

Xdebug

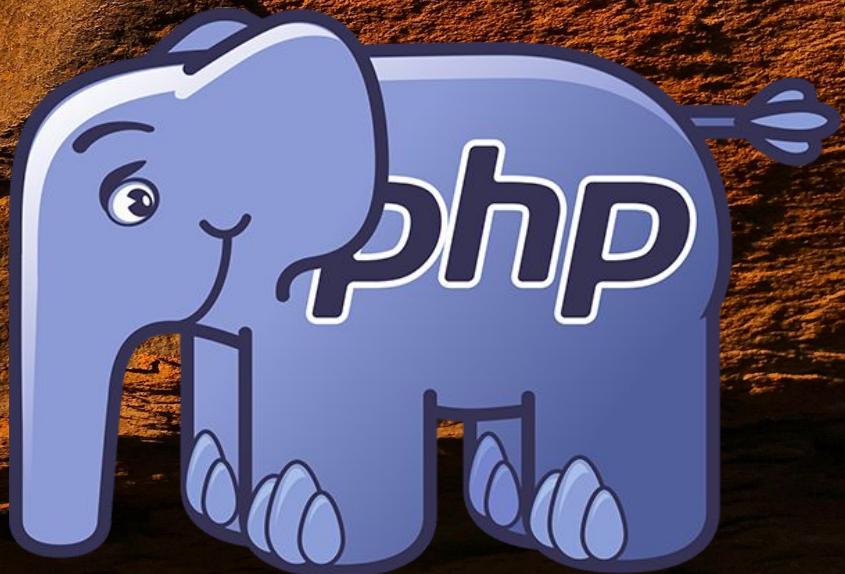
Debugging and profiling PHP

Prepared for PHP Sydney meet-up

28th November 2019

Kev Staunton-Lambert

pyrmontbrewery.com



what is the Xdebug Zend Extension module?

PHP Zend VM is the opcode stack machine interpreter

Xdebug is Derick's extension (*donations welcome*) which uses the common DeBugGer protocol (DBGp) to step opcodes and re-interpret what's on the stack

This allows you see/rerun code remotely with live input

Works well with FPM-PHP, i.e. debug your webservices!

But will hold up a process so important to use separate IP
FastCGI port or dedicated debugging endpoint (not prod!)

how does it work?

Traverses opcodes until hits a PHP line or sees a condition

(Line 10)

```
require_once("lib/some_other.class.php");
```

line	#	*	E	I	O	opcode	operands
9	0		E	>		EXT_STMT	
	1					EXT_FCALL_BEGIN	
	2					INCLUDE_OR_EVAL	'lib%2Fsome.class.php', REQUIRE_ONCE
	3					EXT_FCALL_END	
10	4					EXT_STMT	
	5					EXT_FCALL_BEGIN	
	6					INCLUDE_OR_EVAL	'lib%2Fsome_other.class.php', REQUIRE_ONCE
	7					EXT_FCALL_END	

Vulcan Logic Dumper (vld) - also Derrick's work -
<https://github.com/kevleyski/vld>

why would I want that? echo is my best mate!

Ever written a dev routine to dump a complex object state?

Auto stack trace on exception useful?

Closure debugging

Parallel debugging
(multiple debug ports and IDE ids)

The screenshot shows a PHP debugger interface with the following details:

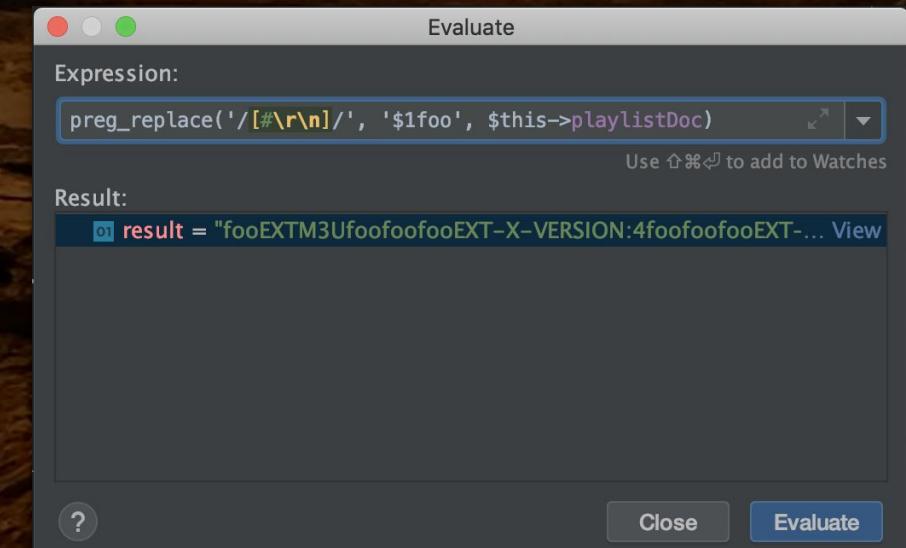
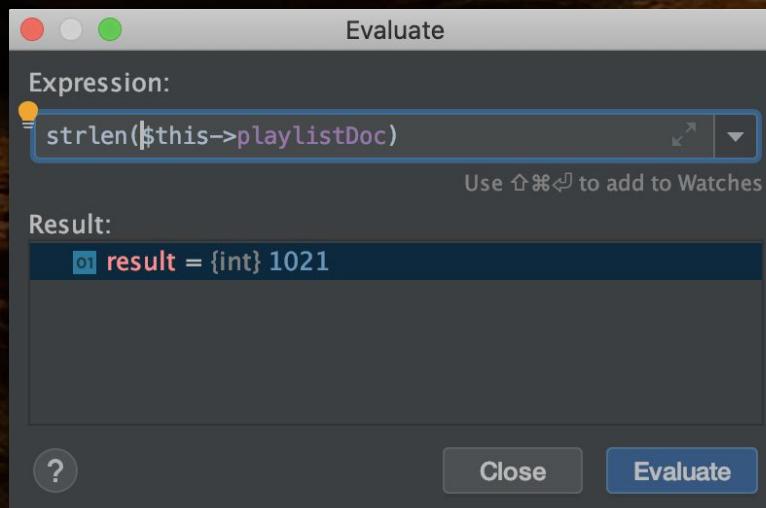
- Code:**

```
2 $favouritePyrmontBeer = 'Half Way Stout!';
3
4 function choosePyrmontBeer()
5 {
6     $favouritePyrmontBeer = 'Purgatory Pale Ale!';
7     $func = function() use (&$favouritePyrmontBeer) $favouritePyrmontBeer: "Purgatory Pale Ale!"
8     {
9         $favouritePyrmontBeer = 'Wokos Cider!';
10    };
11    $func();
12    echo 'My favourite Pyrmont beer is ' . $favouritePyrmontBeer; // My favourite Pyrmont beer is Wokos Cider!
13 }
14 choosePyrmontBeer();
choosePyrmontBeer() > X0
```
- Debug:** PHP closure.php
- Frames:** closure.php:9, {closure:/Users/klambert/v...}, closure.php:11, choosePyrmontBeer(), closure.php:14, {main()}
- Variables:**
 - \$favouritePyrmontBeer = "Purgatory Pale Ale!"
 - \$_SERVER = {array} [31]
- Contextual menu (Variables context):**
 - + New Watch...
 - Remove All Watches
 - Inspect...
 - Mark Object...
 - Set Value...
 - Copy Value
 - Compare Value with...
 - Copy Name
- Bottom right buttons:**
 - Evaluate Expression
 - Evaluate In Console

how else is Xdebug useful - Real time evaluation

Evaluate / Re-run a function whilst on a breakpoint

Helps you figure out regexp/xpaths against your *real data* without restarting script



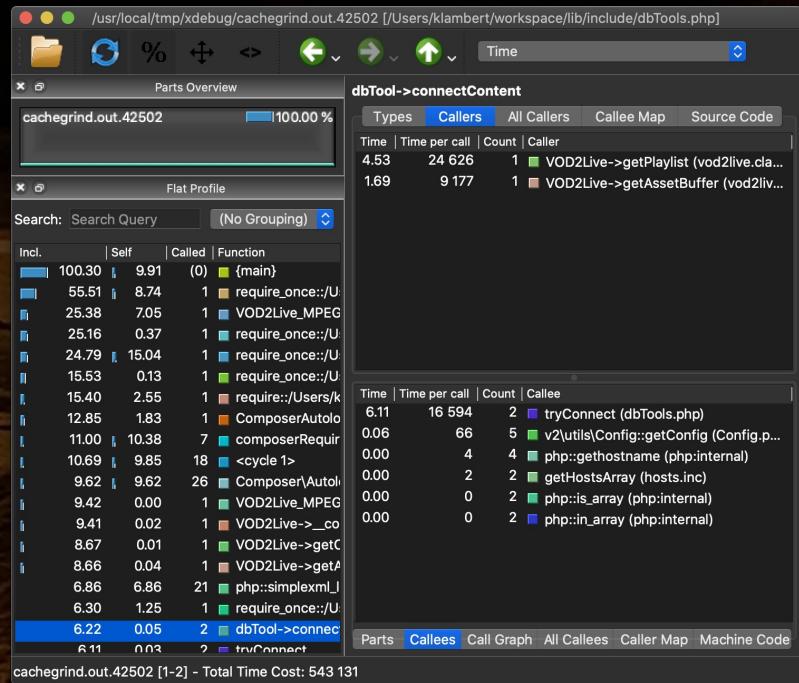
what else? Profile your code!

Want to actually see how well your last edit just ran?

Compare an earlier metric?

Which sets of opcodes (functions) get hit most often? Where to optimise

See code coverage, whats not being hit by PHPUnit



how to set up the extension (php.ini or php.d)

Grab the extension via pickle for your version of PHP

pecl install xdebug

Add the extension to <php_version>/php.d/15-xdebug.ini

zend_extension="php/modules/xdebug.so"

Note: If you want to use Xdebug and OPCache together, you
must load xdebug.so *after* the opache.so extension

(and, if using FPM) systemctl php-fpm.service restart :-)

how to set up the extension - Docker container

Of course can spin up modified Dockerfile for your existing container modified for remote debugging purposes...

```
RUN echo "xdebug.remote_enable=1" >>  
/etc/php/7.3/php.d/15-xdebug.ini
```

```
RUN echo  
"xdebug.remote_host=your_host_name_or_ip" >>  
/etc/php/7.3/php.d/15-xdebug.ini
```

how to configure remote server debugging (.ini)

[Xdebug]

```
zend_extension=/usr/local/lib/php/pecl/2018073/xdebug.so  
xdebug.remote_host=<your_ip_address>  
xdebug.remote_autostart = 1  
xdebug.remote_enable = 1  
xdebug.remote_port="9001"  
xdebug.idekey="klambert2"  
xdebug.remote_connect_back = 0
```

Autostart is particularly important when debugging parallel/multithreaded code else you need to send XDEBUG_SESSION_START=1 for all threads

Set this to 1 if you have multiple people debugging on the same server otherwise use remote_host

how to set up local debugclient (opensource)

Easy enough to just build it yourself from Xdebug repo

```
https://github.com/kevleyski/xdebug.git
```

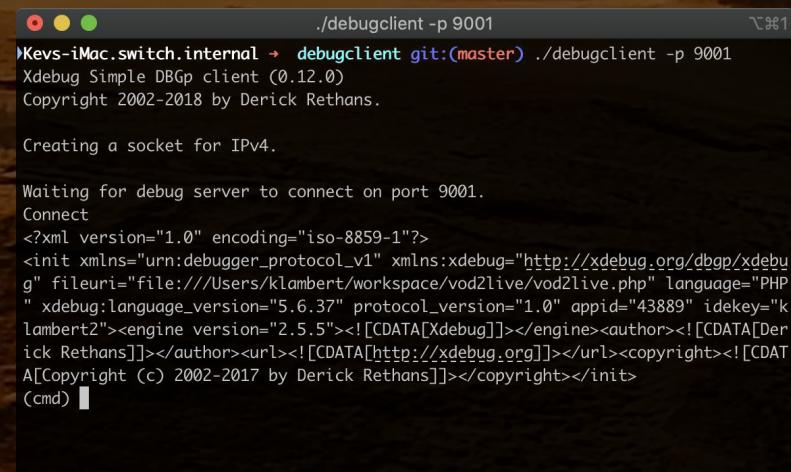
```
cd debugclient
```

```
autoconf
```

```
./configure --with-libedit
```

```
make
```

```
sudo make install
```



```
./debugclient -p 9001
Kevs-iMac.switch.internal ➔ debugclient git:(master) ./debugclient -p 9001
Xdebug Simple DBGp client (0.12.0)
Copyright 2002-2018 by Derick Rethans.

Creating a socket for IPv4.

Waiting for debug server to connect on port 9001.
Connect
<?xml version="1.0" encoding="iso-8859-1"?>
<init xmlns="urn:debugger_protocol_v1" xmlns:xdebug="http://xdebug.org/dbgp/xdebug"
fileuri="file:///Users/klambert/workspace/vod2live/vod2live.php" language="PHP"
xdebug:language_version="5.6.37" protocol_version="1.0" appid="43889" idekey="klambert2"><engine version="2.5.5"><! [CDATA[Xdebug]]></engine><author><! [CDATA[Derick Rethans]]></author><url><! [CDATA[http://xdebug.org]]></url><copyright><! [CDATA[Copyright (c) 2002-2017 by Derick Rethans]]></copyright></init>
(cmd) █
```

how to configure a remote server profiler (.ini)

Decide where you want the raw **cachegrind.out.** files

```
[Xdebug]
```

```
...
```

```
xdebug.profiler_enable=1
```

```
xdebug.profiler_output_dir="/usr/local/tmp/xdebug/"
```

```
xdebug.profile_enable_trigger=1
```

```
xdebug.trace_enable_trigger=1
```

how to set up profiler - qcachegrind (opensource)

Again build it yourself (you'll need Qt5 SDK + graphviz)

```
git clone https://github.com/kevleyski/kcachegrind.git
```

```
cd qcachegrind
```

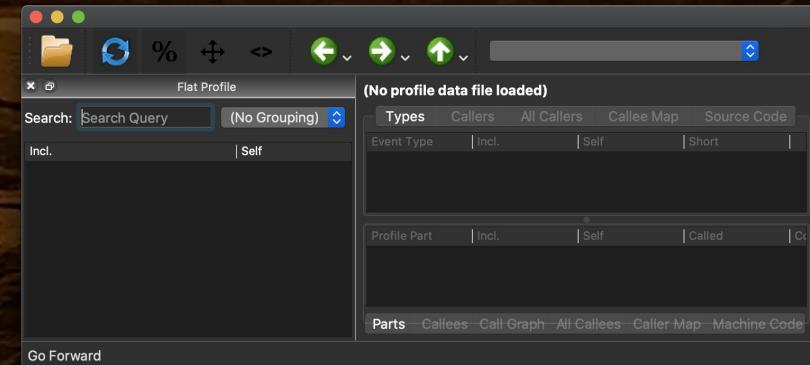
```
qmake -spec 'macx-clang'
```

```
make
```

Note, make sure

`xdebug.profiler_output_dir`

is writable, or this won't work
(you'll also want to scp files)



how to get DBGp commands to your remote

Set debug port same as your remote (confusingly, the default port *9000* is a common FastCGI port - you *probably* don't want that)

```
./debugclient -p 9001
```

```
xdebug Simple DBGp client (0.12.0)
Copyright 2002-2018 by Derick Rethans.
Creating a socket for IPv4.
```

```
Waiting for debug server to connect on port 9001.
```

Connect

debugclient preflight check you're all set to go

Verify your IDE key + remote debug port actually opened

(cmd) **status -i klambert2**

```
<?xml version="1.0" encoding="iso-8859-1"?>

<response xmlns="urn:debugger_protocol_v1"
xmlns:xdebug="http://xdebug.org/dbgp/xdebug"
command="status"
transaction_id="your_ide_key_you_set_earlier"
status="starting" reason="ok"></response>
```

debugclient setting a code breakpoint (line 10)

Set your PHP script running (however you usually do that, e.g. hitting an endpoint `?XDEBUG_SESSION_START=1` or from shell command line

```
XDEBUG_CONFIG="idekey=klambert2" php  
script.php 329 123
```

PHP will pause until you step into it from the debugger

```
(cmd) breakpoint_set -i klambert2 -t line -f  
file:///path/script.php -n 10
```

```
<?xml version="1.0" encoding="iso-8859-1"?>  
<response xmlns="urn:debugger_protocol_v1" xmlns:xdebug="http://xdebug.org/dbgp/xdebug"  
command="breakpoint_set" transaction_id="klambert2" id="477600001"></response>
```

debugclient get property values from the stack

```
(cmd) step_into -i klambert2
```

```
(cmd) property_value -i klambert2 -n ext
```

```
<?xml version="1.0" encoding="iso-8859-1"?>

<response xmlns="urn:debugger_protocol_v1"
xmlns:xdebug="http://xdebug.org/dbgp/xdebug"
command="context_get" transaction_id="klambert2"
context="0">...<property name="$ext" fullname="$ext"
type="uninitialized"></property><property name="$m3u8"
fullname="$m3u8" type="uninitialized"></property></response>
```

wtf? that all seems a bit hard! *Easier alternatives*

As you can see, sending raw DBGP commands to your remote
is not the most inviting debugging experience!

but there is good news! These tools will call DBGP for you

- Vim with Vdebug
- Eclipse IDE (*preferences > php > debug > debuggers*)
- Notepad++ with DBGP Plugin
- Microsoft VSCode with PHP Debug Adapter
- JetBrains PHPStorm, *and others...*

vi Xdebug debugger with vdebug plugin

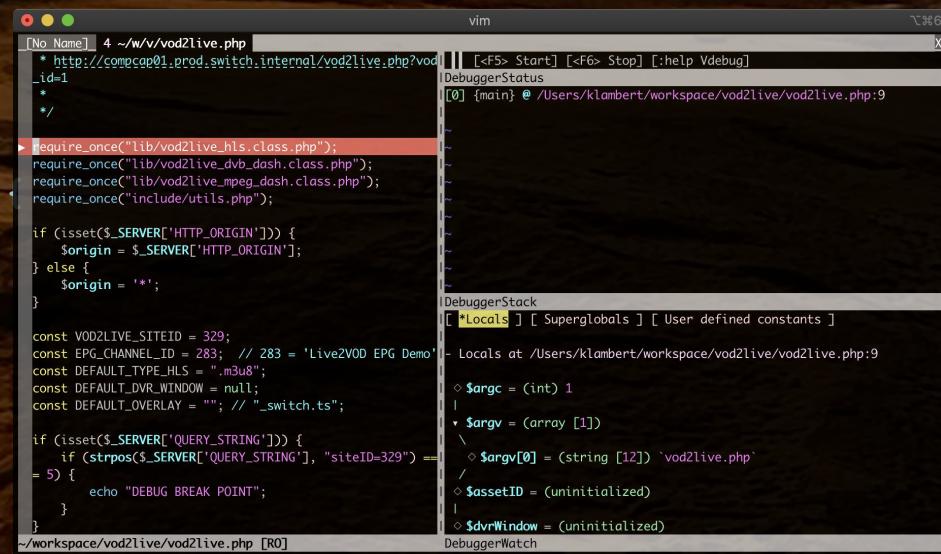
Pre-req you'll need vim built with python3 and pydbgp

```
git clone https://github.com/kevleyski/vdebug  
mv vdebug/* ~/.vim/  
pip3 install komodo-dbgp
```

~/.vim/plugin/vdebug.vim

```
let g:vdebug_options_defaults =  
\\  'port' : 9001,  
\\  'ide_key' : 'klambert2',
```

Connect to debugger F5
(run your PHP, F2 to step)



eclipse Xdebug setup and basic call stack

The screenshot shows the Eclipse IDE interface with the following components:

- Left Sidebar (Preferences):** Shows the "Debuggers" section under the "PHP" category. The "Xdebug" debugger is selected, and its configuration is displayed in the main panel.
- Main Panel (Xdebug Configuration):** Displays the "Connection Settings (Default)" section with fields for "Debug Port: 9001" and "DBGp Proxy". A checkbox for "Use Proxy" is checked, and an "IDE Key: klambert2" is specified. A "Proxy (host:port)" field is also present.
- Code Editor:** Shows two files: "index.php" and "vod2live_hls.class.php". The code in "vod2live_hls.class.php" is highlighted, indicating the current context.
- Call Stack:** A large portion of the screen is occupied by the call stack, which is currently empty.
- Variables View:** Shows a table of variables with their names and values. The variable "\$dvrWindow" is highlighted in blue, indicating it is the current focus.

Name	Value
\$this	VOD2Live_HLS
\$dvrWindow	5
\$ext	...
\$m3u8	## SwitchMedia VOD2
\$makeVOD	false
\$playlistID	"1554693"
\$_COOKIE	Array [0]
\$_ENV	Array [0]
\$_FILES	Array [0]
\$_GET	Array [0]
\$_POST	Array [0]
\$_REQUEST	Array [2]
[siteID]	"329"

- Bottom Right:** A context menu is open over the variable "\$dvrWindow", showing options like Cut, Copy, Paste, Share, Spell, and Subs.

vs code Xdebug setup

The screenshot shows the Visual Studio Code interface with the following components:

- Left Sidebar:** Shows the Extensions view with several recommended extensions listed:
 - PHP Intellisense (2.3.13)
 - PHP Debug (1.13.0) - highlighted with a blue background
 - phpcs (1.0.5)
- Center Panel:** The PHP Debug extension details page is open. It includes:
 - A large "php" logo.
 - The author's name, Felix Becker.
 - A brief description: "Debug support for PHP with XDebug".
 - An "Install" button.
 - A note: "This extension is maintained by the PHP Debug Adapter team.".
 - Links for "Details" and "Contributions".
- Bottom Left:** A preview of the launch.json configuration file, specifically the "Listen for XDebug" target, with the "port": 9001 line circled in red.
- Bottom Right:** The code editor showing a PHP file named "vod2live_hls.class.php". A specific line of code is highlighted with a yellow box:

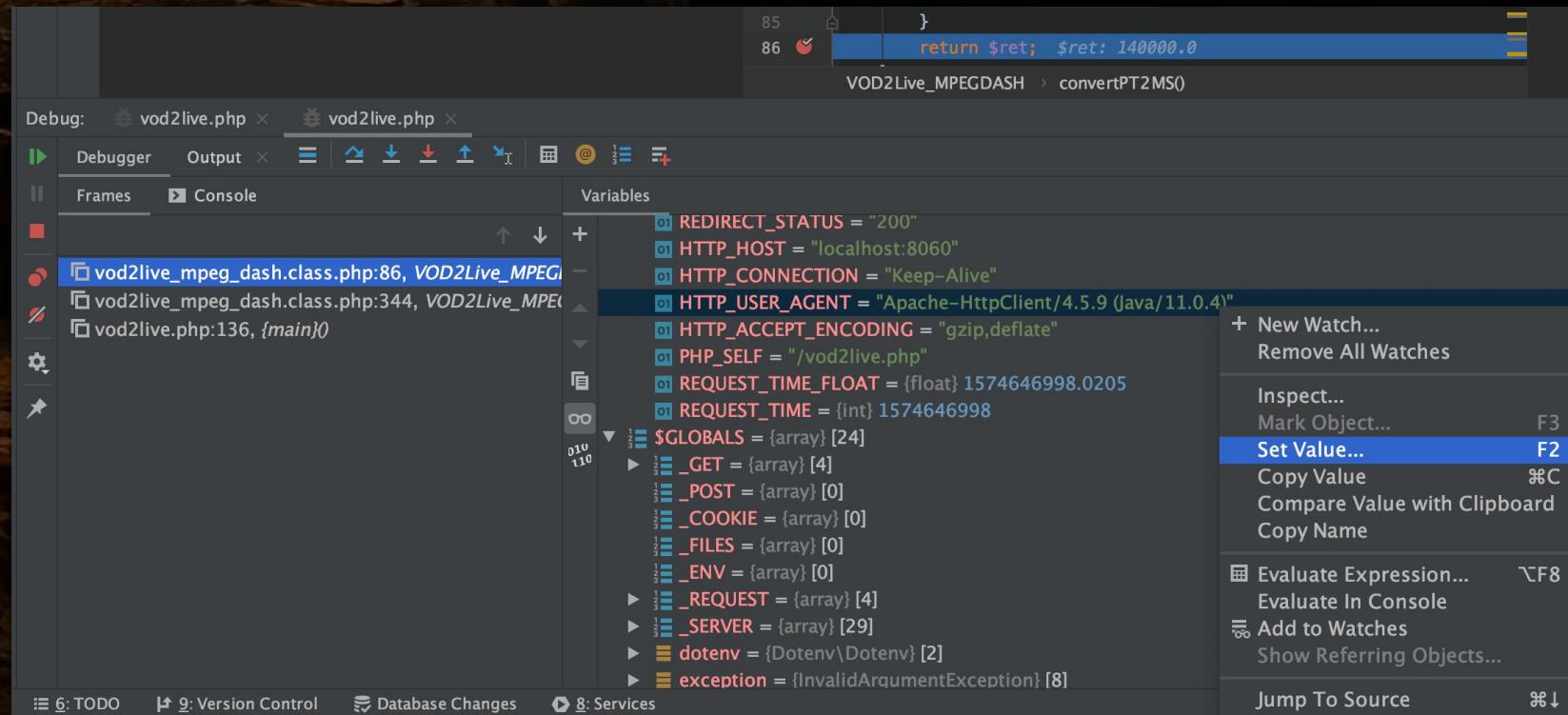
```
if ($self->SUPPORT_X_DEBUG) {  
    $m3u8 .= "## SwitchMedia VOD2Live playlistID=1554693  
}  
else {  
    $m3u8 .= "#EXT-X-VERSION:6\n";  
}
```

PHPStorm Remote server source mapping

The screenshot displays three windows from the PHPStorm IDE:

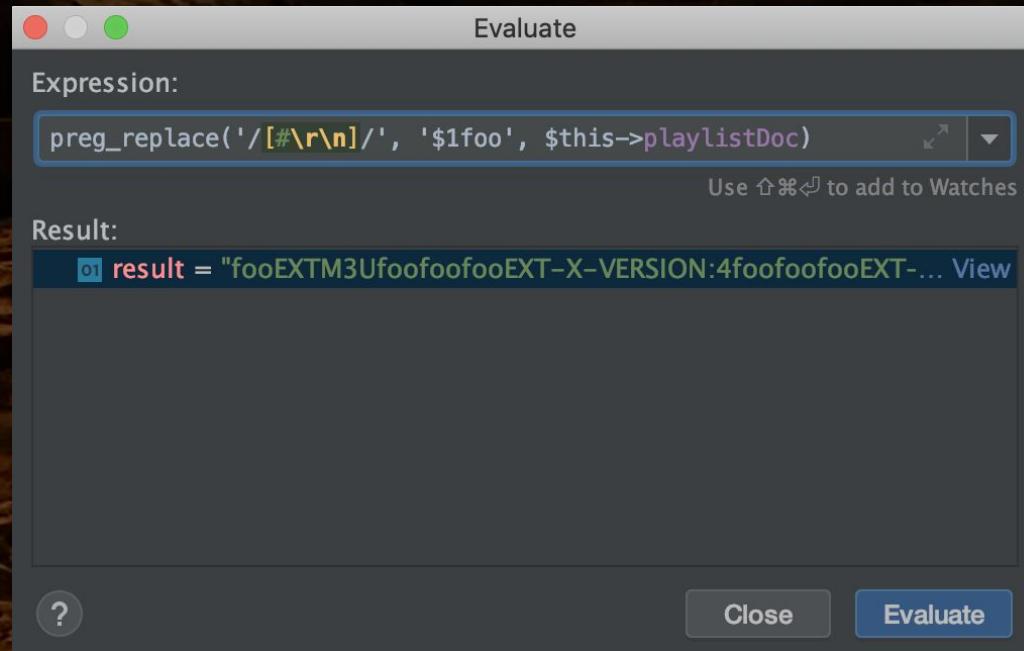
- PHP | Debug**: Shows settings for Xdebug. The "Debug port" is set to **9001**, which is circled in red. Other options include "Ignore external connections through unregistered server config" and "Break at first line in PHP scripts".
- CLI Interpreters**: Shows two interpreters: **PHP 5.6** (selected) and **PHP 7.3**. For PHP 5.6, the "Debugger" is set to **Xdebug 2.5.5**, which is also circled in red.
- Languages & Frameworks | PHP | Deployment**: Shows deployment settings. The "IDE key" is **klambert2**, which is circled in red. The "Connection" tab is selected, showing the "Local path" as **/Users/klambert/workspace/vod2live** and the "Deployment path" as **/vod2live**.

PHPStorm Bloody great debugger! (*but not free*)



PHPStorm Evaluate (super powerful feature)

(re) Eval with real live data (regexp/xpath/sql)



PHPStorm Profiler (source map and quick view)

The screenshot shows the PHPStorm interface with the Profiler tool open. The left sidebar displays the 'HTTP Client' and 'DBGp Proxy' sections, with 'Analyze Xdebug Profiler Snapshot...' selected. Below this, the file tree shows a folder named 'xdebug' containing two files: 'cachegrind.out.10181' and 'cachegrind.out.10215'. The main panel is titled 'Execution Statistics' and 'Call Tree'. It displays two tables: 'Callable' and 'Callees'.

Callable

	Time	Own Time	Calls
/Users/klambert/workspace/vod2live/vod2live.php	544	100.0%	53 9.9%
require_once::/Users/klambert/workspace/vod2live/lib/vod2live_hls.class.php	301	55.3%	47 8.7%
VOD2Live_MPEGDASH->generateLiveMaster	136	25.1%	38 7.1%
require_once::/Users/klambert/workspace/lib/v2/classes/playlist.class.php	136	25.1%	2 0.4%
require_once::/Users/klambert/workspace/lib/v2/classes/playlistItem.class.php	134	24.7%	81 15.0%
require_once::/Users/klambert/workspace/vod2live/lib/vod2live.class.php	84	15.5%	0 0.1%
require::/Users/klambert/workspace/vod2live/vendor/autoload.php	83	15.4%	13 2.6%
ComposerAutoloaderInit8efddde39bf7eb58d4b048a3fbff29bc::getLoader	69	12.8%	9 1.8%
composerRequire8efddde39bf7eb58d4b048a3fbff29bc	59	11.0%	56 10.4%
php::spl_autoload_call	58	10.7%	0 0.1%
Composer\Autoload\ClassLoader->loadClass	57	10.5%	0 0.1%

Callees

	Time	Calls
php::floor	0	0.0%
php::DateInterval->format	0	0.0%
php::hash	0	0.0%
VOD2Live->getCurrentVodID	47	8.6%
VOD2Live->getAssetBuffer	47	8.6%
dbTool->getSingleRowAssoc	8	1.6%
php::mysqli_result->fetch_assoc	0	0.0%
php::mysqli_result->close	0	0.0%
dbTool->doQuery	18	3.4%
php::mysqli->query	18	3.4%
dbTool->__construct	0	0.1%
VOD2Live->setPlaylist	28	5.3%

Thanks!

Slides are here:

<https://tinyurl.com/tcln32s>

DBGp reference:

<https://xdebug.org/docs/dbgp>

Also www.phpinternalsbook.com

php@pyrmontbrewery.com.au

