XDebug Support In PDT 2.0

Version: 1.0
Dave Kelsey

Latest Revision: 31 December 2008

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 – 2.0.3 releases

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 Getting debug information.

XDebug can generate a log which is useful for debugging. To enable logging you need to add the following line to php.ini

```
xdebug.remote_log=<file location>
```

2.4 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.
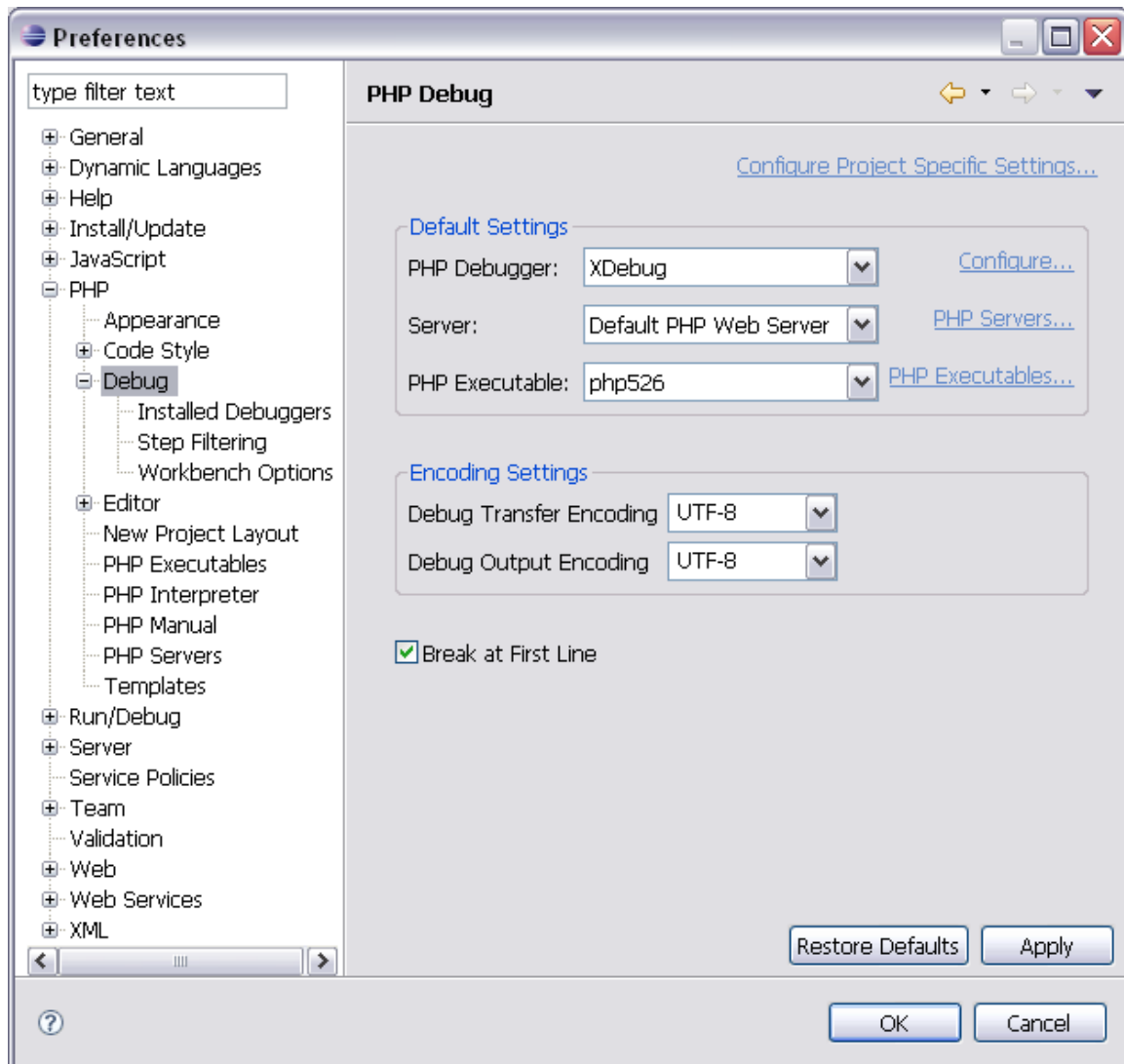
You should change the "PHP Debugger" option to XDebug to ensure that XDebug is selected as the default debugger.

2.4.1 Transfer Encoding

Variable data is transferred to and from PDT in bytes and the bytes have character meaning. This defines the code page of these character bytes so that they can be correctly displayed in PDT when converted to Unicode strings and correctly converted to appropriate bytes when updating a variable in PHP. Most of the time UTF-8 will be the correct encoding to use.

2.4.2 Output encoding

Output from the PHP is in bytes. In order to display these bytes in the Browser Output view and the Debug Output view, the data must be converted to Unicode strings. Output encoding defines the character meaning of the output in PHP so that the data can be displayed in these PDT views.
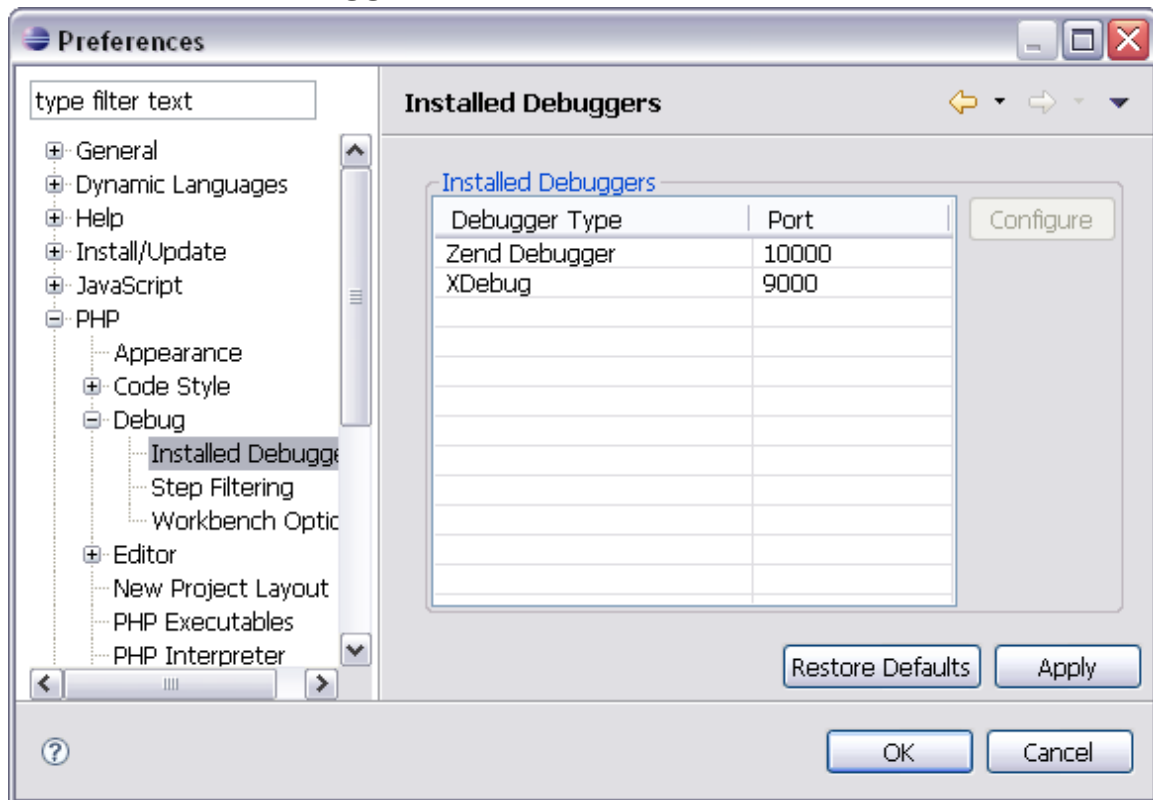
2.4.3 Step Filtering page

XDebug currently doesn't support this configuration option
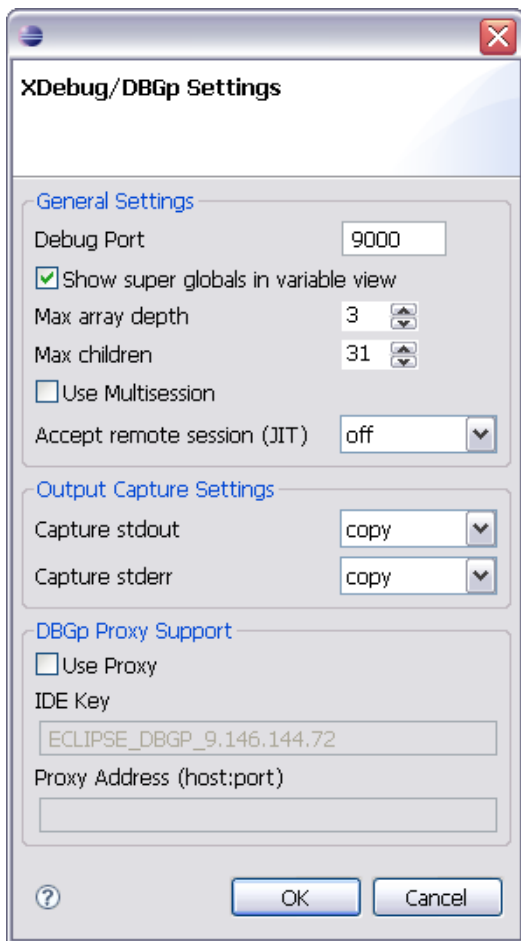
2.4.4 Workbench Options
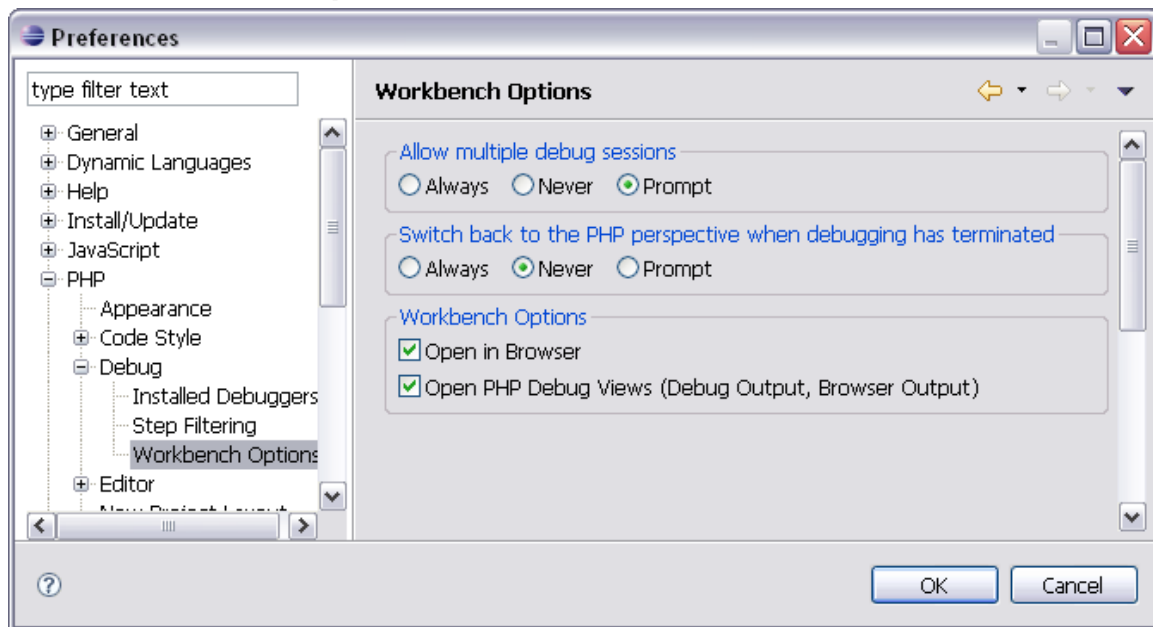
- Allow multiple debug sessions has no effect on XDebug support
- Open in Browser has no effect on XDebug support

2.4.5 Installed Debuggers

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 a single request for nested arrays. This doesn't restrict the depth that can be displayed, just how much information is retrieved per request to XDebug.
Max Children	Defines the maximum number of array children or object properties that will be initially retrieved. This doesn't restrict showing all the entries, just the number of entries retrieved per request.

2.4.6 Multisession

There are times when a developer will want to debug multiple PHP scripts for the same application simultaneously. In PDT you can only have a single web server debug session running (with associated path mappings) so multisession allows multiple debug sessions to run under a single application launch thus allowing a developer to debug multiple sessions at the same time.

2.4.7 Accept Remote Session (JIT)

There are many scenarios where you want to debug an application but don't want to do this via launching a php web script or php script from PDT itself. For example

- You want to use the Firefox XDebug extension to enable/disable debug mode for a web page and drive the application from your web browser yourself.

- You may want to run a script from the command line on a different machine and debug it on your local machine

- You want to use the JIT support of XDebug (xdebug.remote_mode=jit) to force the IDE to go into debug mode when an error condition occurs.

There could be other scenarios as well where this could be useful. You can configure the "Accept Remote Session" option with one of the 4 possible choices

- Off – any remote session request is rejected

- Localhost – only remote sessions initiated from the localhost are accepted

- Any – any remote session from any machine is accepted

- Prompt – prompt the user that a remote session initiation has been requested and for the user to accept or reject it.

Any remotely initiated session will be assumed to be a Web server based initiation, and thus only one debug session can exist within PDT (however you can make use of multisession support to allow for multiple sessions, but remember that they will share the same path mappings). If you want to have multiple command line debug sessions then you need to set the environment variable **DBGP_COOKIE** to a unique number for each separately running command line php script.

2.4.8 Output Capture Settings

Output Capture allows you to collect output from your php script and display it in the Debug Output view and the Browser Output view. The options available are

- Off – no information is captured, nothing will be displayed in the views (output from a php script, a non web page script, will still be displayed in the console)

- Copy – information will be captured and displayed in the views. Also output from the php script, a non web page script, will also be displayed in the console

- Redirect – information will be captured and displayed in the views. Redirect should stop output from the php script being displayed in the console as well, but as of XDebug 2.0.3 there is a bug which means redirect works the same as copy.

The capture option for "stderr" currently has no effect as of XDebug 2.0.3. In the future xdebug may make use of output capture on stderr such that you can control the capture of messages from the php error handler (the error handler outputs notice/warning/error messages).

2.4.9 Proxy Support

PDT 2.0 supports the use of a DBGp Proxy. The proxy allows multiple developers to debug a single or multiple applications on the same web server. PDT 2.0 only supports the ActiveState DBGp Proxy.

To enable support for a proxy, check the use proxy box. The IDE Key is automatically generated but it is important that all IDEs that register with the proxy have a unique IDE key.

Finally you need to enter the address of your configured proxy. For example if your proxy is configured on your machine listening for IDE connections on port 9001, then enter "127.0.0.1:9001".

One important point, if you are running the proxy on your machine you must ensure that PDT XDebug port is not the same as that of the proxy ports. The proxy uses 2 ports, one for listening for debug connections from php and xdebug (default 9000), and the other is for IDE connections (default 9001).
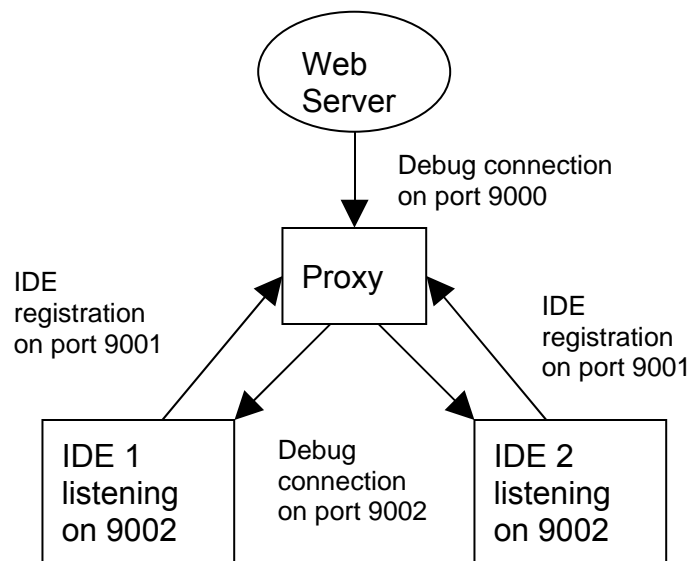
In the above example, each IDE has had the xdebug port changed to listen on port 9002. The proxy is listening on port 9000 for debug connections and port 9001 for ide registrations.

2.4.10 Configuring your web browser

In PDT you can choose which browser is used and whether to have a window inside of PDT for your web browser or for it to run as an external application. You configure this in Windows→Preferences and select "Web Browser"
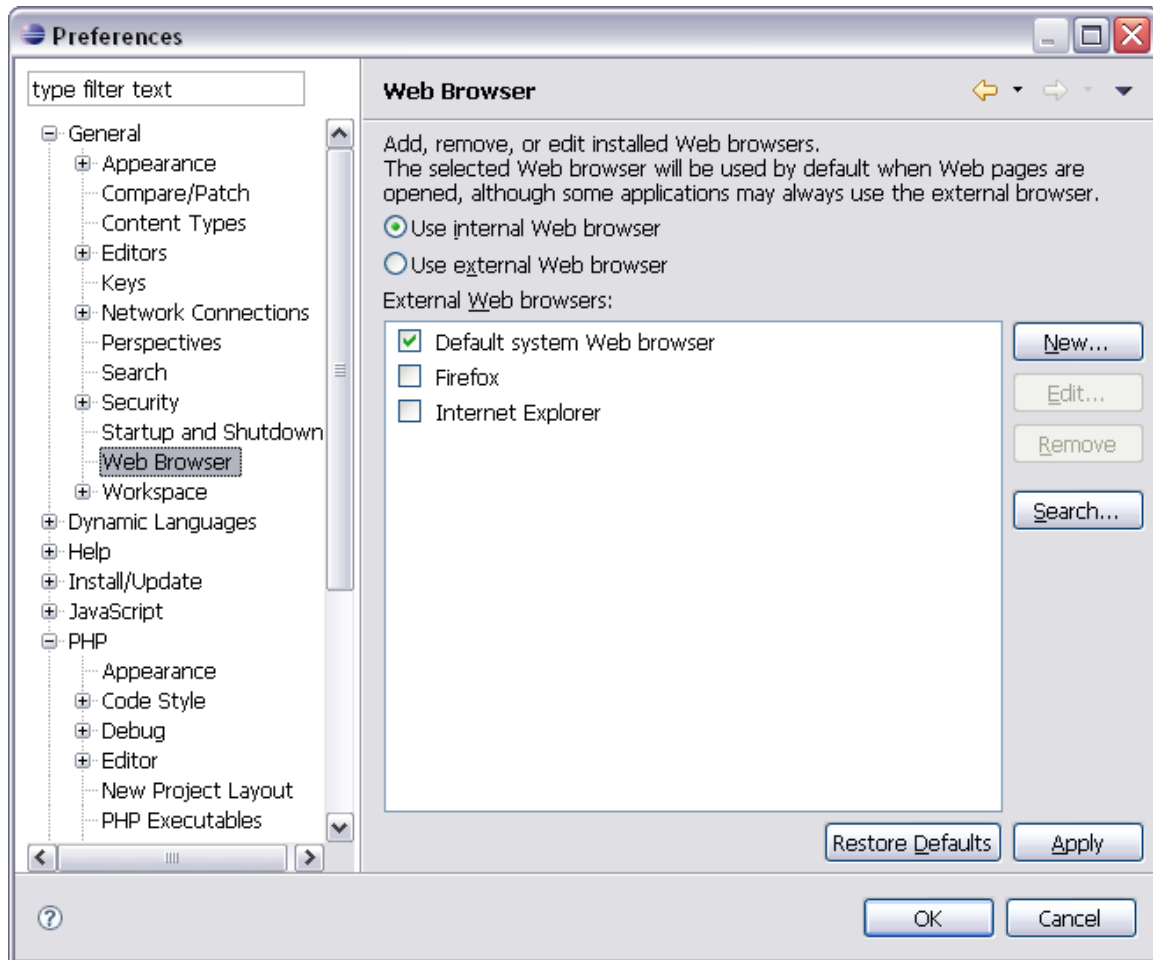
2.4.11 Testing your configuration

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.

2.5 Debugging using XDebug

PDT can be used to debug standalone php scripts as well as web based php applications. You can use PDT to launch the standalone script or web based application, or you can get PDT to switch to debug mode when an application requests to be debugged (using something like the XDebug Firefox extension to enable xdebug for web applications, or by enabling xdebug on the command line for the CLI script).

2.5.1 Debugging standalone scripts launched by PDT.

You will need to define a PHP Executable location. In preferences, select PHP Executables and press the Add… button to get the following dialog
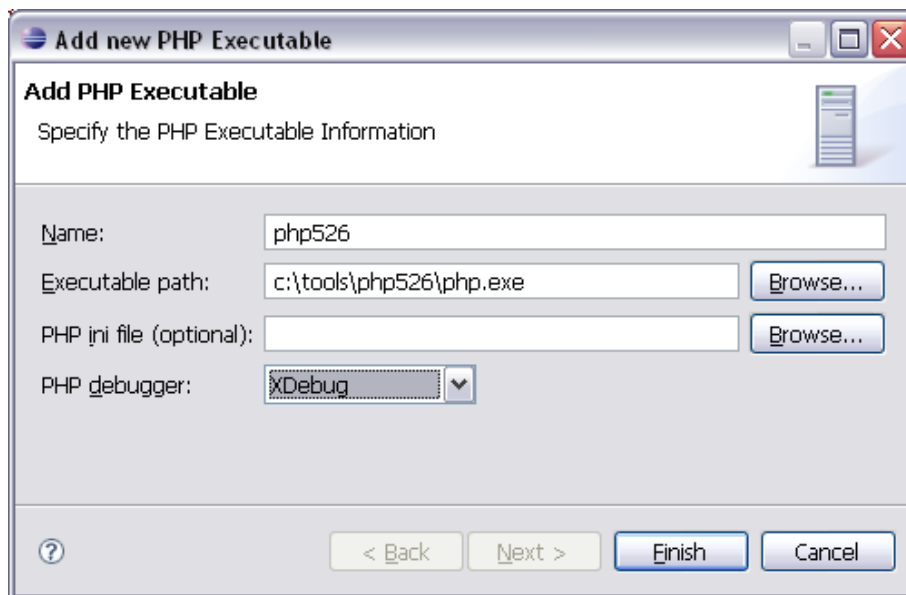

Ensure you have selected the PHP debugger as XDebug and define the path to where you have PHP with XDebug setup.

You can either create a new PHP script launch in the Debug Configurations panel, or you can select the script in your workspace and invoke the "Debug As→ PHP Script" context menu.
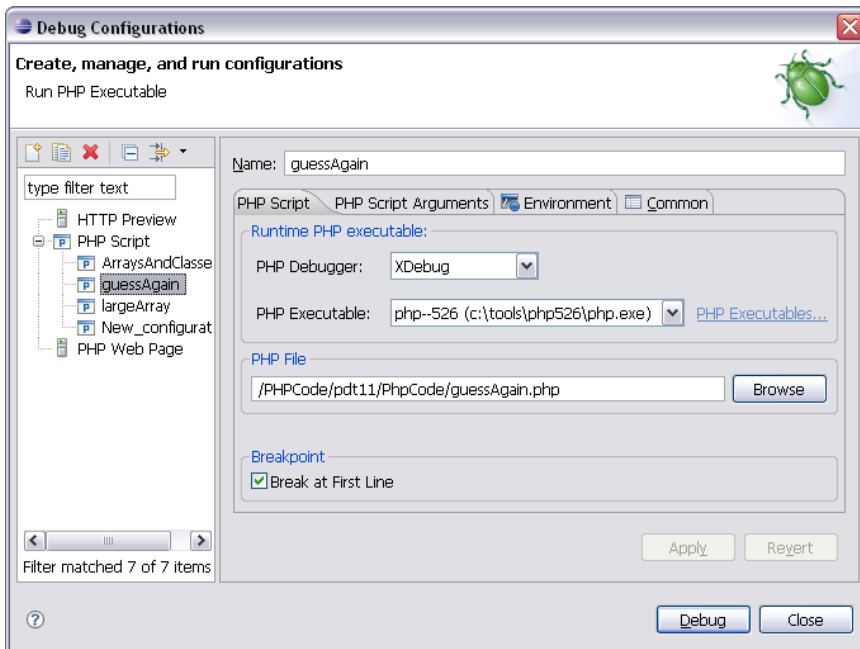
2.5.1.1 Disconnecting while debugging a script

Disconnect will terminate the debug session, but the script will run to completion. In order to disconnect a script, you need to ensure that the top level launch (guessAgain in the example below) or the entry beneath called "PHP Application" is selected in order to activate the disconnect button.
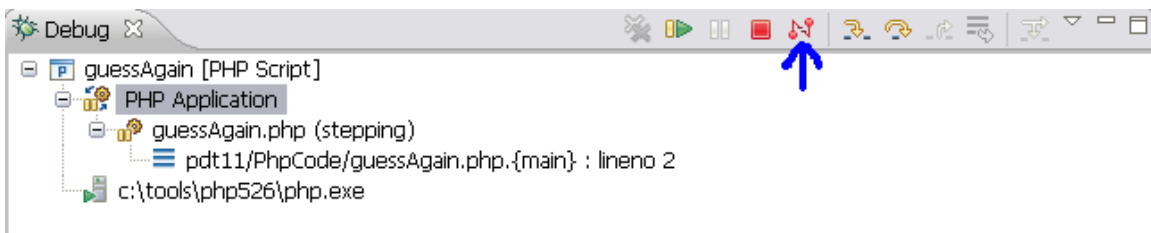
2.5.2 Debugging a web based application

PDT supports different scenarios for this, allowing you to debug an application locally on your machine to debugging an application on a remote web server, but one fundamental requirement is that the same level of code is available to PDT that is being executed by the web server. You can achieve this in many ways, for example

- Point your local web server document root to your workspace or a project in your workspace

- you can use eclipse to create links to local files or folders and make them part of your PHP project

- You could look at Remote System Explorer from the eclipse Target Management project to share files across different systems

Unless you have chosen the 1st option, your web server and PDT may use the same file, but its path will not be seen as the same. To address this, a concept called path mapping is used. Path mapping is about trying to automatically locate the local file that PDT should display to the user based on the file that is being executed by the Web Server. The filename will be the same but the path qualifier will be different so rules are needed to convert from the Web Server path to the PDT path and vice-versa. Path mapping will be discussed later, PDT does attempt to generate path mappings automatically for you but there may be occasions where you need to do some manual configuring.

2.5.2.1 Define your Web Server

You will need to define your server to PDT. PDT has a default web server defined as http://localhost but this may not be what you want. To define a new server Go to "Preferences" in the "Window" drop down menu, expand PHP and select PHP Servers
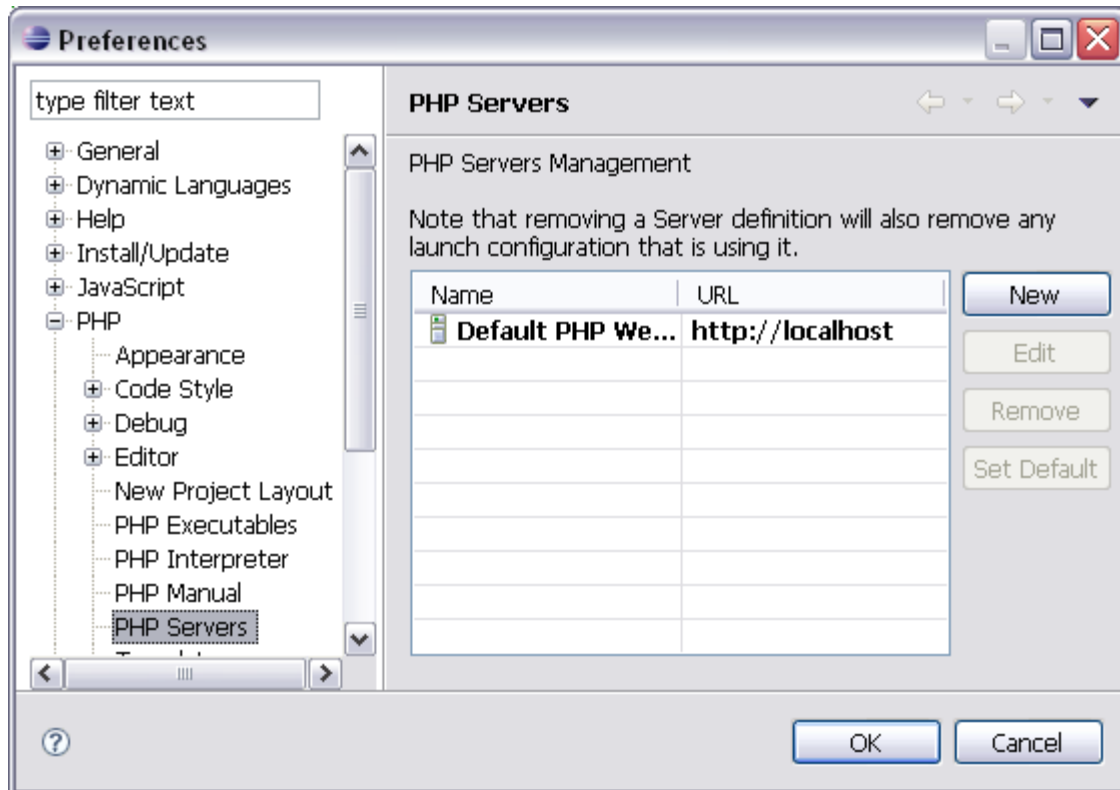
Press New to add a new Web Server with the appropriate base URL for that server, eg http://myserver.com:8080 and press Next.

Here you see the path mapping tab where you can define the path conversion between your web server and PDT. Unless you get a problem you shouldn't have to provide your own configuration here.

Press Finish to complete the server definition.

2.5.2.2 Create a PHP Web Page Launch

You can do this in 2 ways, either in the Debug Configurations dialog or you can using the "Debug As→PHP Web Page" pop up menu. Either way the most important thing here is that the php script you have selected or entered into the dialog matches the script that will be executed when the particular URL that you have also provided is invoked by a Web Browser. This is how PDT creates the path mapping information for you. Getting this wrong could result in PDT showing the wrong file or no file at all.

The Debug As→PHP Web Page is rather limited in what you can enter, so it is recommended you use the Debug Configurations to define your launch.
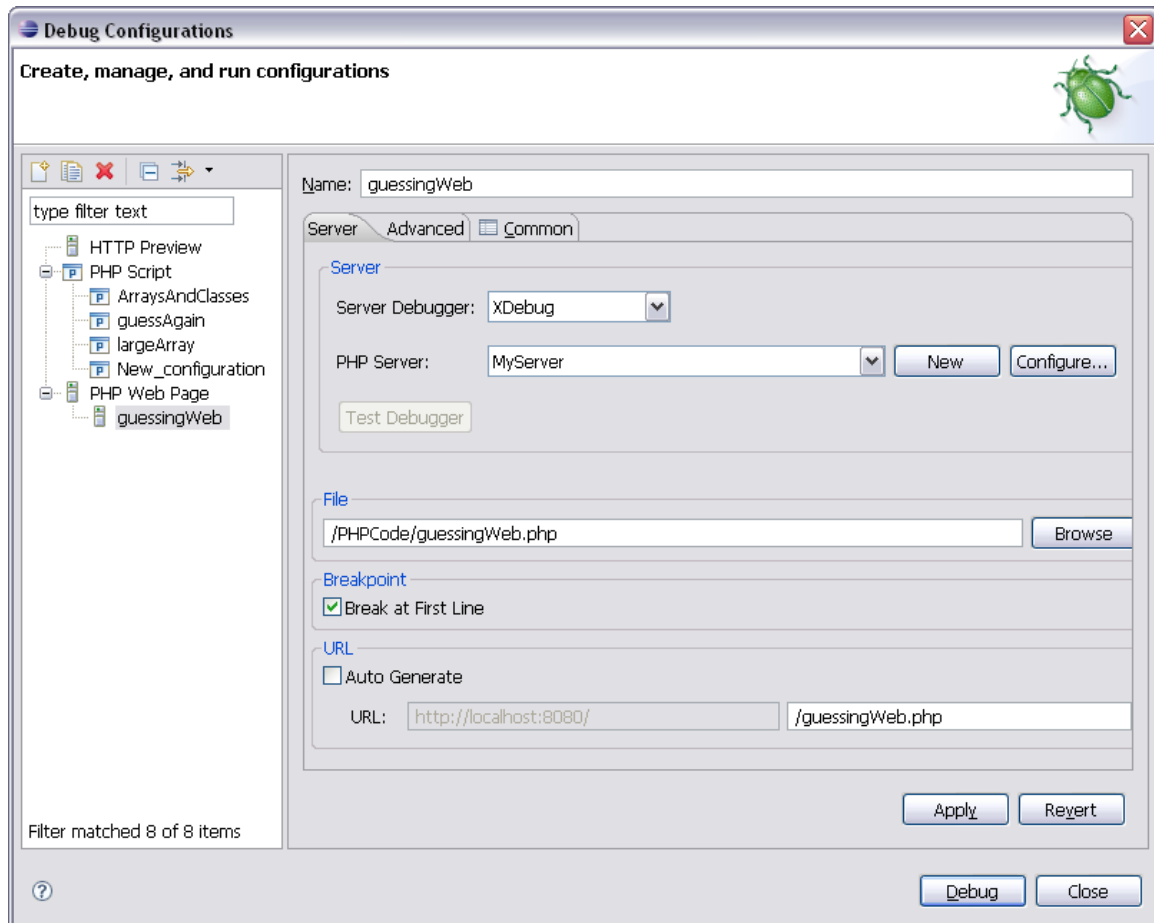
In the above example I have defined my launch and configured the URL myself by removing the "Auto Generate" option.

Note that for Xdebug, the Advanced tab options have no effect.

2.5.2.3 Debugging the web application

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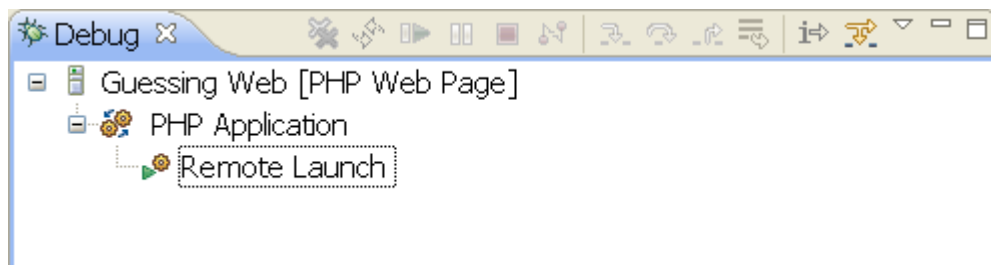
When debugging a script you should see something like this in the debug window
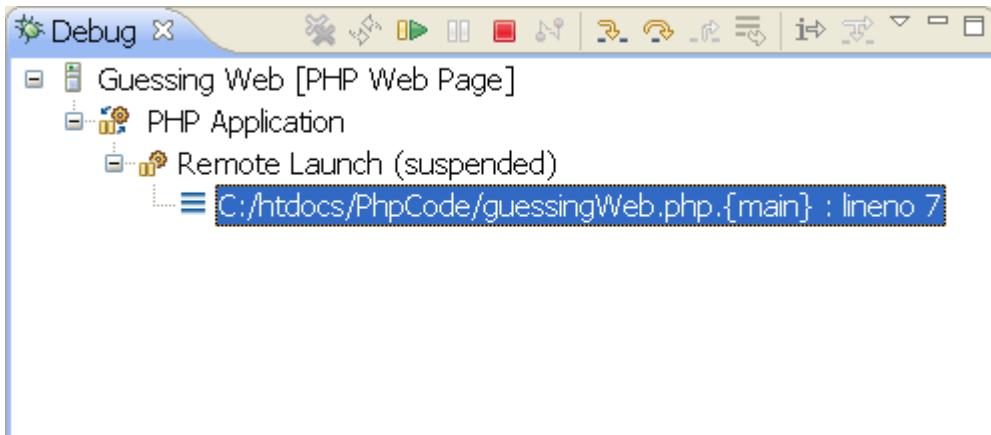

Whenever a request completes, the Web launch debug will remain active waiting for the next request to be debugged until you explicitly stop the debug session by terminating it, ie by selecting either the PHP Web script launch, the "Remote Launch" child entry, or the "PHP Thread" child entry and press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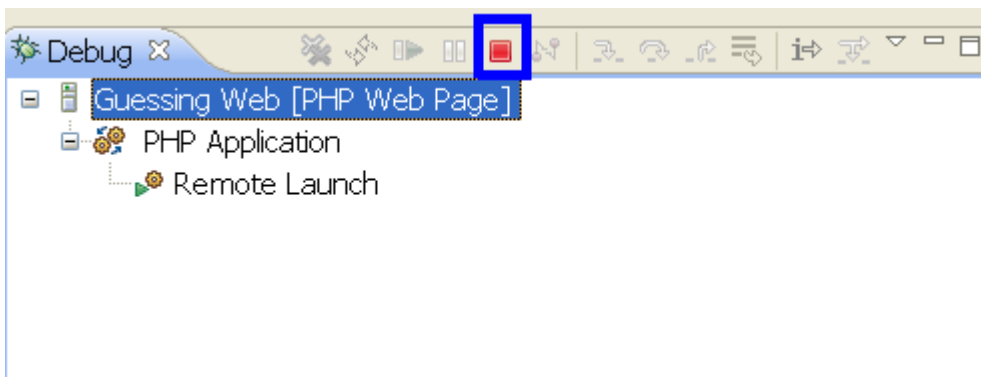
2.5.2.4 Disconnect

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").
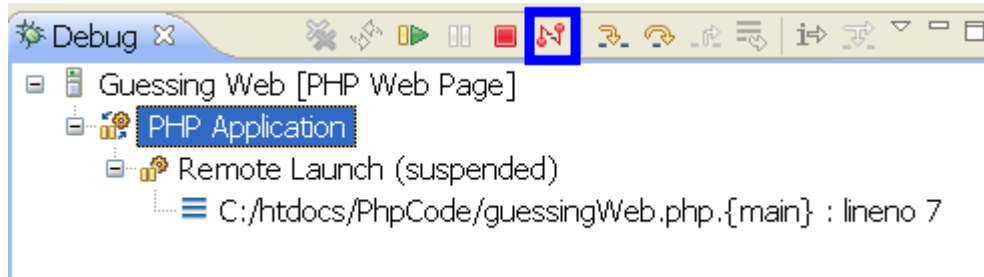

When you disconnect from a web launch, the web launch remains active but stops the script that is currently being executed by your web server. Your browser will still have the XDebug cookie registered which means if you go to that browser press enter or return back to the original launch URL, you should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

### 2.5.3 Debugging using Remote Session initiation

First enable Remote Session Initiation in the xdebug preferences page. With that done you can use something like XDebug Firefox extension to turn on the XDebug cookie which will cause xdebug to start debug sessions or turn on xdebug JIT support.

If you specify "prompt" for Remote Session Initiation, then when a request comes in a dialog will be displayed
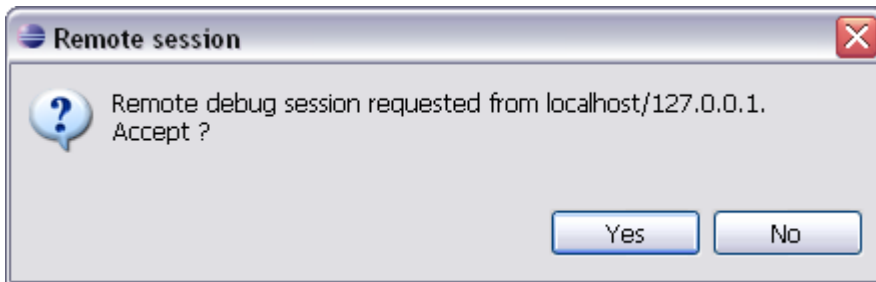

Select yes to accept this session, or no to reject.

When a session is accepted, it is very likely that a path mapping will be required because PDT is not sure which file it needs to work out the path mapping from. PDT is likely to prompt with a dialog similar to the following
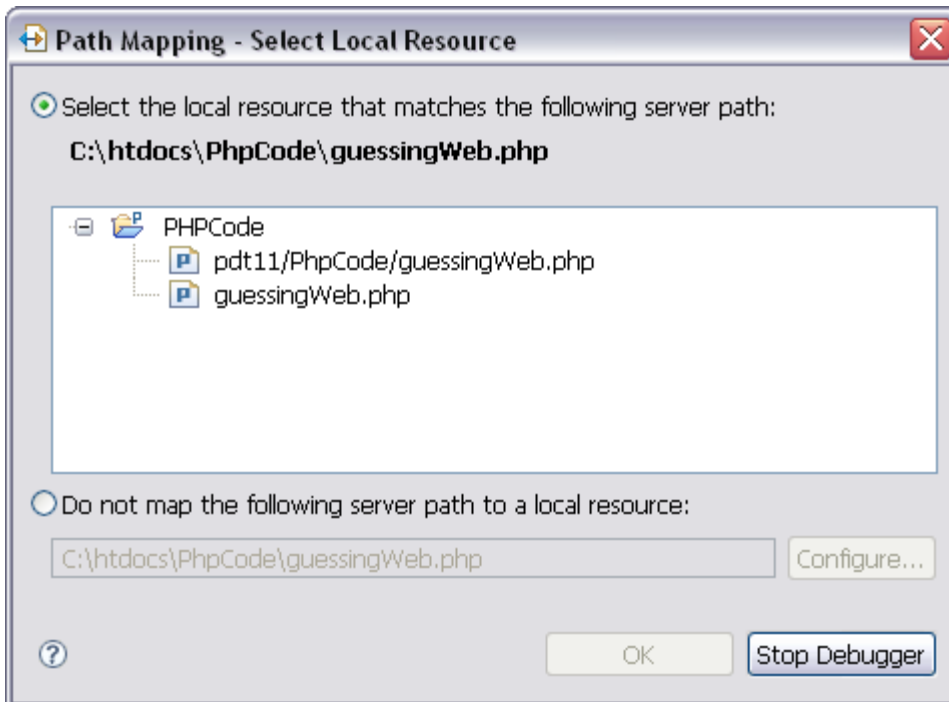
here you select the appropriate file which should be used based on the current script that is being executed by the server (in this case c:\htdocs\PhpCode\guessingWeb.php). In my example I have 2 copies in my PHPCode project and I should select the correct one.

The above will result in a debug session which is like a PHP Web Page launch being invoked.
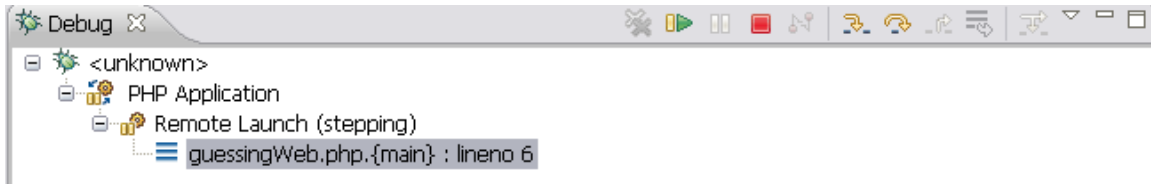
Note that the launch is showing as <unknown> as it did not originate from PDT.

You can also use Multisession support as well.

2.5.3.1 Remote Session with CLI PHP

If you are launching CLI PHP scripts remotely and you want to use Remote Session Initiation, then this is possible as well, first you need to set or export the Xdebug IDE Key as follows

```
export XDEBUG_CONFIG="idekey=ECLIPSE_DBGP"
```
or
```
set XDEBUG_CONFIG="idekey=ECLIPSE_DBGP"
```
(or if you are using the proxy, enter your unique idekey defined in the preferences).

And you also need to set or export a DBGP_COOKIE integer value, for example

```
export DBGP_COOKIE=1223
```
or
```
set DBGP_COOKIE=1223
```
Doing this ensures that PDT recognizes this is a CLI PHP invocation. The value used can be anything.
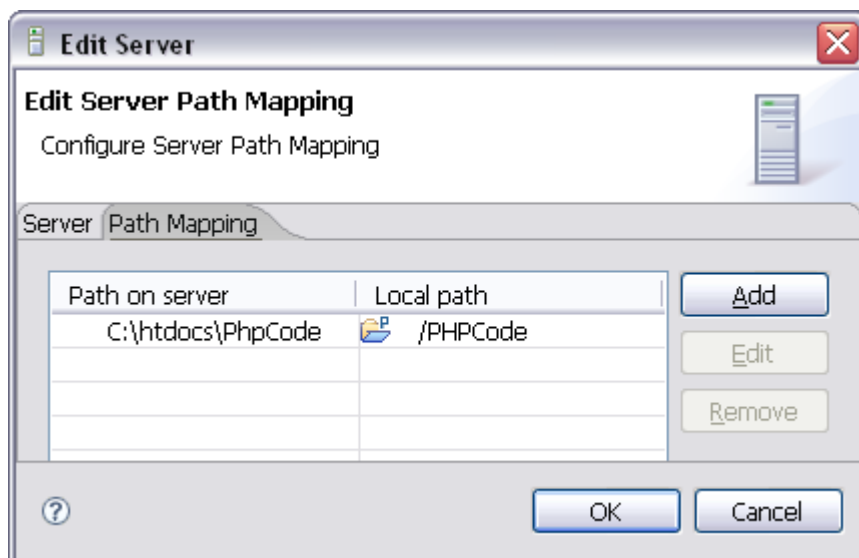
2.5.3.2 Path Mapping

No correlation is made between defined PDT Servers and the server that initiated the remote debug session. Because of this, there is no initially defined path mappings available and are created as needed during the debug session. Once the debug session is terminated, those path mappings are lost.

2.5.4 Advanced Path Mapping

As described before, the concept of path mapping is to try to match the script executing on the server with a script that PDT can access. The scripts that PDT can access must be identical to those stored on the server and also the directory structure should 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. Also when you want to step through a file, you want the script that is currently being run by the web server so again a mapping must occur and the correct file can then be loaded.

As previously described, PDT creates path mapping information from different sources. One is from the PHP Web Page launch which maps the file executed when the URL is invoked to that of the script file listed in the launch. Another is when PDT prompts you to specify which file to select when in a debugging session (seen when doing remote session initiation).

Finally you can modify path mapping yourself for server. Go into Preferences, select PHP Servers and edit your particular server and select the path mapping tab

In the above example, we see a path mapping between a real directory and an eclipse workspace project which was automatically generated by PDT but could have been entered manually as follows
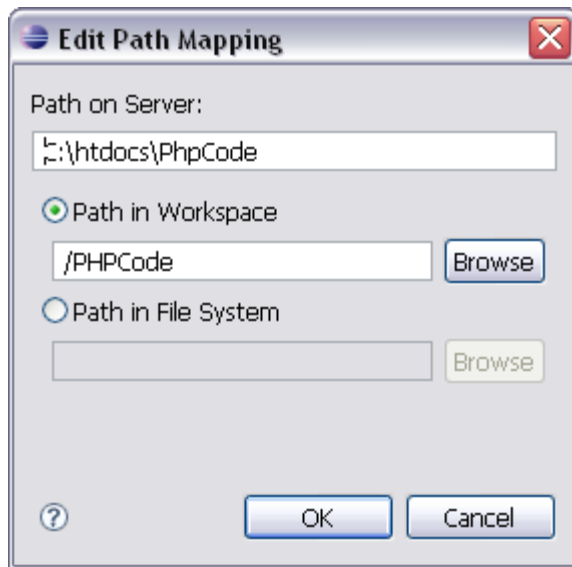
Path Mappings will also work between different operating systems for the server and the PDT user. The server could be linux and PDT could be windows.

2.6 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.

2.6.1 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
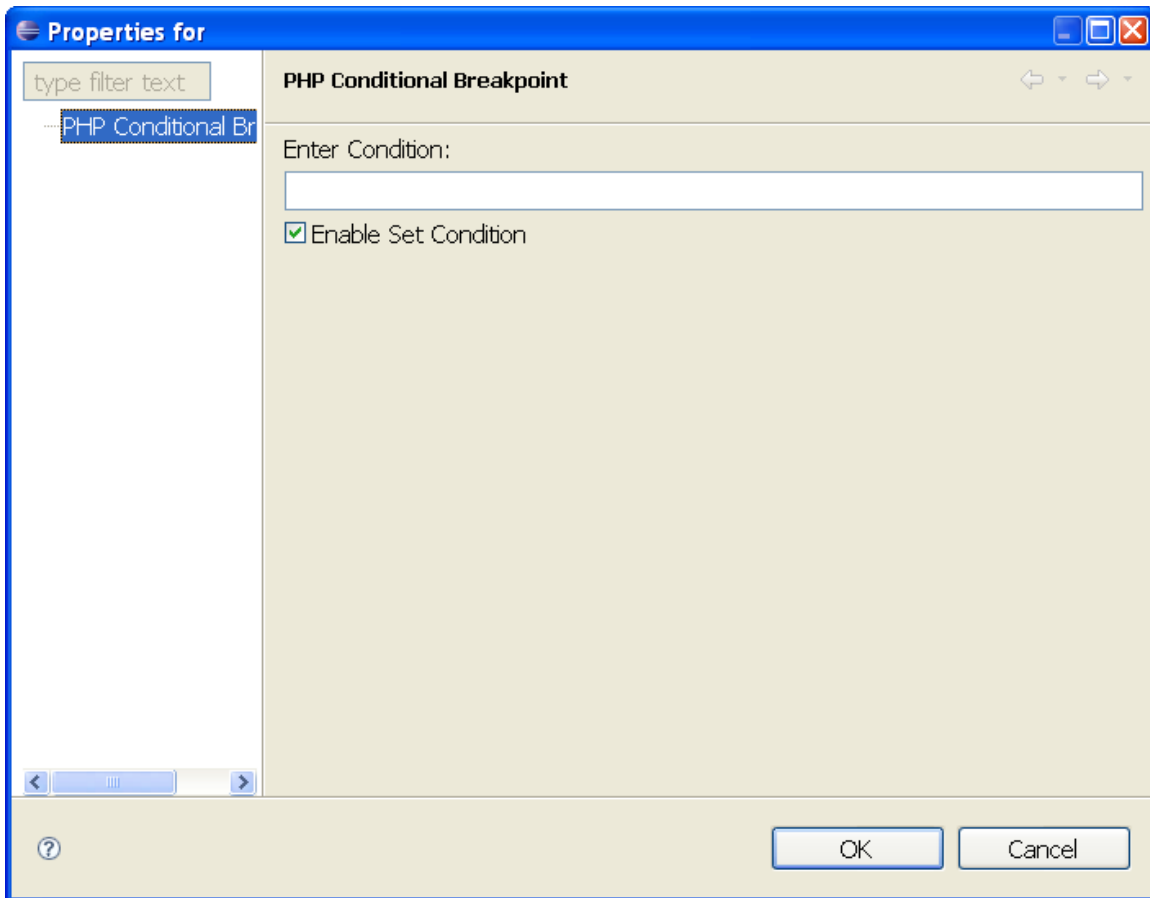
Where you can set the condition

2.6.1.1 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)
```

2.6.1.2 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
```

2.7 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)
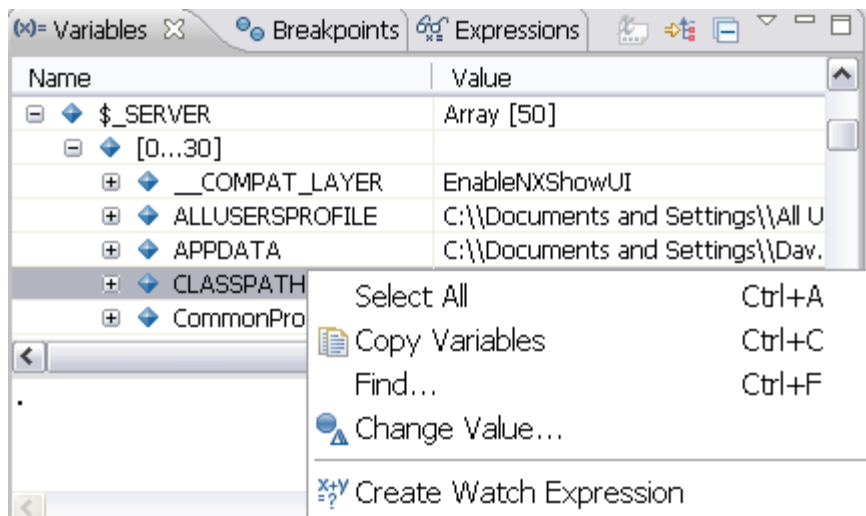
The following shows executing scripts, which include their stack frames
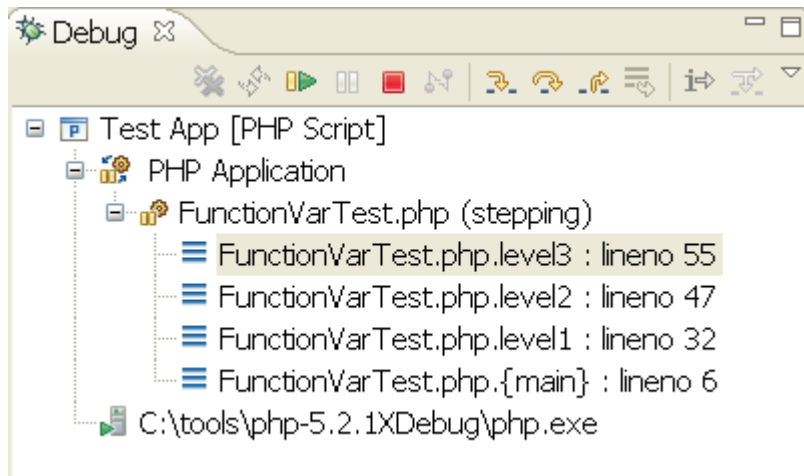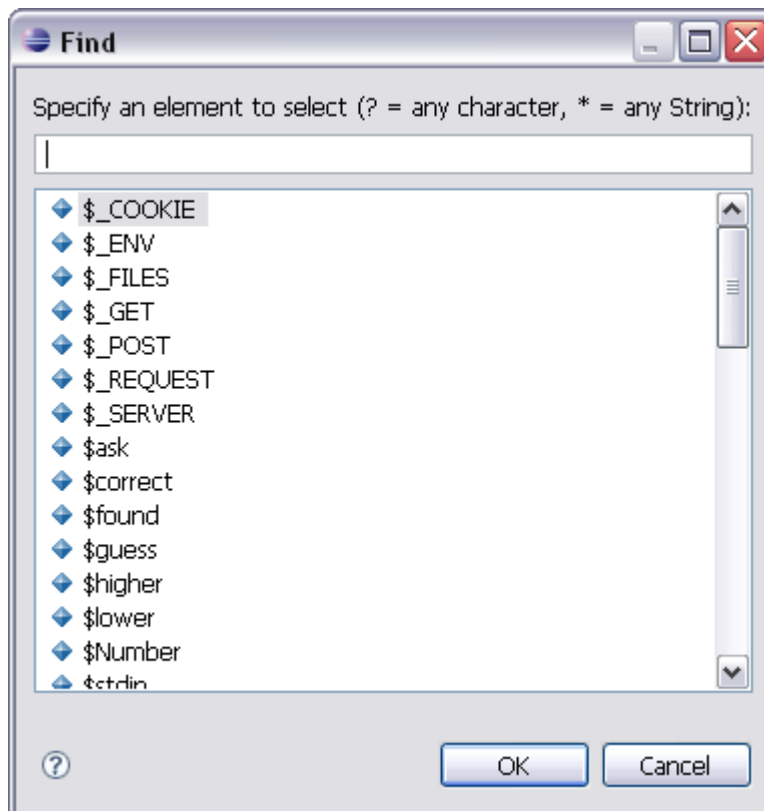

The list of variables and their values exposed for the selected stack frame
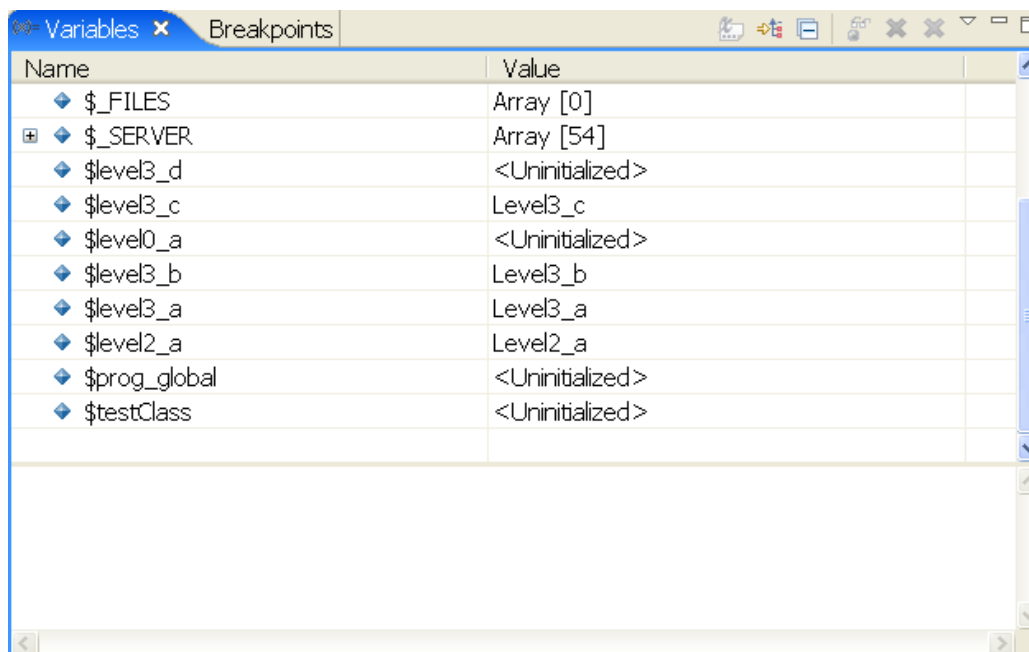
The variables view also has a useful find facility from the popup menu
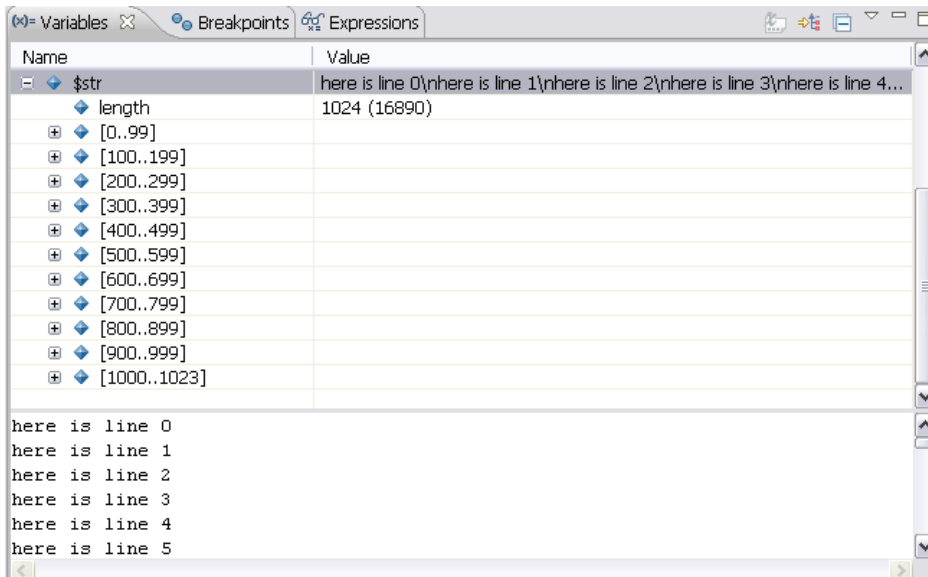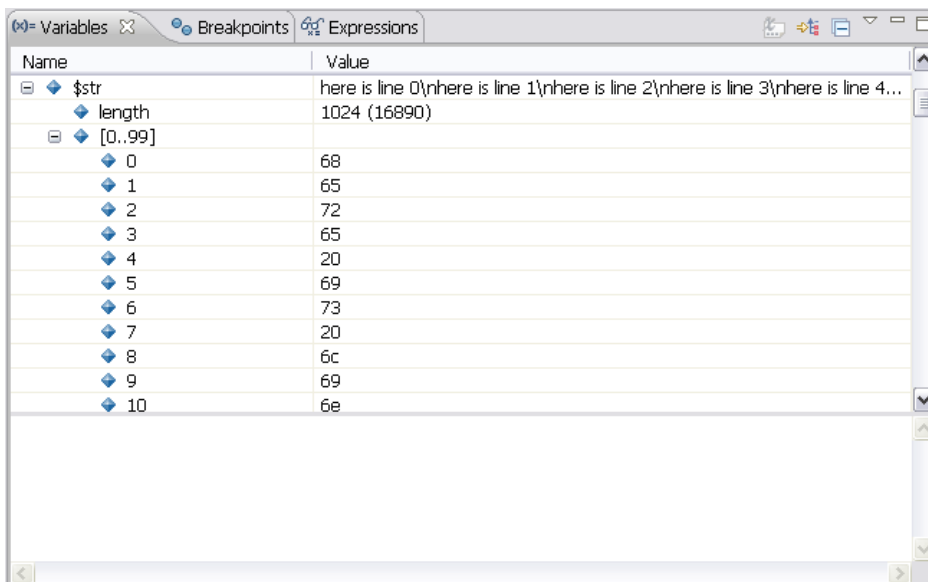
So an example of the find dialog is

2.7.1 Strings in the variables view

In PHP, strings can represent binary data or character data. Also strings can be very large in PHP. To accommodate this, the way strings are presented in the variables view has been changed for PDT 2.0.

Name	Value
$str	here is line 0\nhere is line 1\nhere is line 2\nhere is line 3\nhere is line 4...
length	1024 (16890)
[0..99]	
[100..199]	
[200..299]	
[300..399]	
[400..499]	
[500..599]	
[600..699]	
[700..799]	
[800..899]	
[900..999]	
[1000..1023]	

```
here is line 0
here is line 1
here is line 2
here is line 3
here is line 4
here is line 5
```

In the above example the $str shows a single line representation. Underneath you have the current length of the string and in brackets the true length of the string. The above is saying that the complete length of the string is 16890 bytes, but only the first 1024 bytes are being shown here. Following that you can get a byte view of the string split into blocks

| Name	Value
$str	here is line 0\nhere is line 1\nhere is line 2\nhere is line 3\nhere is line 4...
length	1024 (16890)
[0..99]	
0	68
1	65
2	72
3	65
4	20
5	69
6	73
7	20
8	6c
9	69
10	6e

you cannot change individual bytes however.

The final part is the view is the string in a multiline view.

If however you need to view the complete string, then you can get the complete string by selecting the string and invoking "Change Value…" on the popup menu
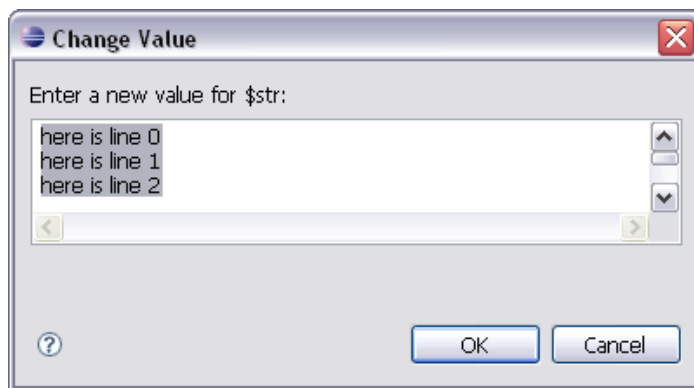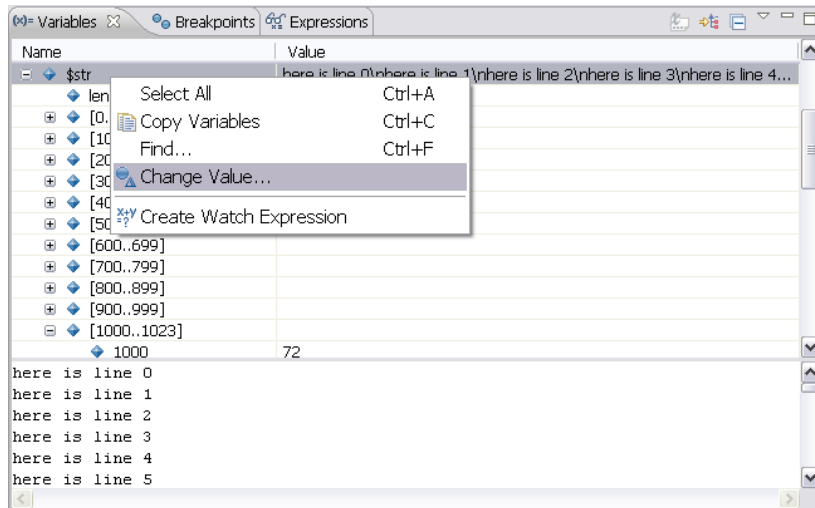

Then press cancel so that you don't change the value (unless you want to of course), and you will now see the $str value has changed in the variable view to include all the contents of the string.
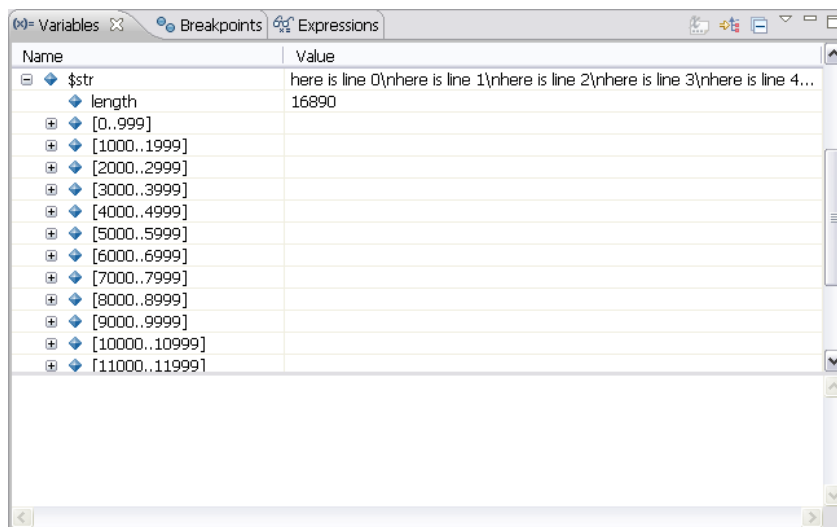
2.7.2 Hover

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
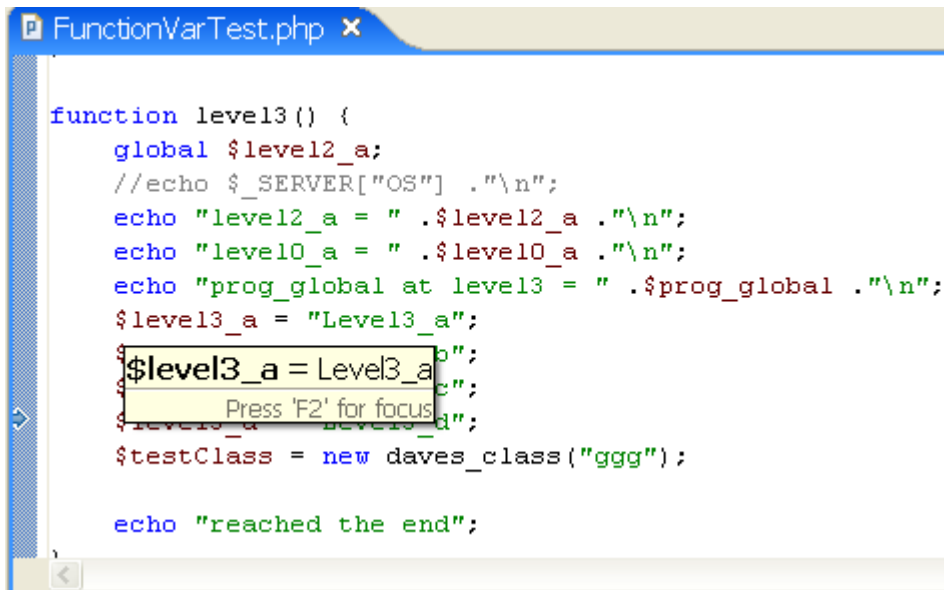

2.7.3 Expression view

You can also create expressions in the expression window which will be evaluated and displayed whenever your script or request is suspended.
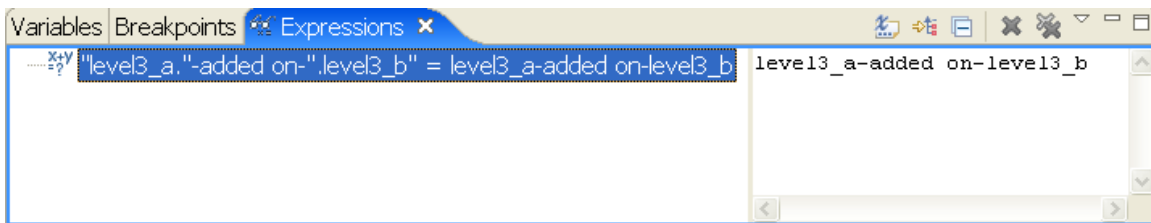
One quick way to add a variable to the expressions window is through the watch pop up menu item. To do this highlight a variable (include the $), you can do this by double clicking the variable, bring up the pop up menu and select "watch"

2.7.3.1 Issues with Expressions

The PDT expression evaluator uses "eval" to determine the expression. This makes it powerful but can also cause the script you are debugging to suddenly terminate. The reason for this is that every time a script is suspended, all enabled expressions are evaluated at the current context. If you try to invoke functions in an expression and that function is not available then the script will terminate and you will see a dialog
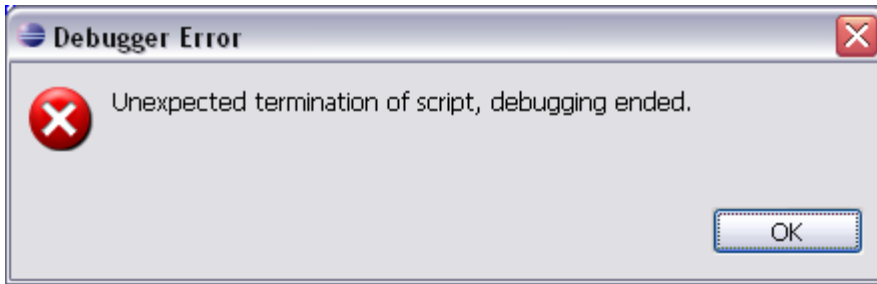
If you are getting this error then you may have to disable your expressions to stop your script from terminating.

2.8 Other 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

2.9 Tips

2.9.1 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

- A 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.

- A PHP script launch produces a firewall pop-up which you have missed or denied

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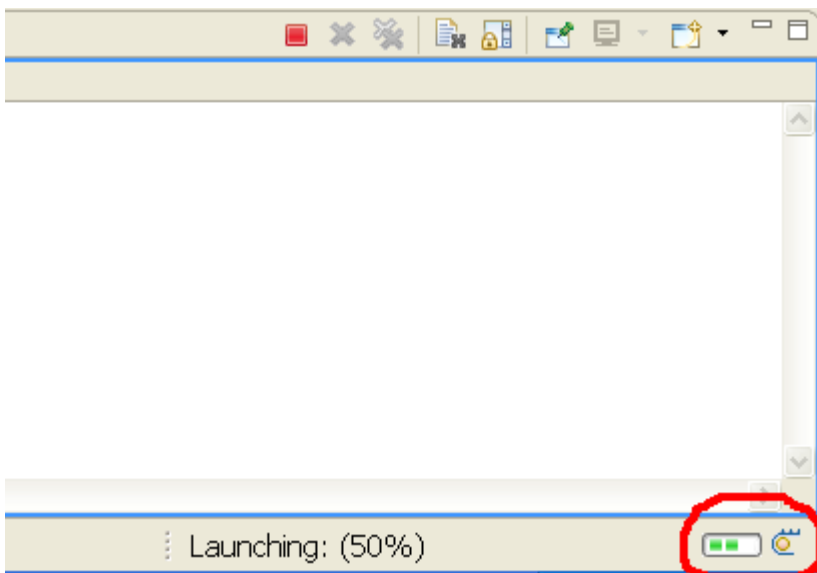
You can terminate the debug session in the usual ways, as well as terminating the launch from the launch view.

2.9.2 Watch Expressions returning Objects/Arrays

Watch expressions are evaluated and the results are returned. If your expression returns and array of data (for example when you watch something like get_include_files()), there is no variable defined to hold the results in PHP. xdebug by default returns only the 1st 32 elements of the array which is why you can see this information in the watch. However when you try to expand the next set of elements it needs a way to query the next set of elements, and there is no way to do that as there is no variable in PHP which is holding the results. It is also not possible to re-evaluate the expression and get it to pass the next set of elements.

One way round this is to use the "Max Children" preference option. This would control the amount of information xdebug would send back when querying information about arrays/objects such that if you set this value to say 100 it would transfer 100 elements back on each request (and that show up as array chunks of [0-99] [100-199] etc, and for the watch expression you would set it to a value such that all the information you require will appear in the first block.

END-OF-DOCUMENT