# XDebug Support In PDT 1.0

Version: 2.01
Dave Kelsey

Latest Revision: 2 October 2007

*This document provides an overview of the features and idiosyncrasies of XDebug support in PDT (PHP Developer Toolkit)*

# 1 Contents

# 2 XDebug Support User Guide

The following versions of XDebug are currently supported

- Official 2.0.0 release

XDebug is available from PECL or http://www.xdebug.org, prebuilt windows binaries are available and you need to select the appropriate version for the level of PHP you are running. For Linux, you will need to download the source and build it yourself. The instructions for this are on http://www.xdebug.org .

This website also provides loads of information about setting up xdebug and issues with xdebug itself, please visit the website for more information.

## 2.1 Determining the correct INI file to update

For EXE launches, PDT will take a copy of the INI file stored in the same directory as the PHP executable, create a copy and will use that copy. Make sure that you add your Xdebug information to this INI file for launches.

For Web launches, you need to modify the INI file that is used by the version of PHP that is executed by your web browser. You can determine this file by invoking a simple script with the following contents

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

and look at the output, specifically the line

```
Configuration File (php.ini) Path => C:\WINDOWS\php.ini
```

Which indicates the ini file being used.

## 2.2 Required configuration of XDebug

The following minimal configuration is required for XDebug in your PHP.INI file if you are using the thread safe, non debug version of PHP (This is the default build for the windows binary version on PHP.net).

```
[xdebug]
xdebug.remote_enable=1
xdebug.remote_host=<hostname>
xdebug.remote_port=<port>
xdebug.remote_handler="dbgp"
zend_extension_ts=<xdebug library location>
```

Where

- <hostname> is the name of the host where your IDE will be running
- <port> is the port you have configured your IDE to listen on (9000 is the default)

An example set of entries may look like

```
[xdebug]
```

```
xdebug.remote_enable=1
xdebug.remote_host="localhost"
xdebug.remote_port=9000
xdebug.remote_handler="dbgp"
zend_extension_ts="C:\php\php_xdebug-2.0.0-5.2.2.dll"
```

Ensure you place your entries at the bottom of your ini file or XDebug could fail to load.

You may need to change the “zend_extension_ts” to “zend_extension” if you are using the non thread safe version of PHP or to “zend_extension_debug” if you are using the debug version.

To determine if Xdebug has been located successfully, you can either

- launch PHP with the –m option to list the loaded modules
- launch PHP with the –i option to output definition information
- run a PHP script which calls the “phpinfo()” function.

With the –m option you should get something like

```
[PHP Modules]
bcmath
calendar
...
...

[Zend Modules]
Xdebug
```

With the –i option and phpinfo you should be looking for the following entries

```
This program makes use of the Zend Scripting Language Engine:
Zend Engine v2.1.0, Copyright (c) 1998-2006 Zend Technologies
        with Xdebug v2.0.0, Copyright (c) 2002, 2003, 2004, 2005, 2006, 2007, by Der
ick Rethans
```

As well as information about the XDebug extension and its configuration settings.

```
xdebug

xdebug support => enabled
Version => 2.0.0

Supported protocols => Revision
DBGp - Common DeBuGger Protocol => $Revision: 1.125 $
GDB - GNU Debugger protocol => $Revision: 1.87 $
PHP3 - PHP 3 Debugger protocol => $Revision: 1.22 $
…
…
```

If you don't get this and you are sure the path is correct then you need to make sure you have the correct entry for zend_extension in your PHP.INI file. When you do PHP –i or run a script with phpinfo() in it you need to look for 2 entries

```
Debug Build => no
Thread Safety => enabled
```

The above output shows a non debug build that has thread safety so you should use "zend_extension_ts".

If thread safety was "disabled" then you should have the php.ini entry of

```
zend_extension=<xdebug library location>
```

if it was a debug build you need to have the entry

```
zend_extension_debug=<xdebug library location>
```

If you are running a thread safe debug version of PHP, then your ini entry must be of the form

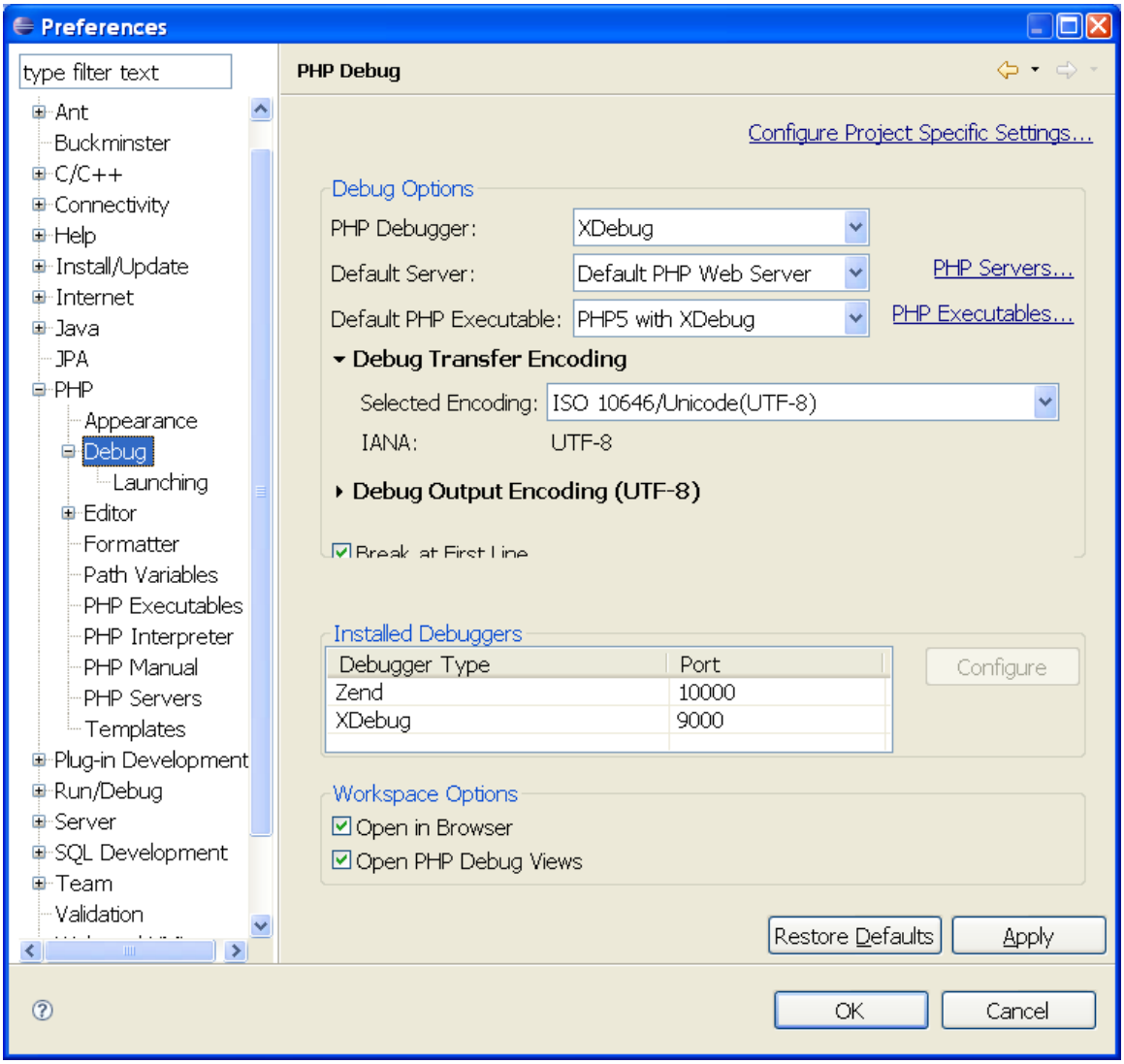
```
zend_extension_debug_ts=<xdebug library location>
```

In summary the rule is (in the stated order)

1. start with "zend_extension"
2. if you have a debug build: `Debug Build => yes` , add "_debug"
3. if you have thread safety: `Thread Safety => enabled` , add "_ts"

## 2.3 Configuring XDebug support in PDT

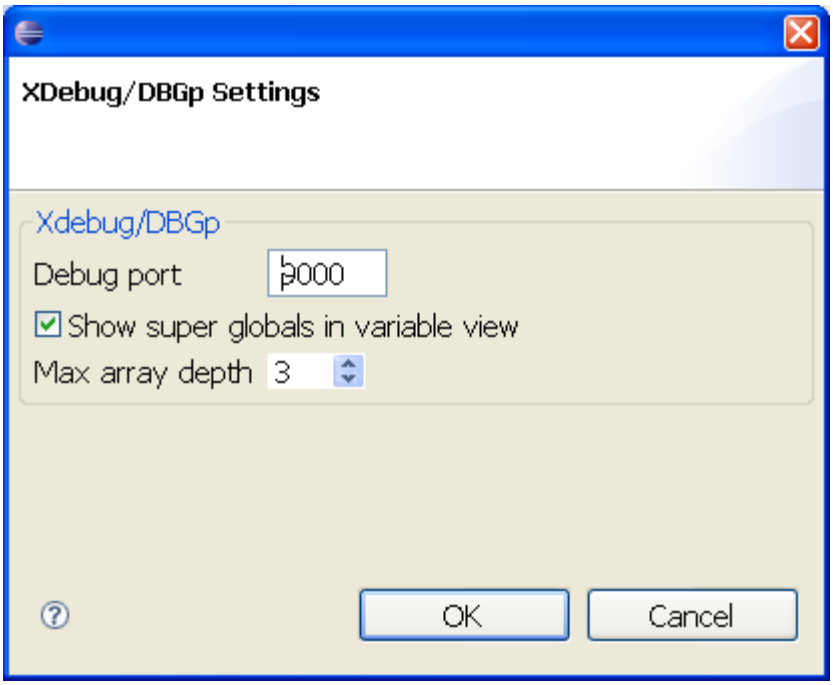
PDT has a preference page for debug which allows you to configure various aspects of the debug environment. These options are also available on a project specific basis. An example of this dialog is shown here.




Currently the XDebug support doesn't make use of "Debug Transfer Encoding" or "Debug Output Encoding". You should change the "PHP Debugger" option to XDebug to ensure that XDebug is selected as the default debugger.

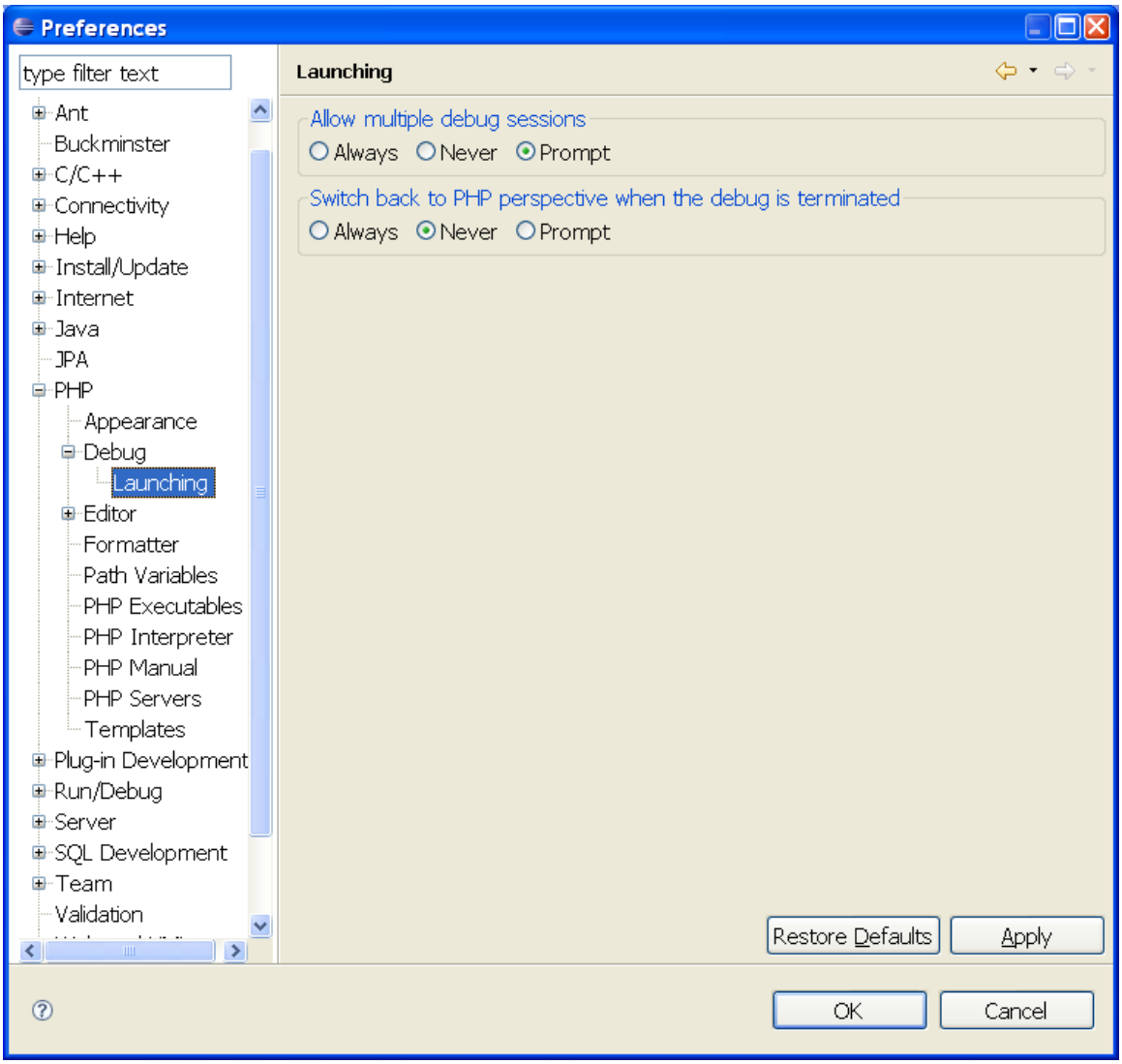
Port 9000 is the default for XDebug, but it will depend on how you have configured XDebug in PHP.INI

There are some specific XDebug options as well, to change these select "XDebug" in the "Installed Debuggers" panel and press the Configure button.




| Item | Description |
|---|---|
| Debug port | Corresponds to the value you specified for “xdebug.remote_port” in PHP.INI |
| Show super globals in variable view | Display the super globals in the variable view when debugging |
| Max array depth | Defines how much data is retrieved on s ingle request for nested arrays. This doesn’t restrict the depth that can be displayed, just how much information is retrieved per request to XDebug. |

In the “Launching” subsection of the debug preferences, XDebug doesn’t support the “Allow multiple debug sessions” options.




### 2.3.1 Testing your setup.

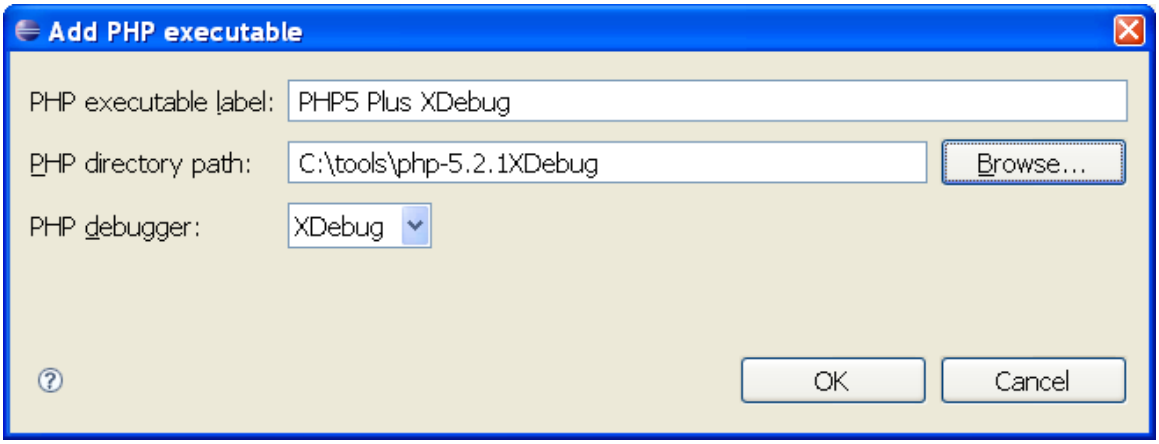
It is highly recommended that you do a debug EXE and/or Web launch of a script containing

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

and review the output to see if xdebug is loaded correctly as described in section 2.2.

## 2.4 Debugging using XDebug

If you plan on debugging standalone scripts, you will need to define a PHP Executable location. In preferences, select PHP Executables and press the Add… button to get the following dialog




Ensure you have selected the PHP debugger as XDebug and define the path to where you have PHP with XDebug setup.

Debugging a PHP script or Web Page works in the same way no matter which debugger you have selected. When using the pop-up menu “Debug As” ensure you have set XDebug as the PHP Debugger either in preferences on specifically on the project to ensure XDebug is used.

### 2.4.1 Web Launch

The simplest way to debug web PHP scripts is to have a local web server on your machine and set up it up so that its Document Root points to your workspace or a project under your workspace in PDT as the location of your PHP Scripts. For example if your workspace was at

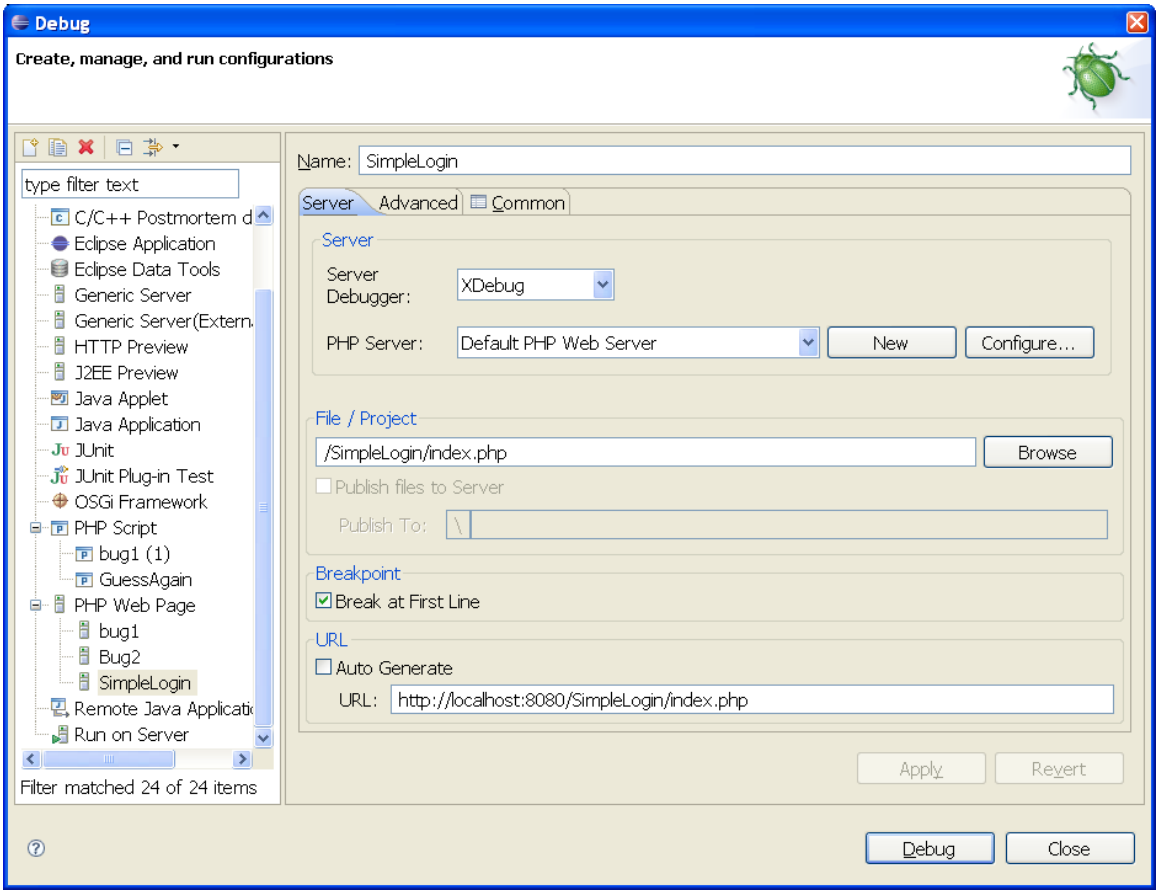
C:\development\Phase1

Then you could start Apache as follows

```
Apache.exe -c "DocumentRoot "C:\development\Phase1""
```

Note the quotes and ensure you don't have any unnecessary white space.

Defining your document root at the workspace level will mean you cannot use the facility to automatically generate the URL, you will need to enter the project where the initial script is, in the example below the URL has had the text "SimpleLogin" added by hand.




If you define your document root at the project level, for example

C:\development\Phase1\SimpleLogin

Then you can use the Auto Generate facility. The URL path will be correct.

In this scenario, you can choose to untick the “Auto Generate” URL to define it yourself, and the contents of “File / Project” have no effect on debugging (ie the contents can be any php file or php project).

#### 2.4.1.1 Path Mapping

If you cannot setup a webserver on your local machine and have to debug a remote server, then that is still possible to do this, but this requires path mapping. The concept of path mapping is to try to match the script executing on the server with the script stored in your PDT workspace. The scripts in the PDT workspace must be identical to those stored on the server and also the directory structure must also be the same. XDebug needs to know the fully qualified name of the file to place the breakpoints on, but the breakpoints are placed on the files in PDT which are not the same files so a remapping must occur.

In order to work out this mapping, the “File / Project” must contain the file in the PDT workspace that is a copy of the initial script that will be executed when the launch URL is invoked.

In the above example, the URL “http://localhost:8080/SimpleLogin/index.php” invokes the script c:\htdocs\SimpleLogin\index.php, the workspace is located at d:\pdt-workspace, so the information in “File / Project” refers to file d:\pdt-workspace\SimpleLogin\index.php

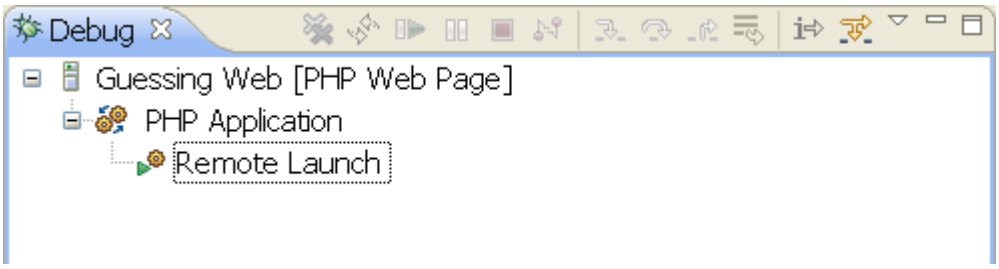
#### 2.4.1.2 Debugging the web script

The application is driven through a web browser. The type of browser and whether it is internal (ie a view inside of eclipse) is defined under the Window→ Preferences in the “Web Browser” entry under “General” Section.

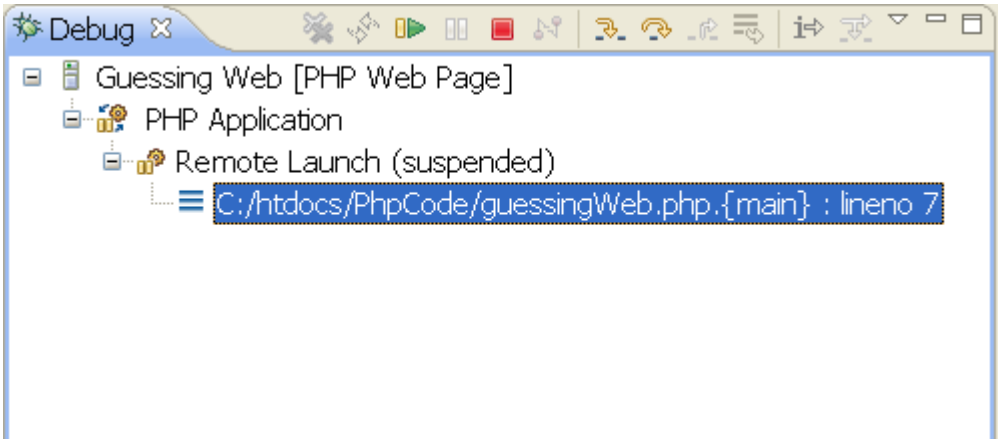
If you have problems trying to get the internal view to work, then you can change to using an external browser.

As your application is driven through a web browser and XDebug works by saving a persistent cookie to your web browser to ensure that multiple requests and redirections/links to other scripts still continue to be debugged, you can only ever have a single Web launch active at any one time.

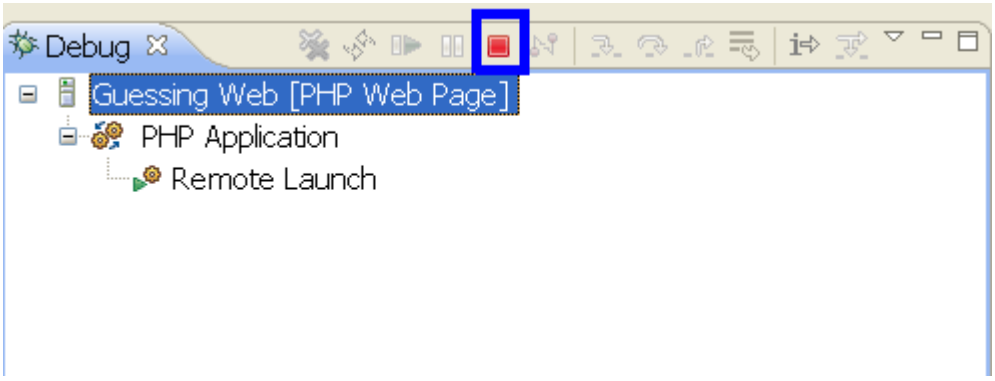
When a debug session is waiting for the next request to be processed and be debugged your Debug window will look like this



When debugging a script you should see something like this in the debug window

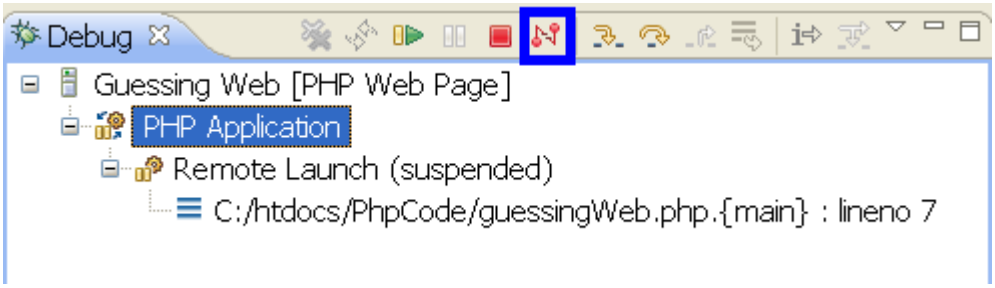



Whenever a request completes, the Web launch debug will remain active waiting for the next request to be debugged until you explicitly stop the debug session by terminating it, ie by selecting either the PHP Web script launch, the “Remote Launch” child entry, or the “PHP Thread” child entry and press the red button.



#### 2.4.1.3 Disconnect

You can disconnect a web launch at any time by selecting either “PHP Application” or the higher level parent which is the name of the launch configuration itself (In the example below it is “Guessing Web”).




When you disconnect from a web launch, the web launch remains active but stops the script that is currently being executed by your web server. Your browser will still have the XDebug cookie registered which means if you go to that browser press enter or return back to the original launch URL, you should still be in debug mode and eclipse will receive a new debug session initiation.

This is different to terminate which will stop the script from running, but also send the stop URL to remove the XDebug cookie from the browser (and thus stop xdebug from debugging further from that browser) and it also terminates the web launch.

## 2.5 Breakpoint support

Breakpoints work in a similar manner to other IDEs and languages. It is best to add breakpoints before you run your script or web application or add new break points when your script is suspended due to a break point or you are stepping through code.

Break points added to a file while a script is not running will not be noticed until your script suspends due to an already existing break point. In other words if you are blocked in your script (for example a blocking function call such as fgets) or you are in a long loop, any break points you add will not be honoured. So you cannot for example attempt to stop a script from running a long loop by putting a breakpoint on a line within that loop.

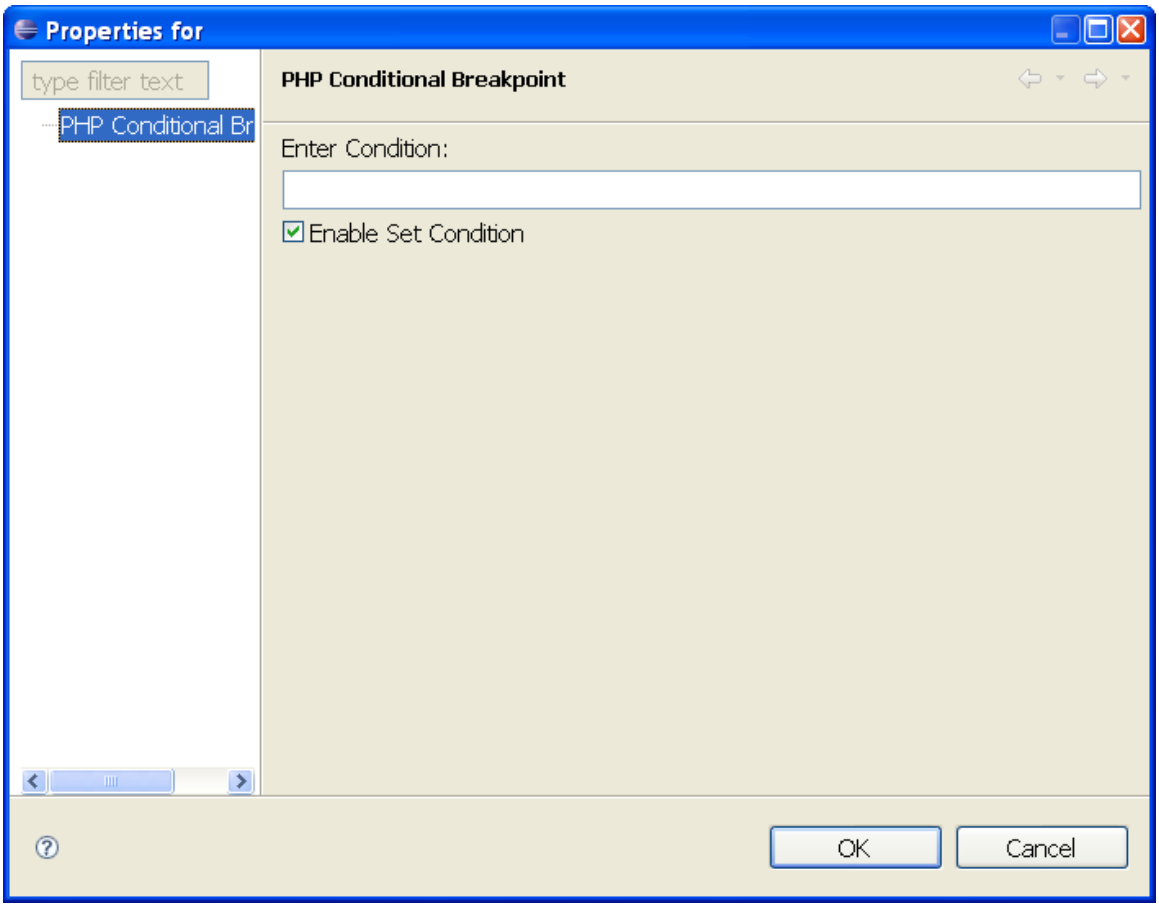
Any break point added to a file while a script is actively running (in the case of a Web launch this will be while the request is running) and not suspended, are deferred until the script is suspended. If the script ends or the request ends then the breakpoints will be installed before the script is run or another request is processed.

### 2.5.1 Conditional Breakpoint support

There are 2 types of conditional breakpoints supported

- Hit counts
- Expression evaluation

You set a condition of a breakpoint by bringing up the popup menu for an existing breakpoint and selecting “Breakpoint properties…”. This will bring up a dialog box




Where you can set the condition

#### 2.5.1.1 Hit counts

You can control when a breakpoint will suspend a script based on the number of times the line where the breakpoint is set is executed. There are 3 types of hit count you can specify

| expression | Behaviour |
|---|---|
| hit(== x) | Suspend when you execute this line for the ‘x’th time only |
| hit(>=x) | Suspend when you execute this line after but including the ‘x’th time |
| hit(%x) | Suspend when you execute this line every ‘x’th time |

So for example if you want a script to suspend every 10th time a specific line is executed, you would enter into the condition box

```
hit(%10)
```

#### 2.5.1.2 Expression evaluation

This allows you to control when a breakpoint will suspend a script when an expression evaluates to true. So for example when a counter variable exceeds a given number, you may wish to have the script suspend. An example of the expression you would enter into the condition box is

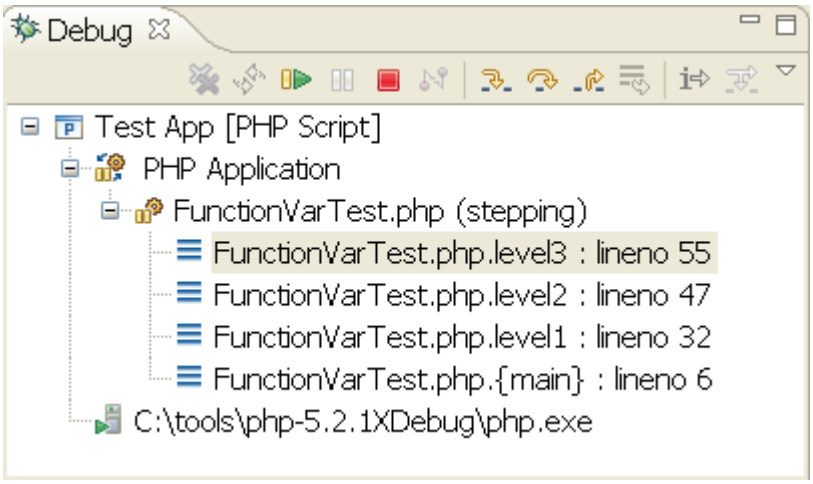
```
$counter >= 20
```

## 2.6 The debug views

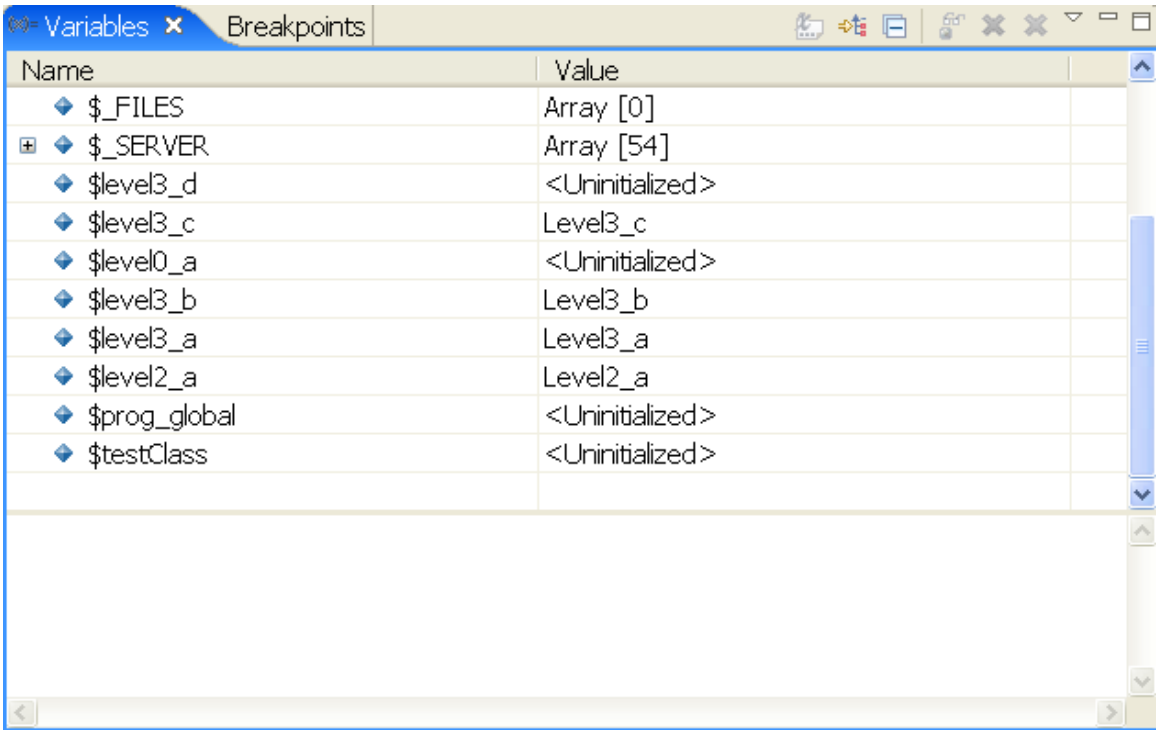
Apart from the source file, the other important views while debugging are the Stack frames and the variables view. Only when the script or request is suspended will these views display anything.

Variables are only visible at a certain stack level so when you select a different stack level of your suspended script you will see the list of variables change. Some variables will be common such as globals and super globals (if you have chosen to have super globals be shown in the variables view)

The following shows executing scripts, which include their stack frames

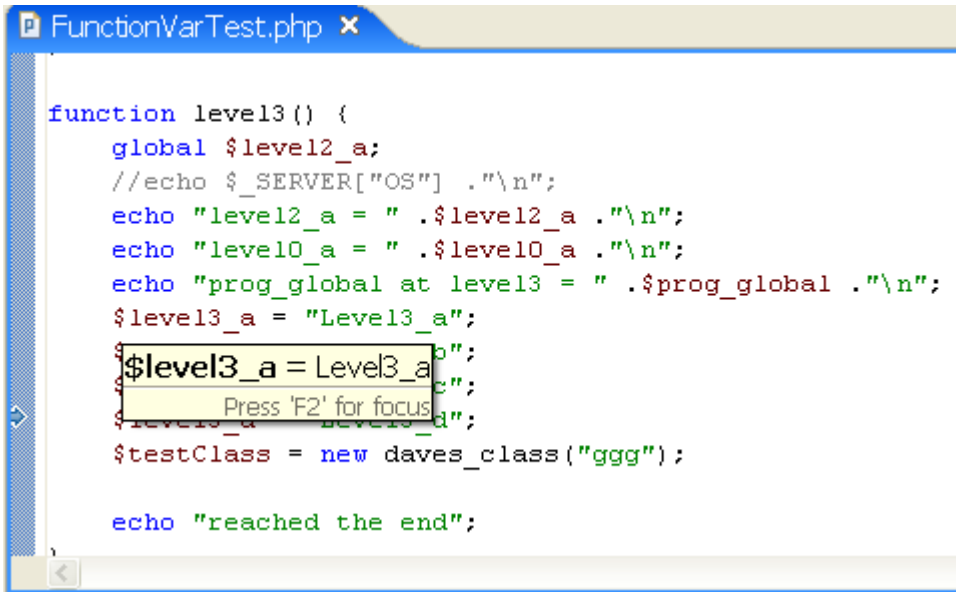



The list of variables and their values exposed for the selected stack frame



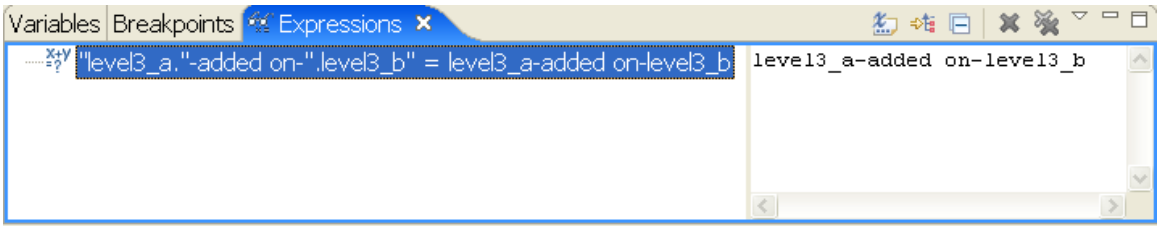
### 2.6.1 Hover

While suspended you can select an entry in the stack frame and it will show you the file and highlight the line it is at (so long as the file is part of your workspace). You can place your mouse over variables and if they are within scope the contents of that variable will be highlighted




### 2.6.2 Expression view

You can also create expressions in the expression window which will be evaluated and displayed whenever your script or request is suspended.




One quick way to add a variable to the expressions window is through the watch pop up menu item. To do this highlight a variable (include the $), you can do this by double clicking the variable, bring up the pop up menu and select “watch”

## 2.7 Known issues

- Currently it is not possible to change the type of a variable when you change its contents. For example you cannot change a boolean to a string or an integer/float to a string. You can change a string to a number and you can interchange between integers and floats
- Linked files will not be found by the debugger
- Do not attempt to enter watch expressions that invoke methods. There seems to be a bug in xdebug 2.0.0 that causes the script to terminate.

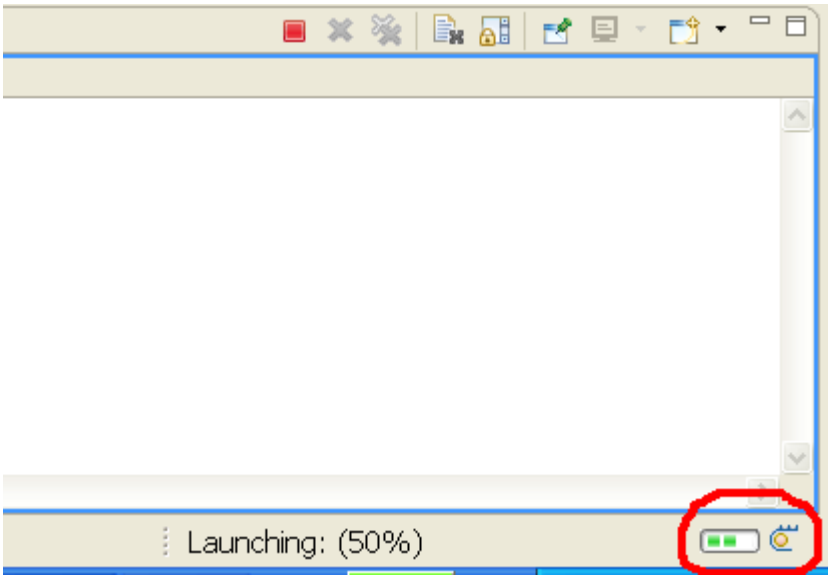
## 2.8 Tips

### 2.8.1 Launch waiting for debug session

You have done a launch but you find that you aren't debugging and the launch status window shows that the application is still launching. You can still interact with the script but you cannot debug. This can occur when

- An PHP script launch or PHP Web Page launch is done but you either have no XDebug information defined in your INI file or the XDebug information doesn't contain the correct Server, Port or debug protocol.
- An PHP script launch produces a firewall pop-up which you deny
- A PHP Web Page launch but either the web server is not running or the defined URL in the launch is incorrect

You will see the following at the bottom right of your IDE. You can click on the very right icon to bring up the launch view for more information




You can terminate the debug session in the usual ways, as well as terminating the launch from the launch view.

END-OF-DOCUMENT