipse plug-in that can highlight common syntax errors and can be used in conjunction with a debugger to set breakpoints.

## Setting up

To learn the concepts described in this article, you are going to need PHP, a Web server, and Eclipse. The latest version of PHP supported by the debugger extension is V5.0.3.

We need a Web server to parse the pages you create in PHP and display them to the browser. This article uses Apache V2. However, any Web server will suffice.

To take advantage of some of the debugging techniques in this article, you need to install Eclipse V3.1.1 and the plug-in: PHPeclipse V1.1.8. Since Eclipse requires Java™ technology, you also need to download that.

You also need the debugger module extension for PHP. Installing it is a bit tricky. Carefully follow the instructions for installing the debugger extension. For now, comment out the lines where you are asked to load and configure the extension in PHP in the php.ini file. We’ll uncomment those lines when we’re ready to use the debugger.

See [Related topics](https://www.ibm.com/developerworks/library/os-debug/#artrelatedtopics) for download information. Now let's move on to error messages.

## Error messages

Error messages are your first line of defense as a developer. You don't want to be developing code in PHP on a server that is not configured to display error messages. However, keep in mind that when your code is debugged and ready to go live, you want to make sure error reporting is turned off because you don't want visitors to your site seeing error messages that may give them enough knowledge to exploit a weakness and hack your site.

You can also use error messages to your advantage because they display the exact line of code that threw or generated an error. This makes debugging a matter of looking at the line number shown on the browser by the generated error and checking that line number in your code. Later, you will see that the PHPeclipse plug-in aides significantly in the development and debugging process by underlining syntax errors on the fly and by marking syntax errors with a red "x" when saving your file.

Let's take a look at how to turn error reporting on in the php.ini file and set the level of error reporting. Then you'll learn how to override these settings in the Apache configuration file.

### Error reporting in PHP

There are many configuration settings in the php.ini file. You should already have set up your php.ini file and placed it in the appropriate directory, as shown in the instructions in the Install PHP and Apache V2 on Linux document (see [Related topics](https://www.ibm.com/developerworks/library/os-debug/#artrelatedtopics)). There are a couple configuration variables you should know about when debugging your PHP applications. Here they are with their default values:

|  |  |
| --- | --- |
| 1  2 | display\_errors = Off  error\_reporting = E\_ALL |

You can discover the current default values of these variables by searching for them in the php.ini file. The purpose of the display\_errors variable is self-evident -- it tells PHP whether or not to display errors. The default value is Off. To make your life easier in the development process, however, set this value to On by replacing Off:

|  |  |
| --- | --- |
| 1 | display\_errors = On |

The error\_reporting variable has a default value of E\_ALL. This displays everything from bad coding practices to harmless notices to errors. E\_ALL is a little too picky for my liking in the development process because it clutters the browser output by displaying notices on the screen for small things like uninitialized variables. I prefer to see the errors, any bad coding practices, but not the harmless notices. Therefore, replace the default value of error\_reporting as follows:

|  |  |
| --- | --- |
| 1 | error\_reporting = E\_ALL & ~E\_NOTICE |

Restart Apache, and you're all set. Next, you'll learn how to do the same thing on Apache.

### Error reporting in the server

Depending on what Apache is doing, turning error reporting on in PHP may not work because you may have multiple PHP versions on your computer. It's sometimes hard to tell which PHP version Apache is pointing to because Apache can only look at one php.ini file. Not knowing which php.ini file Apache is using to configure itself is a security problem. However, there is a way to configure PHP variables in Apache to guarantee the setting of the correct error levels.

Also, it's good to know how to set these configuration variables on the server side to veto or pre-empt the php.ini file, providing a greater level of security.

You should already have toyed with basic configurations in the http.conf file at <apache2-install-dir>/conf/httpd.conf when you configured Apache.

To do the same as you just did in the php.ini file, add the following lines to your httpd.conf to override any and all php.ini files:

|  |  |
| --- | --- |
| 1  2 | php\_flag display\_errors on  php\_value error\_reporting 2039 |

This overrides the flag you have set for display\_errors in the php.ini file, as well as the value of error\_reporting. The value 2039 stands for E\_ALL & ~E\_NOTICE. If you prefer E\_ALL, set the value to 2047, instead. Again, make sure you restart Apache.

Next, we'll test error reporting on your server.

### Testing error reporting

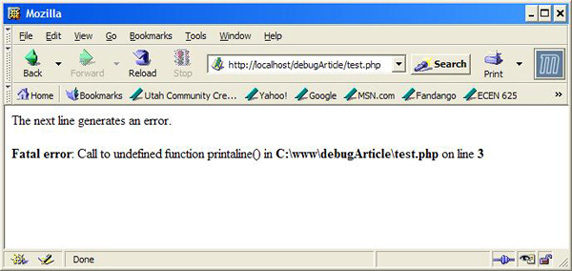
You will save a great deal of time if you leave error reporting enabled. Errors in PHP point you right to the error in your code. Create a simple PHP file, test.php, and define it as shown in Listing 1.

##### Listing 1. A simple PHP that generates an error

|  |  |
| --- | --- |
| 1  2  3  4  5 | <?php  print("The next line generates an error.<br>");  printaline("PLEASE?");  print("This will not be displayed due to the above error.");  ?> |

The first print() statement should display its contents to the Web browser. However, the second statement generates and displays an error to the Web page. This results in the last print() statement do nothing, as shown in Figure 1.

##### Figure 1. Generating an error



You have error reporting turned on! Next, we use print statements to help debug applications.

## Introducing print statements

Because functional bugs in your application don't generate errors, knowledge on how to accurately place and use print or die statements to debug your PHP application can be a great asset in your arsenal of debugging strategies. You can use print statements to narrow down the locations of problem statements in your code that may not be syntactically incorrect or bugs in the code, but they are bugs in the functionality of your code. These are the hardest bugs to find and debug because they throw no errors. You only know that what is being displayed to the browser isn't what you intended, or that what you thought was being stored in your database isn't being stored at all.

Suppose you are processing form data sent in via a GET request and want to display the information to the browser, but for whatever reason, the data is either not being submitted properly, or it isn't being read from the GET request properly. To debug such a problem, it's important to know what the value of the variable is, using a print() or a die() statement.

The die() statement halts program execution and displays text to the Web browser. The die() statement is particularly useful if you don't want to have to comment out your code, and you only want everything up to the error and the error displayed and nothing after.

Let's test this concept of using print statements in PHP.

### Debugging using print statements

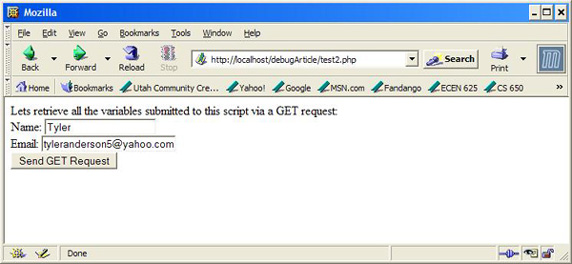
In my years as a programmer where I developed applications on Linux® boxes with no handy GUI to tell me where my bugs were, I quickly caught on that the more print statements I laced my program with the more chances I had of narrowing down the bugs in my application to a single line. Create another PHP file, test2.php, and define it as shown in Listing 2.

##### Listing 2. Display all variables submitted via GET

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <?php  $j = "";  print("Lets retrieve all the variables submitted to this ");  print("script via a GET request:<br>");  foreach($\_GET as $key => $i){  print("$key=$j<br>");  }  if($\_GET['Submit'] == "Send GET Request")  $j = "done!<br>";  ?>  <form method="GET">  Name: <input name="name"><br>  Email: <input name="email" size="25"><br>  <input name="Submit" type="submit" value="Send GET Request">  </form> |

You may be able to spot the bug in Listing 2 very easily. Good for you! Note that this is a simple script, and is shown as an example in using print statements to debug. The script simply takes all variables in the GETrequest, if any, and displays them back to the browser. A form is also supplied for you to send variables to the server using a GET request for testing. See the output, as shown in Figure 2.

##### Figure 2. Output of test2.php



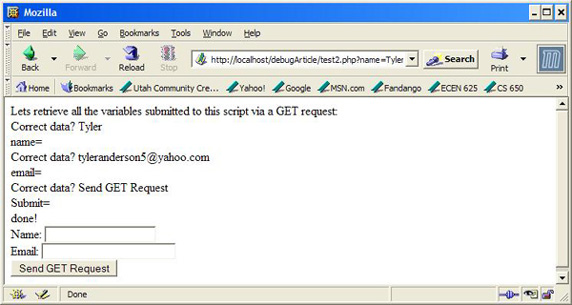
Now click the Send GET Request button. Notice that only the keys of the $\_GET request got displayed to the browser, and the correct values did not. You can place a print statement within the loop to verify that data indeed exists at each element in the foreach loop. See Listing 3.

##### Listing 3. Using a print statement to verify code functionality

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | ...  foreach($\_GET as $key => $i){  print("Correct data? " . $\_GET[$key] . "<br>");  print("$key=$j<br>");  }  ... |

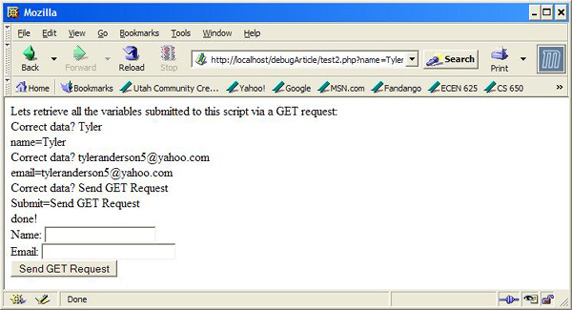
The placed print statement is in bold font. Notice that you already know that the displayed $key values on the Web browser are correct, but for some reason, the values aren't being displayed correctly. See the new output, as shown in Figure 3.

##### Figure 3. Output of modified test2.php



Now you know that your application is receiving the variables in the GET request correctly, so there must be a bug in your code. You look over and notice that the variable you are using for displaying the values, $j, is the wrong one. You specified $i in the foreach statement, which must have the correct values, but you accidentally typed in $j. You quickly fix the problem by replacing $j with $i and see the correct output by reloading the page, as shown in Figure 4.

##### Figure 4. Output of fixed test2.php



You can now remove or comment out the print statement you added because you have discovered the bug in your code. Notice that this is a small subset of the many errors you might experience while debugging your application. A good solution to a problem you may encounter when working with a database is to print out your SQL statements to make sure that what you are executing SQL you intended to execute.

Now you'll take a look at using the Eclipse IDE and the PHPeclipse plug-in and debugger extension to further aid you in your debugging ventures.

## Using PHPeclipse

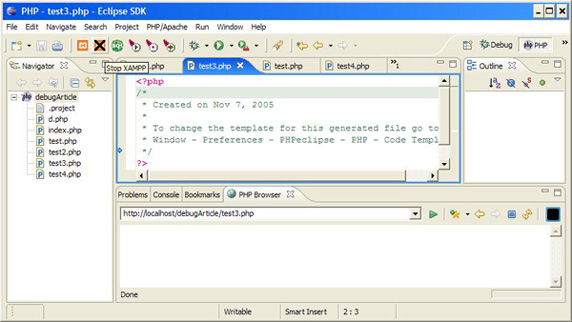
You have probably used Eclipse, but you aren't familiar with it. See [Related topics](https://www.ibm.com/developerworks/library/os-debug/#artrelatedtopics) for introductions to the Eclipse platform.

The PHPeclipse plug-in for Eclipse is a popular tool used for developing PHP applications. Start Eclipse and specify your workspace directory as the www directory for Apache (c:\www on my machine). Now click onFile > New > Project. The New Project wizard will pop up. Double-click on the PHP folder and select PHP Project. Click Next, enter a project name, debugArticle, and click Finish.

If you set up your Web server to listen to port 80, you don't need to change anything. Otherwise, go to the Navigator window and right-click on your PHP project, debugArticle. Select Properties, then click PHP Project Settings. Click Configure Workspace Settings and change localhost appropriately or add the port your Web server is listening on (http://localhost:8080, for example). Click Apply and you're set.

The Navigator Window should display your project and a single .project file. Right-click on your project as you did before, except this time select New > PHP File. Replace \*.php with the name of the PHP file you want to create, test3.php, and click Finish. A new file should appear in the Eclipse IDE. You may have to navigate the bottom window to the PHP Browser to view the current output of your PHP file (see Figure 5).

##### Figure 5. The PHPeclipse plug-in for Eclipse



Note that only Windows® users can use the PHP browser as shown in Listing 5. The exact same functionality can be used by opening up a separate browser window and pointing your browser to the directory where your test scripts are located.

Now let's demo this application and prove its awesome capabilities.

In the Using the debugger section, you will learn how to debug your PHP application using Eclipse, PHPeclipse and the debugger PHP extension downloaded earlier. You'll start by learning how to use its syntax parsing abilities.

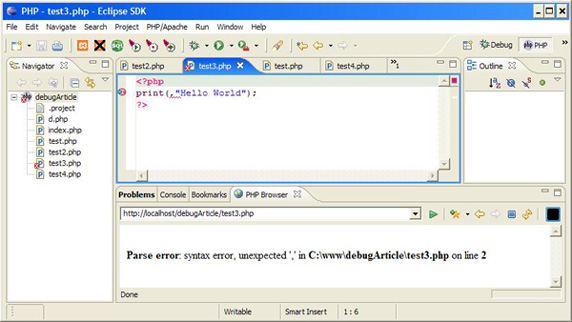
### Syntax parsing and underlining

Let's start by seeing how PHPeclipse provides you with real-time syntax parsing abilities to assist you with debugging PHP applications. To see this feature in action, start by defining the test3.php file in Eclipse, as shown below.

|  |  |
| --- | --- |
| 1  2  3 | <?php  print(,"Hello World!");  ?> |

Notice that two characters underlined in Listing 4 get underlined in Eclipse, notifying you of incorrect syntax. Saving the file by pressing Ctrl+S displays the Parse error in Eclipse by placing a red "x" by the line that matches the parse error in your code, as shown in Figure 6.

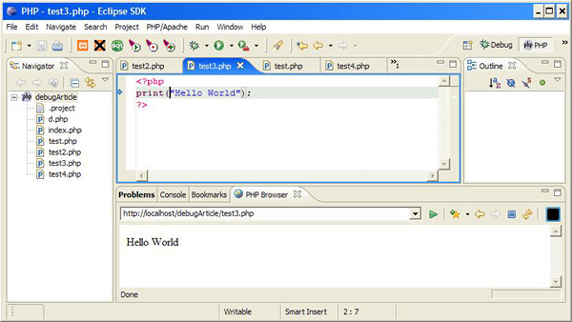
##### Figure 6. Syntax error highlighting



Now let's demo the PHP browser. This window gives you a preview of the current PHP script, as shown in Figure 6.

Remove the comma (,) from test3.php, defined above. Save the file by pressing Ctrl+S, and watch the PHP browser window update by displaying Hello World (see Figure 7).

##### Figure 7. Previewing PHP scripts in PHPeclipse



Next up is setting breakpoints in PHP using the debugger.

## Using the debugger

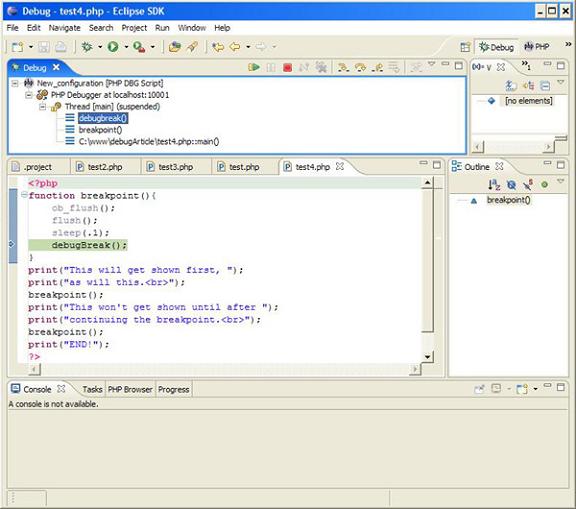
Using the debugger, you can set breakpoints and see the browser output from your PHP code up to the breakpoint you set. You can then resume code execution and see the rest of the browser output up to the next breakpoint and the next until your PHP script has completed.

Uncomment the lines you commented out in the php.ini file in the Setting up section and restart Apache. The debugger is loaded, and Eclipse will be able to hook into it.

Now let's set up the debug environment in Eclipse. Create a new test4.php file and leave it empty for now. Now click Run > Debug. Select PHP DBG Script on the left-side panel and click New. Now go to the Filetab, and type in your current project, debugArticle, and the file you want to debug, test4.php. Now go to the Environment tab, then to the Interpreter subtab. Browse for your php.exe file in your PHP install directory (mine is c:\apps\php5.0.3\php.exe). Now click on the Remote Debug subtab, select Remote Debug, and if you're not using Windows, uncheck the "Open with DBGSession URL in internal Browser box." Set the Remote Source path equal to the absolute path (not the Web path) to the PHP script you're going to test (I have mine set to c:\www\debugArticle\test4.php). Now click Debug.

The Debug perspective should load, as shown in Figure 8. Otherwise, click Window > Open Perspective > Other, and select Debug.

##### Figure 8. Debug perspective in Eclipse



Now you're ready to set breakpoints.

With the versions of the plug-in and extension used in this article, a breakpoint function is required because PHP buffers output before sending it to the browser. Besides that, you need to do more than set a breakpoint to flush the current display data to your Web browser, so define your test4.php file, as shown below and in Figure 8 above.

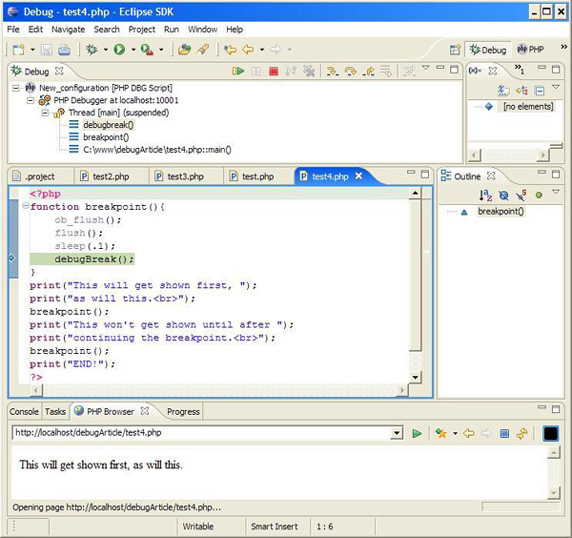
##### Listing 4. Setting and creating breakpoints

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <?php  function break-point(){  ob\_flush();  flush();  sleep(.1);  debugBreak();  }  print("This will get shown first, ");  print("as will this<br>");  breakpoint();  print("This won't get shown until after ");  print("continuing the break-point<br>");  breakpoint();  print("END!");  ? |

The breakpoint() function flushes output buffered and any other buffered data to the Web browser. The call to sleep(.1) is necessary so the server has enough time to flush the data out to the Web browser before code execution is halted at debugBreak(), a function known internally to the PHP debugger extension you downloaded earlier. Thus, calling breakpoint() flushes data from HTML blocks, print()and echo() statements to the browser, then halts code execution.

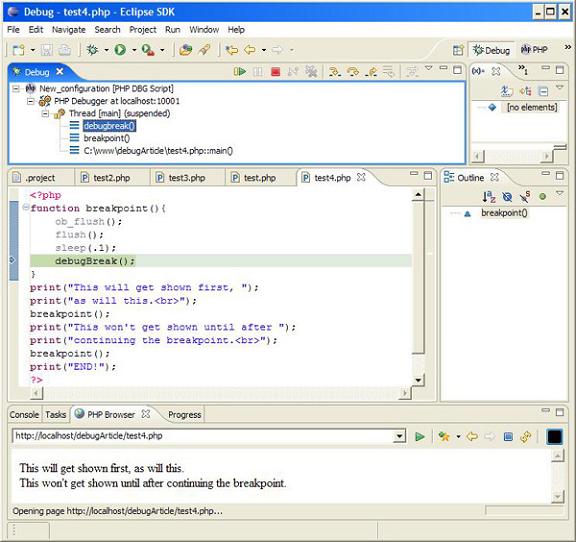
After you have written the code shown in Listing 4, you can open a browser and point to test4.php or you can look at the PHP browser window (http://localhost/debugArticle/test4.php for me). Each time you type and save the file, the debugging sequence is already initiated in the PHP browser window. If you are not using Windows, look at test4.php through your browser. After saving your file, resume code execution withF8 or by clicking Run > Resume. Do this until you see END! as the last line of output (see figures 9, 10 and 11).

##### Figure 9. Initial PHP browser output up to first breakpoint



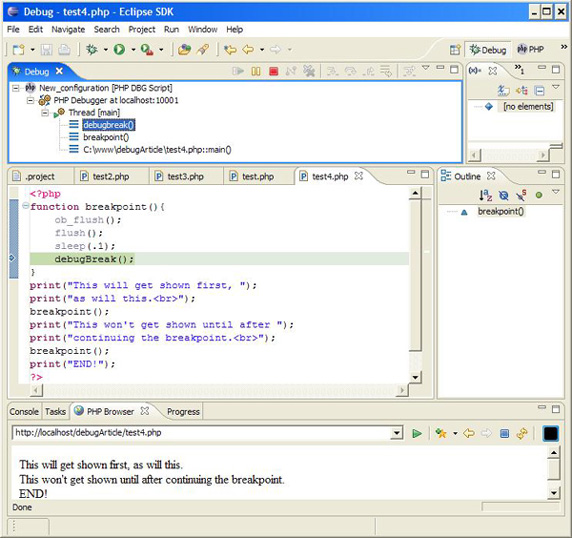
Notice how the Debug window in Figure 9 shows the execution output as suspended.

##### Figure 10. PHP browser output after first breakpoint and before second breakpoint



The Debug window in Figure 10 still shows execution as suspended and the second set of data is shown in the PHP browser.

##### Figure 11. Full PHP browser output



Notice that the code is no longer suspended in the Debug window in Figure 11, and the entire script has executed, as shown in the PHP browser of Figure 11.

Since you have witnessed the advantages of developing with PHPeclipse and the debugger extension, you'll wonder how you ever got along without it.

## Summary

Now that you have added the use of error reporting, print statements, PHPeclipse, and the debugger extension to your arsenal of debugging techniques in PHP, you will become a more effective PHP coder by reducing the number of errors you create per line of code. See [Related topics](https://www.ibm.com/developerworks/library/os-debug/#artrelatedtopics) for some PHP tutorials on which to test your new skills.

content 3

# Debugging PHP Source Code in the NetBeans IDE

**Contents**



* [Getting Ready](https://netbeans.org/kb/docs/php/debugging.html#gettingReady)
* [How PHP Debugging with XDebug Works in NetBeans IDE](https://netbeans.org/kb/docs/php/debugging.html#howDebuggerWorks)
* [Debugging Options](https://netbeans.org/kb/docs/php/debugging.html#options)
* [Working with the Toolbar and Editor](https://netbeans.org/kb/docs/php/debugging.html#work)
* [Debugger Windows](https://netbeans.org/kb/docs/php/debugging.html#editorLayout)
* [Debugging Session](https://netbeans.org/kb/docs/php/debugging.html#debuggingSession)
* [Sample Debugging Session](https://netbeans.org/kb/docs/php/debugging.html#sampleDebuggingSession)
* [Using Additional Watches](https://netbeans.org/kb/docs/php/debugging.html#usingAdditionalWatches)
* [Mixed PHP and HTML Cases](https://netbeans.org/kb/docs/php/debugging.html#mixedPHPHTMLCases)
* [Path Mapping, Debugger Proxy, and Starting a Debug Session on a Custom URL](https://netbeans.org/kb/docs/php/debugging.html#debug_url)

**To follow this tutorial, you need the following software and resources.**

|  |  |
| --- | --- |
| **Software or Resource** | **Version Required** |
| [NetBeans IDE](https://netbeans.org/downloads/index.html) | PHP download bundle |
| A PHP engine | Version 5 |
| A web server | [Apache HTTP Server 2.2](http://httpd.apache.org/download.cgi) is recommended. |
| A PHP debugger | [XDebug 2.0 or later](http://www.xdebug.org/) |

## Getting Ready

To successfully debug PHP applications in the NetBeans IDE for PHP, you need to have the PHP engine, the Apache local web server, and the XDebug debugger [installed and configured for PHP development](https://netbeans.org/kb/trails/php.html#configuration). If you have difficulties getting XDebug to work, see the [NetBeans wiki on XDebug](http://wiki.netbeans.org/HowToConfigureXDebug) and/or ask the community at users@php.netbeans.org.

## How PHP Debugging with XDebug Works in NetBeans IDE

When you run XDebug from NetBeans IDE, PHP program execution pauses at every line where you set a breakpoint. When the program execution is paused, XDebug can retrieve information about the current program state, such as the values of the program variables. Practically, this means the following workflow:

1. Set a breakpoint at each line where PHP source code execution should pause.
2. Start a debugging session.
3. When a line with a breakpoint is reached, execute the script one line after another by pressing F7 and F8. Monitor the state of the application in the [debugger windows](https://netbeans.org/kb/docs/php/debugging.html#editorLayout).
4. Close the debugging session.

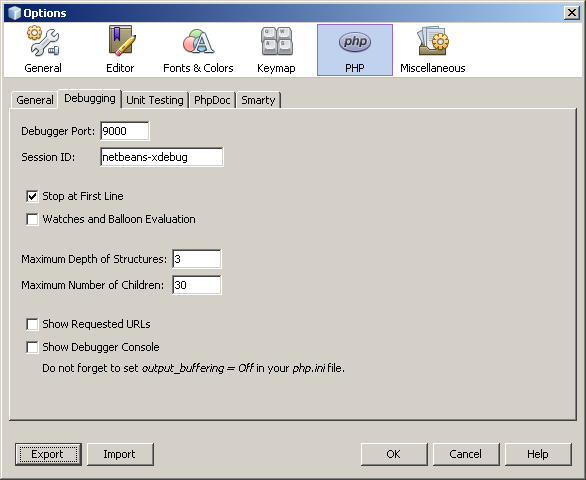
For a detailed workflow of using XDebug with NetBeans IDE, see [Debugging Session](https://netbeans.org/kb/docs/php/debugging.html#debuggingSession).

NetBeans IDE provides a debugging toolbar that helps you step through your files. See [Working with the Toolbar and Editor](https://netbeans.org/kb/docs/php/debugging.html#work).

## Debugging Options

The NetBeans IDE Options include a tab for changing certain default settings for debugging PHP. To open these options, go to Tools > Options (NetBeans > Preferences on Mac), select the PHP options, and select the Debugging tab.

**Note:** The Debugging tab was introduced in NetBeans IDE version 7.1. Earlier versions of NetBeans have debugging options in the General PHP tab. Not all options in version 7.1 are available in earlier versions.



You can change the following options in this panel:

* **Debugger port** This is the port that XDebug uses, as set in php.ini. By default it is port 9000. *The port number in this dialog must be the same as the debugger port you set in php.ini. You cannot affect the port that XDebug uses in this dialog. You only inform NetBeans IDE of the port that XDebug uses.*
* **Session ID** Arbitrary name of the debugging session. By default it is netbeans-xdebug. This value is important to note in cases where you have to set the xdebug.idekey property in php.ini, such as some remote debugging scenarios.
* **Stop at First Line** Tick this option for the debugger session to stop at the first line of code instead of at the first breakpoint. This keeps your screen focused in the IDE instead of switching to a browser window when you launch a debugging session.
* **Watches and Balloon Evaluation** Watches and balloon evaluation is disabled by default. Watches and ballon evaluation destabilizes XDebug.
* **Maximum Depth of Structures.** Sets the visibility of nested structures (such as nested arrays, objects in objects, etc.)
* **Maximum Number of Children.** Sets the visibility of array items during Watch evaluation. (If you set Maximum Number of Children to 1 you will see only the first item of an array even if the array has more then one item.)
* **Show Requested URLs.** Opens new Output window during debugging. This Output window is named PHP Requested Urls and it shows currently processed URLs. The URLs are clickable. Click on a URL in the Output window to open the URL in a browser window.
* **PHP Debugger Console.** Opens new Output window that shows the output of debugged scripts.
* **Note:** Set output\_buffering = Off in your php.ini file. Otherwise there will be a delay in seeing script output in the Output window.

For the purposes of this tutorial you do not need to change any of these settings, except optionally to enable watches.

## Working with the Toolbar and Editor

The editor enables you to view the contents of files. So when you are debugging, the editor, in combination with the debugger toolbar, provides you with the ability to step through your code during execution in order to see how the contents of your files affect what is happening in a browser.

* [Using the Debugger Toolbar](https://netbeans.org/kb/docs/php/debugging.html#toolbar)
* [Setting Breakpoints](https://netbeans.org/kb/docs/php/debugging.html#editorBreakpoints)
* [Examining Tooltips](https://netbeans.org/kb/docs/php/debugging.html#editorTooltips)

### Using the Debugger Toolbar

When you run a debugging session, the debugger toolbar displays above the editor.

Debugger toolbar in suspended state

The toolbar provides you with the following actions:

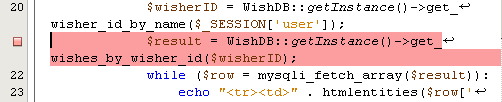
|  |  |
| --- | --- |
| **Finish Session** ( Finish Debugging Session button ) | Finish debugging session |
| **Pause** ( Pause button ) | Suspend debugging session |
| **Resume** ( Resume button ) | Resume debugging session |
| **Step Over** ( Step Over button ) | Step over an execution statement |
| **Step Into** ( Step Into button ) | Step into a function call |
| **Step Out** ( Step Out button ) | Step out of the current function call |
| **Run to Cursor** ( Run to Cursor button ) | Run execution to the position of the cursor |

### Setting Breakpoints

Set breakpoints in your files to tell the debugger where to stop during execution.

**Important:** You *must* set breakpoints in your PHP code to use XDebug.

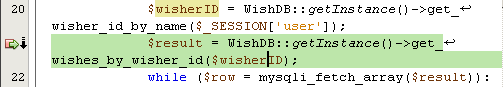
To set a breakpoint, click in the left margin of the editor on the line where you want the breakpoint to be set.



You can remove the breakpoint by clicking on the breakpoint badge ( Breakpoint badge ).

It is also possible to temporarily disable breakpoints. To do so, right-click on a breakpoint badge and unselect Breakpoint > ✔Enabled. This toggles the breakpoint into a disabled state, resulting in a grey badge ( Breakpoint badge ) displaying in the left margin.

If the debugger encounters a breakpoint upon executing, it stops on the breakpoint, enabling you to examine variables in the debugging windows, and step through any code following the breakpoint.



### Examining Tooltips

When the debugger is suspended during a debugging session, you can hover your mouse over a PHP identifier in the editor to display a tooltip. If the identifier is valid in the selected call stack frame, its value is displayed. You can also select PHP expressions. The value of the expression is shown in a tooltip.

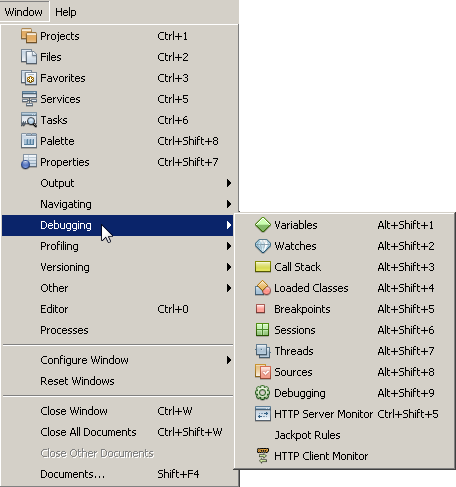
Tool tip displayed in editor

## Debugger Windows

When you start a debugging session, a set of Debugger windows opens below the main editor window. The debugger windows enable you to keep track of variable and expression values as you step through code, examine the call stack of an executing thread, verify source URL's, and switch between sessions if you are running concurrent debugging sessions.

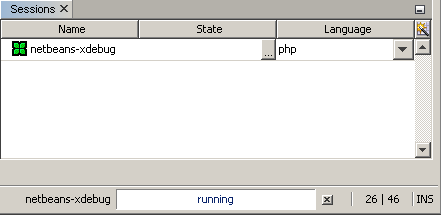
* [Sessions Window](https://netbeans.org/kb/docs/php/debugging.html#sessions)
* [Variables Window](https://netbeans.org/kb/docs/php/debugging.html#localVar)
* [Watches Window](https://netbeans.org/kb/docs/php/debugging.html#watches)
* [Call Stack Window](https://netbeans.org/kb/docs/php/debugging.html#callStack)
* [Threads Window](https://netbeans.org/kb/docs/php/debugging.html#threads)
* [Sources Window](https://netbeans.org/kb/docs/php/debugging.html#sources)
* [Breakpoints Window](https://netbeans.org/kb/docs/php/debugging.html#breakpoints)

All debugger windows can be accessed from the IDE's Window > Debugging menu. Once a debugging session is active, you can begin making use of the debugger windows.

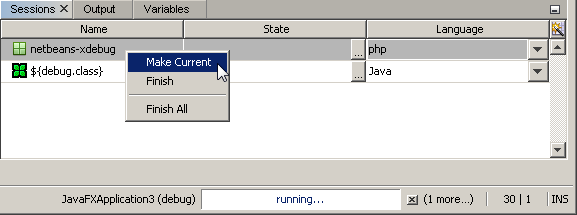


### Sessions Window

The Sessions window shows any debugging sessions that are currently active. When you start a PHP debugging session, you can see an entry for the PHP debugger in the Sessions window.



The NetBeans IDE also enables you to run multiple debugger sessions simultaneously. For example, you can debug a Java and a PHP project at the same time. In this case, you can identify two sessions listed in the Sessions window.



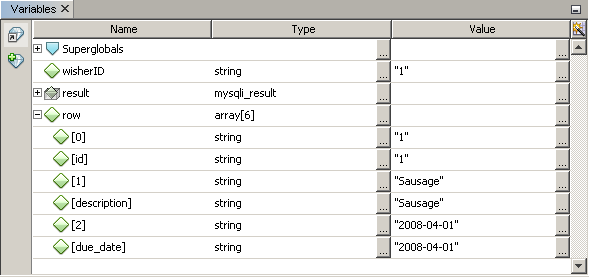
The current session (i.e., the session that you can control using the debugger toolbar) is indicated by the more prominent icon ( Current session icon ). To switch sessions, you can double-click on the session you want to make current, or right-click a non-current session and choose Make Current.

**Note:** It is not recommended to switch sessions if the session you are currently working in is suspended.

You can also use the right-click pop-up window to terminate a session (right-click and choose Finish), or toggle between debugging the current thread or all threads within the session (right-click and choose Scope > Debug All Threads or Debug Current Thread).

### Variables Window

When the debugger is suspended, the Variables window displays the variables of the current window object for the selected callstack frame. A node is displayed for every variable in the current window. Superglobals are grouped under a separate node.



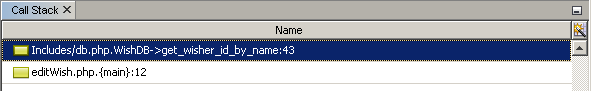
As you step through the code, the values of some local variables may change. Such local variables are shown in bold in the Local variables window. You can also click directly into the Value column and manually change variable values.

### Watches Window

Setting watches destabilizes XDebug and is not recommended. Watches are disabled by default. However if you want to set watches, see [Using Additional Watches](https://netbeans.org/kb/docs/php/debugging.html#usingAdditionalWatches).

### Call Stack Window

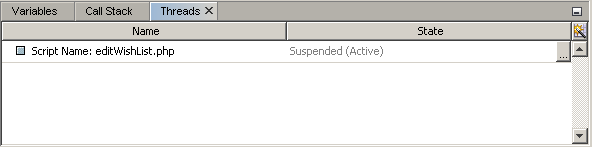
The Call Stack window lists the sequence of calls made during execution. When the debugger is suspended, the Call Stack window shows the sequence of function calls (i.e., the *call stack*). On initial suspension, the top-most stack frame is selected automatically. Double-click on a function call in the window to go to that line in the editor. If the call is made to a PHP class, the Navigator window will also go to that line when you double-click the call.



You can double-click on a call stack frame to select it, then explore any variable or expression values for that frame in the [Variables](https://netbeans.org/kb/docs/php/debugging.html#localVar) and [Watches](https://netbeans.org/kb/docs/php/debugging.html#watches) windows.

### Threads Window

The Threads window indicates which PHP script is currently active and whether it is suspended at a breakpoint or running. If the script is running, you need to go to the browser window and interact with the script.

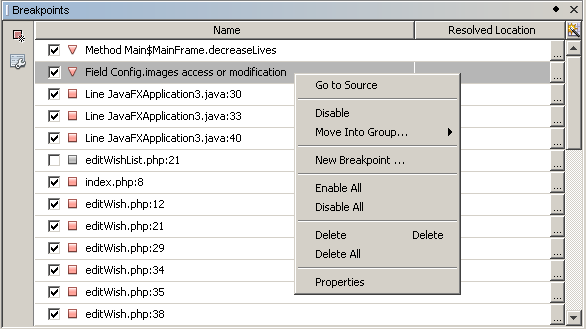


### Sources Window

The Sources window displays all the files and scripts loaded for the debugging session. The Sources window does not currently function for PHP projects.

### Breakpoints Window

You can use the Breakpoints window to view all breakpoints that you have set in the IDE.



From the Breakpoints window you can enable or disable breakpoints in the Context window. You can also create breakpoint groups.

## Debugging Session

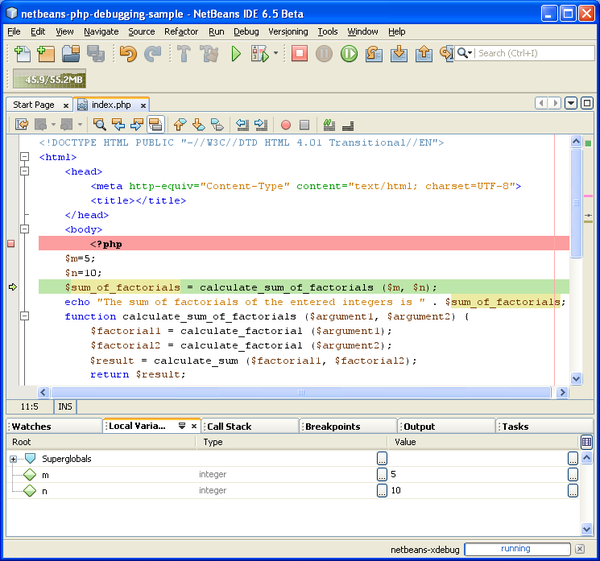
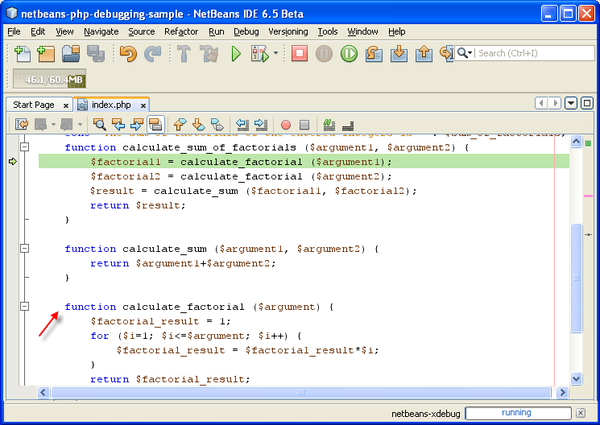
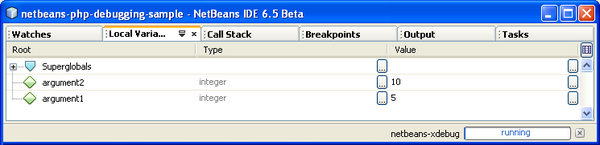
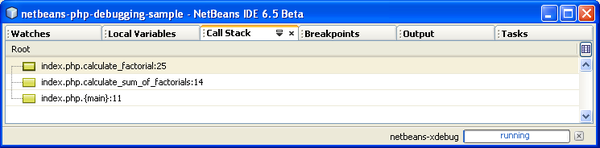
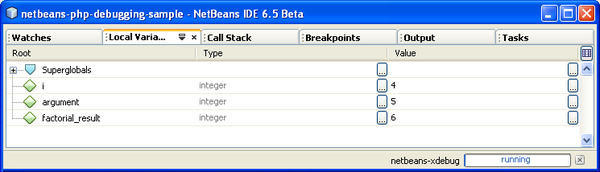
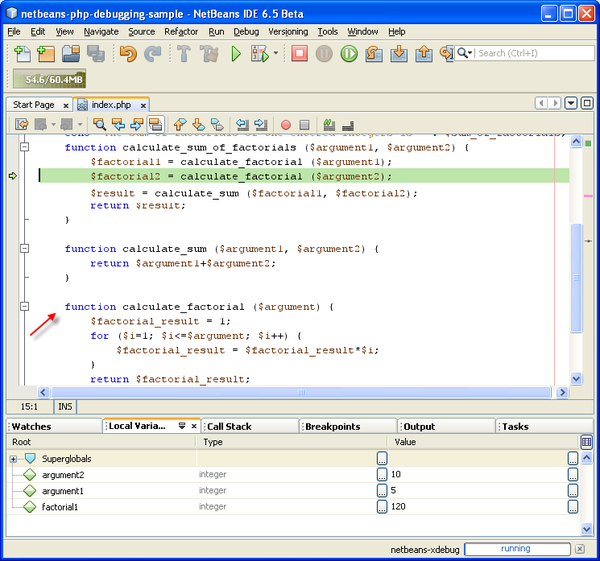
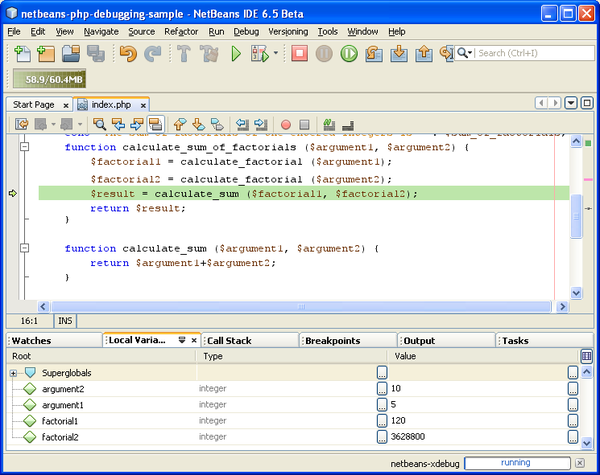
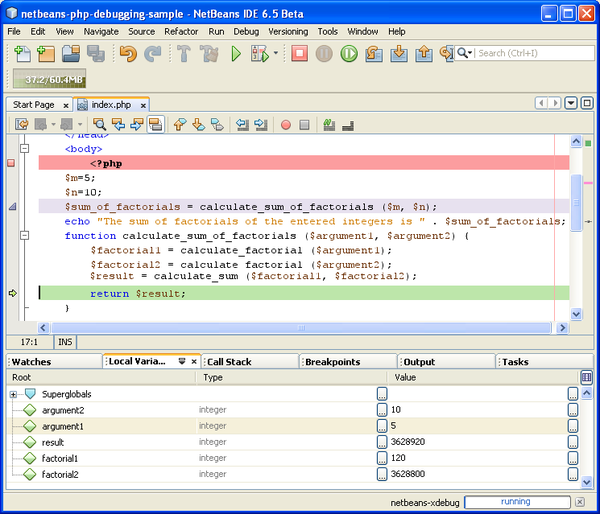
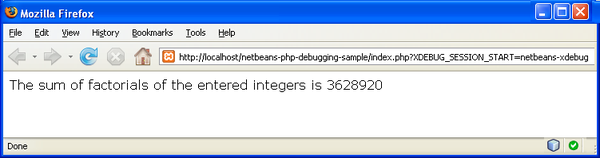
The following procedure is the workflow of a typical debugging session.

**To run a debugging session:**

1. Start the ide and open the file that contains the source code that you want to debug.
2. Set a breakpoint at each line where you want the debugger to pause. To set a breakpoint, place the cursor at the beginning of a line and press Ctrl-F8/⌘-F8 or choose Debug > Toggle Line Breakpoint.
3. In the Projects window, navigate to the current project node, click the right mouse button, and choose Debug from the popup menu. The IDE opens the Debugger windows and runs the project in the debugger until the breakpoint is reached.
4. **Note:** If the current project is set as Main, you can choose Debug > Debug Main Project, or press Ctrl-F5, or click Debug main project button.
5. Switch to the Local Variables window. The window shows all the variables that have been initialized within the current function, their types, and values.
6. To view the value of a variable outside the function, bring the cursor on an occurrence of the variable. The tooltip shows the variable value.
7. To execute the program line by line including the lines within all the called functions, press F7 or choose Debug > Step Into and watch the changes to the variables' values in the Local Variables window.
8. To check the logic of the program by watching the changes of expressions, define a new watch:
   1. To open the Watches window, choose Window > Debugging > Watches or press Ctrl-Shift-2. The Watches window opens.
   2. Anywhere inside the Watches window, click the right mouse button and choose New Watch from the popup menu. The New Watch window opens.
   3. Enter the watch expression and click OK.
9. Now you can make an additional check during debugging.
10. **Important:** You must enable watches in the [Debugging tab of PHP Options](https://netbeans.org/kb/docs/php/debugging.html#options) in order to set watches.
11. To cancel the line by line execution of the code within a function and spring to the next line after the function call, press Ctrl-F7/⌘-F7 or choose Debug > Step Out.
12. To skip the line by line execution of the code in a function, get the value returned by the function, and spring to the next line after the function call, press F8 or choose Debug > Step Over.
13. To pause the debugging session, choose Debug > Pause.
14. To continue the debugging session, choose Debug > Continue or press Continue debugging session button.
15. To cancel the debugging session, press Cancel debugging session button.
16. When the program reaches the end, the debugger windows close.

## Sample Debugging Session

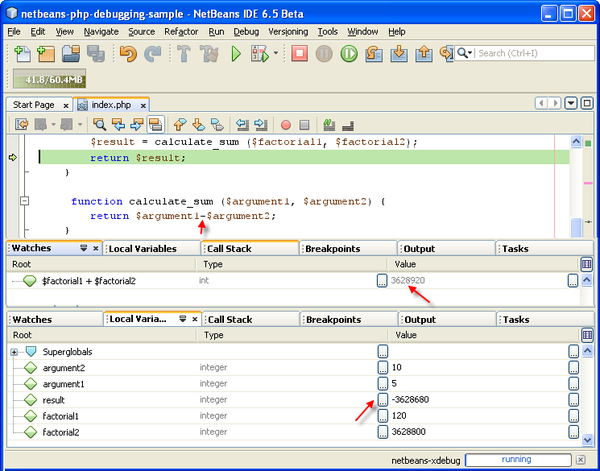
The sample in this section illustrates the basic debugger functions, including stepping into and over functions. It also shows typical debugger window output.

1. Create a new PHP project with the following parameters:
   * Project type - PHP application
   * Sources location - the default htdocs folder location
   * Run configuration - Local Web Site
2. Find more details about setting up a PHP project in the document [Setting Up a PHP Project](https://netbeans.org/kb/docs/php/project-setup.html).
3. To enable the use of hotkeys during the session, position the cursor on the project node and choose Set as Main Project from the popup menu.
4. In the index.php file, enter the following code:
5. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
6. <html>
7. <head>
8. <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
9. <title>NetBeans PHP debugging sample</title>
10. </head>
11. <body>
12. <?php
13. $m=5;
14. $n=10;
15. $sum\_of\_factorials = calculate\_sum\_of\_factorials ($m, $n);
16. echo "The sum of factorials of the entered integers is " . $sum\_of\_factorials;
18. function calculate\_sum\_of\_factorials ($argument1, $argument2) {
19. $factorial1 = calculate\_factorial ($argument1);
20. $factorial2 = calculate\_factorial ($argument2);
21. $result = calculate\_sum ($factorial1, $factorial2);
22. return $result;
23. }
25. function calculate\_factorial ($argument) {
26. $factorial\_result = 1;
27. for ($i=1; $i<=$argument; $i++) {
28. $factorial\_result = $factorial\_result\*$i;
29. }
30. return $factorial\_result;
31. }
33. function calculate\_sum ($argument1, $argument2) {
34. return $argument1 + $argument2;
35. }
36. ?>
37. </body>
38. </html>
39. The code contains three functions:
    * The calculate\_factorial () function
    * The calcualte\_sum () function
    * The calculate\_sum\_of\_factorials () function that calls the calculate\_factorial function twice, then calls the calcualte\_sum () function once, and returns the calculated sum of factorials.
40. Set a breakpoint (Ctrl-F8/⌘-F8) at the beginning of the PHP block:
41. <?php
42. To start debugging, click Debug main project button. The debugger stops at the breakpoint.
43. Press F7 three times. The debugger stops at the line where the function calculate\_sum\_of\_factorials () is called. The Local Variables window shows the variables $m and $n with their values:
44. 
45. To step into the function calculate\_sum\_of\_factorials(), press F7. The debugger starts to execute the code within the function calculate\_sum\_of\_factorials () and stops at the call of the function calculate\_factorial().
46. 
47. The Local Variables window now shows the local variables $argument1 and $argument2 that are declared within the function calculate\_sum\_of\_factorials ().
48. 
49. Press F7. The debugger starts to execute the code with the function calculate\_factorial(). The Call Stack window shows the stack of calls to the functions in the reverse order with the last called function at the top of the list:
50. 
51. Press F7 to step into the loop. View the values of the variables in the Variables window.
52. 
53. When you make sure that the code works correctly, press Ctrl-F7/⌘-F7 to cancel the function execution. The program returns to the line next after the call of the function calculate\_factorial().
54. **Note:** Alternatively, you can press F7 until the program completes execution of the function calculate\_factorial(). You will also return to the line next after its call.
55. 
56. Because you have just checked the function calculate\_factorial() and know that it works correctly, you can skip its execution ("step over"). To step over, press F8. The program stops at the call of the function calculate\_sum().
57. 
58. To step into the function calculate\_sum(), press F7.
59. To step over, press F8. In either case the debugger stops at the last line in the function calculate\_sum\_of\_factorials().
60. 
61. Press F7. The debugger moves to the line with the echo statement.
62. Press F7 till the debugger exits the program. The browser window opens and shows the result of program execution:
63. 

## Using Additional Watches

You can define an additional watch expression to follow up your program execution. This may help you catch an error.

**Warning:** Setting additional watches destabilizes XDebug. Watches are disabled by default in [Debugging Options](https://netbeans.org/kb/docs/php/debugging.html#options).

1. Update the code as follows (replace a plus with a minus):
2. function calculate\_sum ($argument1, $argument2) {
3. return $argument1 - argument2;
4. }
5. Suppose that this resulted from misspelling but actually you need to calculate sum.
6. Choose Debug > New Watch or press Ctrl/⌘-shift-F7. The New Watch window opens.
7. Enter the following expression and click OK.
8. $factorial1+$factorial2
9. The new expression appears in the Watches window.
10. Run the debugging session. When the debugger stops at the line
11. return $result;
12. compare the value of the expression in the Watches window and the value of $result in the Local Variables window. They should be the same but they differ.
13. 

This example is very simple and is supposed to give some general notion of using watches.

## Mixed PHP and HTML Cases

You can debug code that contains both PHP and HTML blocks. In the example from the [Sample Debugging Session](https://netbeans.org/kb/docs/php/debugging.html#sampleDebuggingSession) section the values are hardcoded. Expand the code with an HTML input form for entering values.

1. Add the following HTML code above the <?php ?> block:
2. <form action="index.php" method="POST">
3. Enter the first integer, please:
4. <input type="text" name="first\_integer"/><br/>
5. Enter the second integer, please:
6. <input type="text" name="second\_integer"/><br/>
7. <input type="submit" name="enter" value="Enter"/>
8. </form>
9. Find more information about [HTML input forms](https://netbeans.org/kb/docs/php/wish-list-lesson2.html#htmlForm).
10. Replace the following lines at the top of the <?php ?> block:
11. $m=5;
12. $n=10;
13. $sum\_of\_factorials = calculate\_sum\_of\_factorials ($m, $n);
14. echo "The sum of factorials of the entered integers is " . $sum\_of\_factorials;
15. with the following code:
16. if (array\_key\_exists ("first\_integer", $\_POST) && array\_key\_exists ("second\_integer", $\_POST)) {
17. $result = calculate\_sum\_of\_factorials ($\_POST["first\_integer"], $\_POST["second\_integer"]);
18. echo "Sum of factorials is " . $result;
19. }
20. Set a breakpoint at the beginning of the <?php ?> block and start the [debugging session](https://netbeans.org/kb/docs/php/debugging.html#debuggingSession).
21. Press F7. The debugger steps into the program. The browser window opens but the input form is not displayed. This is correct behavior of the debugger because it has to pass through the entire source code of a web page before the page can be displayed. Practically, this means that the debugger passes through the code twice. First time the debugger processes the code to display the HTML input form. The second time the debugger executes PHP code step by step.
22. Press F7 till the debugger reaches the end of the program and the input form opens.
23. Fill in the form and click Enter. The debugging session continues as described in section [Sample Debugging Session](https://netbeans.org/kb/docs/php/debugging.html#sampleDebuggingSession).

## Path Mapping, Debugger Proxy, and Starting a Debug Session on a Custom URL

It is possible to debug both scripts and web pages, and web pages can be debugged either locally or remotely. For Remote Debugging, unfortunately the debugged php file on the remote server isn't the same as the file opened in NetBeans IDE running on a local machine. Debugger support in NetBeans must thus be able to map server paths to local paths. However, due to many complications, path mapping cannot be resolved automatically for every individual scenario. Therefore, starting in NetBeans 6.7 you can manually define path mapping through the [project setup](https://netbeans.org/kb/docs/php/project-setup.html) for individual run configurations. You can also specify the proxy server, if any, and the URL at which the debugging session starts. If you do not specify this URL, debugging starts at your index file.

**To set up path mapping and enable custom debugging URLs:**

1. Right-click the project's node in the Projects window and open the project's Properties from the context menu.
2. In the Project Properties dialog, go to the Run Configuration category.
3. Click the Advanced button. The Advanced Web Configuration dialog opens.
4. Add the server path and the project path for path mapping.
5. Under "Debug URL", select one of the following (do not leave the default selected):
   * Ask Every Time, which has the IDE prompt you for the URL when you start a debugging session.
   * Do Not Open Web Browser, which requires you to open the browser and enter the URL manually (you need the GET/POST XDEBUG\_SESSION\_START variable).
6. If you are using a proxy server for debugging, enter the server's host name and port in the Debugger Proxy section.

For more information, please see the [Path Mapping in PHP Debugger](http://blogs.oracle.com/netbeansphp/entry/path_mapping_in_php_debugger) post in the Net Beans for PHP blog.

nhận xét

kết quả tìm kiếm ra đầu tiên là 1 tang web nước ngoài

với h1 “debug in php” có debug php trùng với keyword đã search

và trong đó content là những contribute note(code và kinh nghiệm chia sẽ)