

# Xdebug tutorial

Dutch PHP Conference - Amsterdam, Netherlands  
Derick Rethans - [derick@derickrethans.nl](mailto:derick@derickrethans.nl) - twitter:  
[@derickr](https://twitter.com/derickr)

<http://derickrethans.nl/talks.php>

<http://joind.in/1530>

## Derick Rethans

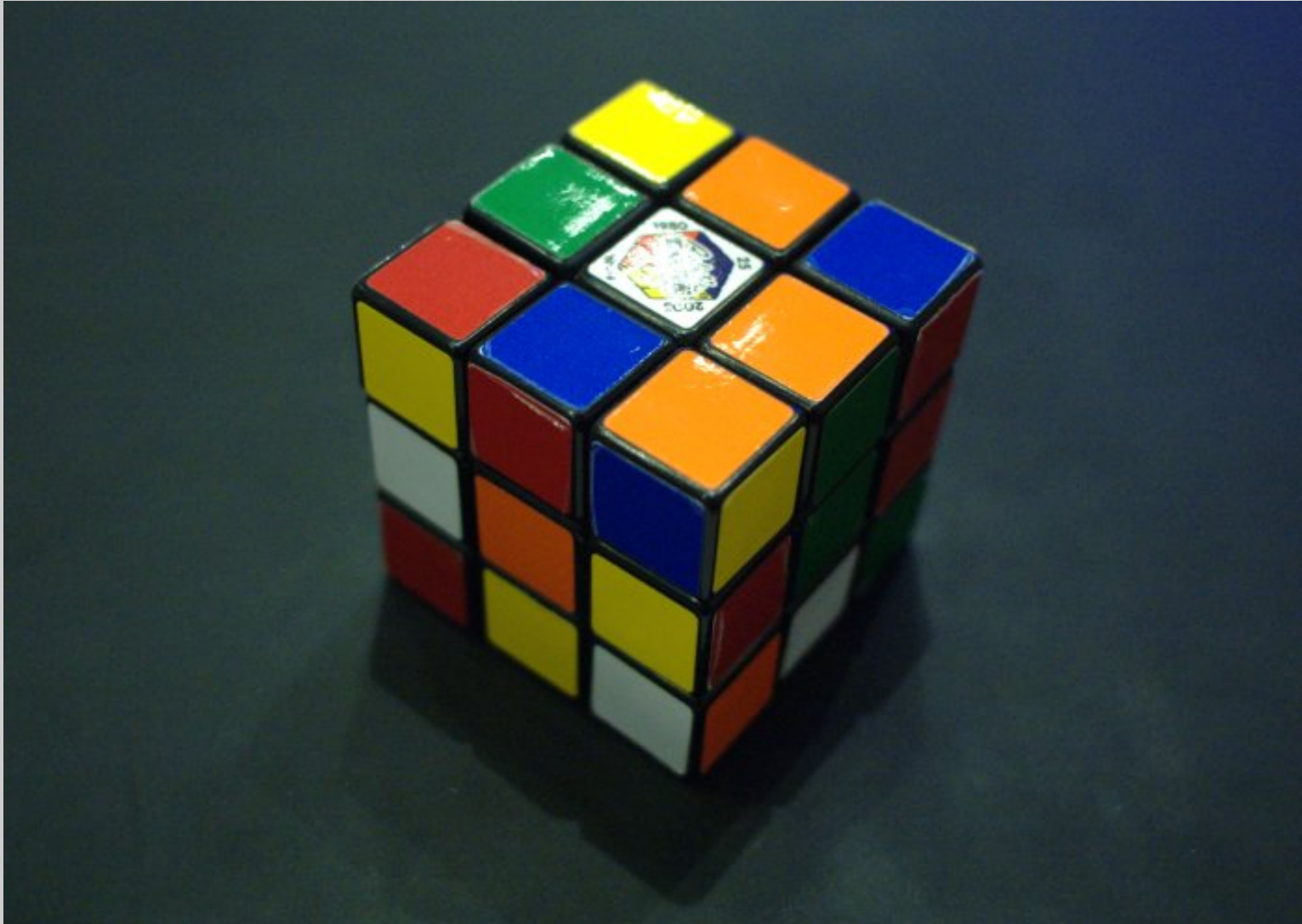


- Dutchman living in London
- PHP development
- Author of the `mcrypt`, `input_filter`, `dbus`, `translit` and `date/time` extensions
- Author of `Xdebug`
- Contributor to the Apache Zeta Components Incubator project (formerly eZ Components)
- Freelancer doing PHP (internals) development

# I Do Not Need a Debugger

- `printf()`, `var_dump()` and `echo` are good enough

# I Do Not Need a Debugger





- Xdebug: An Open Source debugging tool
- About 8 years old
- Works on "every" operating system
- Version 2 released about three years ago

- Installation overview
- Downloading, compiling and configuring Xdebug
- Basic function overview
- Playing with settings, stack traces and function traces

break

- Profiling
- Profiling your own code
- Code coverage
- (Setting up code-coverage with PHP Unit)

break

- Debugging
- Setting up your IDE for debugging

# Installation

# The Xdebug extension

- Zend extension, and not a PHP extension
- Xdebug is very PHP-version sensitive
- Different compilers under Windows
- Debug/non-debug
- Threadsafe or not

## In PHP 5.1 and 5.2:

```
zend_extension=/local/php/lib/php/extensions/no-debug-non-zts-20090626/xdebug.so
zend_extension_ts=/local/php/lib/php/extensions/debug-zts-20090626/xdebug.so
zend_extension_debug=/local/php/lib/php/extensions/debug-non-zts-20090626/xdebug.so
```

## In PHP 5.3:

```
zend_extension=/local/php/lib/php/extensions/no-debug-non-zts-20090626/xdebug.so
zend_extension=/local/php/lib/php/extensions/debug-zts-20090626/xdebug.so
zend_extension=/local/php/lib/php/extensions/debug-non-zts-20090626/xdebug.so
```

# Installing Xdebug

## Compiling

With PEAR:

- `pecl install xdebug-2.1.0RC1`
- Ignore: You should add "extension=xdebug.so" to `php.ini`

Without PEAR:

- `wget http://xdebug.org/files/xdebug-2.1.0RC1.tgz`
- `tar -xvzf xdebug-2.1.0RC1.tgz`
- `cd xdebug-2.1.0RC1`
- `phpize`
- `./configure`
- `make`
- `make install`

Tailored installation instructions:

# Installing Xdebug

## On Windows

- Download the .dll for your PHP version from <http://xdebug.org/download.php>
- in php.ini add:  
`zend_extension_ts=c:\php\xdebug.dll`
- Tips on finding the correct binary:

# Installing Xdebug

## Gotchas

- `--enable-versioning` prevent Xdebug from loading
- Zend's extensions (optimizer, debugger, cache) prohibit Xdebug (and other non-Zend zend-extensions) from loading

play time

# Function Overview

# How did it start?

```
[derick@localhost derick]$ cat recursive.php
<?php
    function a() {
        a();
    }

    a();
?>
[derick@localhost derick]$ php recursive.php
Segmentation fault
```

In PHP 5.3 and later, it will just use up all the available memory.

- Stack overflow in PHP
- Infinite recursion

Xdebug protects, level can be set with:

```
xdebug.max_nesting_level=4
```

Fatal error: Maximum function nesting level of '4' reached, aborting! in /home/httpd/html/test/xdebug/infinite.php on line 11.

# Help By Error Messages

```
Warning: DOMDocument::load() [domdocument.load]: Document is empty in /home/httpd/presentations, line: 1 in /home/httpd/pres2/show2.php on line 165
```

---

```
| ( ! ) Warning: DOMDocument::load() [domdocument.load]: Document is empty |
| in /home/httpd/presentations, line: 1 in /home/httpd/pres2/show2.php on |
| line_165 |
| Call Stack |
| # | Time__ | Memory_ | Function_____ | Location_____ |
| 1 | 0.0022 | _786328 | {main}(_)_____ | ../show2.php:0 |
| 2 | 0.0180 | 1397248 | Presentation->display(_) | ../show2.php:114 |
| 3 | 0.0183 | 1399032 | DOMDocument->load(_)_____ | ../show2.php:165 |
```

```
xdebug.default_enable=1
html_errors=1
```

---

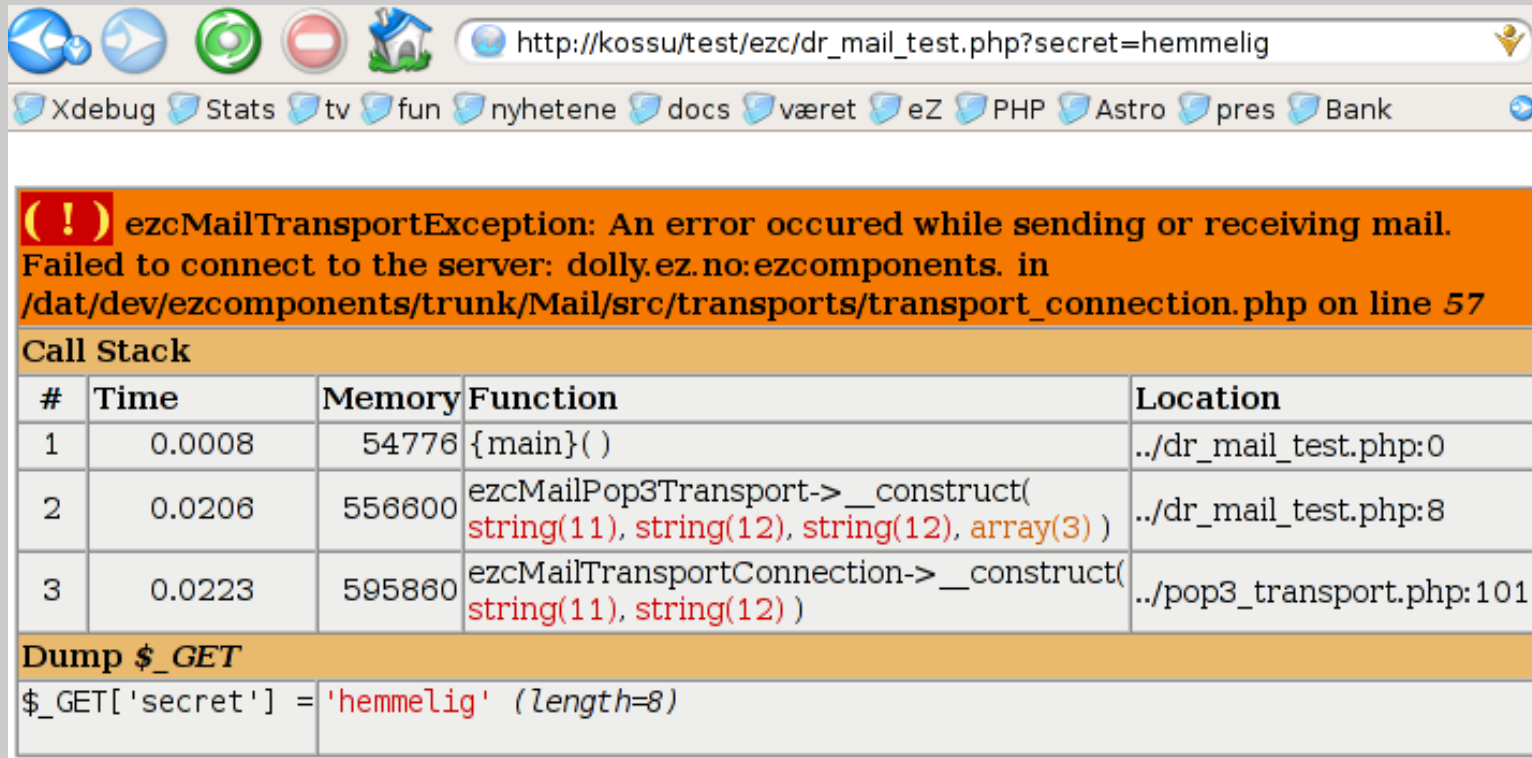
```
| ( ! ) Warning: DOMDocument::load() [domdocument.load]: Document is empty |
| in /home/httpd/presentations, line: 1 in /home/httpd/pres2/show2.php on |
| line_165 |
| Call Stack |
| # | Time__ | Memory_ | Function_____ | Location_____ |
| 1 | 0.0024 | _786328 | {main}(_)_____ | ../show2.php:0 |
| 2 | 0.0183 | 1397248 | Presentation->display(_null_)__ | ../show2.php:114 |
| 3 | 0.0185 | 1399032 | DOMDocument->load(_string(32)_)| ../show2.php:165 |
```

```
xdebug.collect_params=1
```

---

```
| ( ! ) Warning: DOMDocument::load() [domdocument.load]: Document is empty |
| in /home/httpd/presentations, line: 1 in /home/httpd/pres2/show2.php on |
| line_165 |
| Call Stack |
| # | Time__ | Memory_ | Function_____ | Location_____ |
| 1 | 0.0022 | _787720 | {main}(_)_____ | ../show2.php:0 |
| 2 | 0.0183 | 1398552 | Presentation->display(_null_)__ | ../show2.php:114 |
| 3 | 0.0186 | 1400336 | DOMDocument->load(_string(32)_)| ../show2.php:165 |
```

# Debugging With Request Variables



The screenshot shows a web browser window with the address bar containing `http://kossu/test/ezc/dr_mail_test.php?secret=hemmelig`. Below the address bar, there are several navigation icons and a list of bookmarks including Xdebug, Stats, tv, fun, nyhetene, docs, været, eZ, PHP, Astro, pres, and Bank.

An error message is displayed in a red box with a white exclamation mark icon: **ezcMailTransportException: An error occurred while sending or receiving mail. Failed to connect to the server: dolly.ez.no:ezcomponents. in /dat/dev/ezcomponents/trunk/Mail/src/transport/transport\_connection.php on line 57**

Below the error message, a "Call Stack" table is shown:

#	Time	Memory	Function	Location
1	0.0008	54776	{main}()	../dr_mail_test.php:0
2	0.0206	556600	ezcMailPop3Transport->__construct( string(11), string(12), string(12), array(3) )	../dr_mail_test.php:8
3	0.0223	595860	ezcMailTransportConnection->__construct( string(11), string(12) )	../pop3_transport.php:101

Below the call stack, a "Dump \$\_GET" section is shown:

```
$_GET['secret'] = 'hemmelig' (length=8)
```

`xdebug.dump.GET=*`

`xdebug.dump.POST=username,password`

Also for: COOKIE, ENV, FILES, REQUEST, SERVER  
and SESSION



# How Many Functions

```
<?php  
echo xdebug_get_function_count();  
?>
```

```
<?php  
echo xdebug_get_function_count();  
?>
```

# Scream

- PHP's @ operator hides warnings and errors
- `xdebug.scream=1` makes PHP ignore @

# Function trace

```
TRACE START [2010-03-24 11:03:12]
  0.0022      787776    -> {main}() /home/httpd/pres2/show2.php:0
  0.0028      803160
-> require(/home/derick/dev/ezcomponents/trunk/Base/
src/ezc_bootstrap.php) /home/httpd/pres2/show2.php:2
  0.0029      803552
-> dirname('/home/derick/dev/ezcomponents/trunk/
Base/src/ezc_bootstrap.php') /home/derick/dev/ezcomponents/trunk/Base/
src/
ezc_bootstrap.php:12
  0.0030      803968    -> explode('/',
'/home/derick/dev/ezcomponents/
trunk/Base/src') /home/derick/dev/ezcomponents/trunk/Base/src/ezc_boot
strap.php:
13
  0.0031      807352    -> count(array (0 => '', 1 => 'home', 2
=>
'derick', 3 => 'dev', 4 => 'ezcomponents', 5 => 'trunk', 6 => 'Base',
7 =>
'src')) /home/derick/dev/ezcomponents/trunk/Base/src/ezc_bootstrap.php
:15
  0.0032      807800    -> array_slice(array (0 => '', 1 =>
'home', 2 =>
'derick', 3 => 'dev', 4 => 'ezcomponents', 5 => 'trunk', 6 => 'Base',
7 =>
'src'), 0,
-2) /home/derick/dev/ezcomponents/trunk/Base/src/ezc_bootstrap.php:17
  0.0033      808952    -> join('/', array (0 => '', 1 => 'home',
```

# Vim Syntax

## ~/.vim/filetype.vim

```
augroup filetypedetect
au BufNewFile,BufRead *.xt setf xt
augroup END
```

## ~/.vim/syntax/xt.vim (<http://xdebug.org/files/xt.vim>)

```
if version < 600
syntax clear
elseif exists("b:current_syntax")
finish
endif

syn match begin          "^TRACE START"
syn match end            "^TRACE END"
syn match date           "\[.*\]"

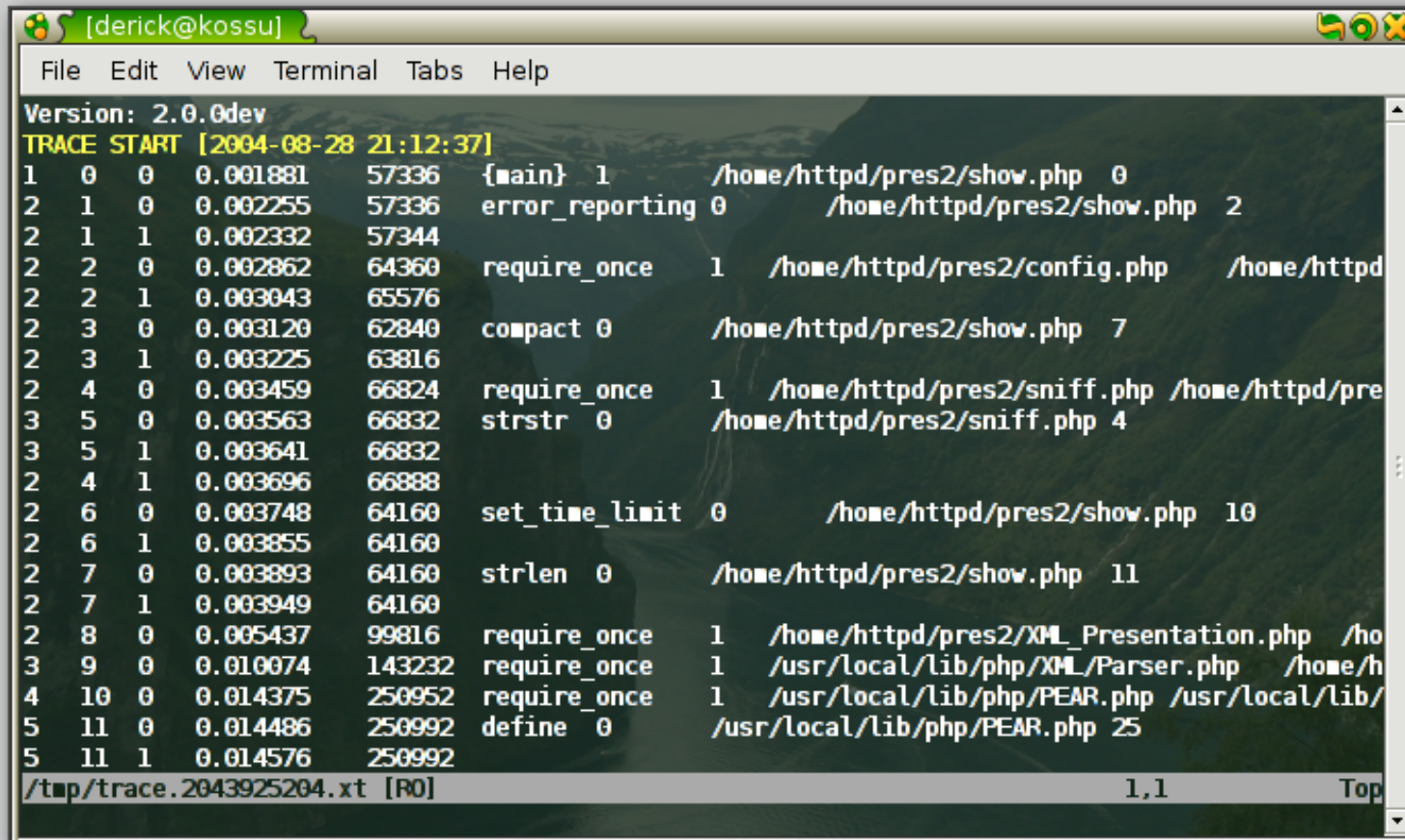
syn match min_memory    "+\d\+"
syn match pls_memory    "-\d\+"
syn match nll_memory    "+0"

syn match level         "->"
syn match lineno        ":\d\+$"

syn match methodcall    "\k\+-->"
syn match staticcall    "\k\+::"
syn match functionb     "\k\+("
syn match functione     ") "
```

# Function trace to file

## Automatic readable format



```
[derick@kossu]
File Edit View Terminal Tabs Help
Version: 2.0.0dev
TRACE START [2004-08-28 21:12:37]
1 0 0 0.001881 57336 {main} 1 /home/httpd/pres2/show.php 0
2 1 0 0.002255 57336 error_reporting 0 /home/httpd/pres2/show.php 2
2 1 1 0.002332 57344
2 2 0 0.002862 64360 require_once 1 /home/httpd/pres2/config.php /home/httpd
2 2 1 0.003043 65576
2 3 0 0.003120 62840 compact 0 /home/httpd/pres2/show.php 7
2 3 1 0.003225 63816
2 4 0 0.003459 66824 require_once 1 /home/httpd/pres2/sniff.php /home/httpd/pre
3 5 0 0.003563 66832 strstr 0 /home/httpd/pres2/sniff.php 4
3 5 1 0.003641 66832
2 4 1 0.003696 66888
2 6 0 0.003748 64160 set_time_limit 0 /home/httpd/pres2/show.php 10
2 6 1 0.003855 64160
2 7 0 0.003893 64160 strlen 0 /home/httpd/pres2/show.php 11
2 7 1 0.003949 64160
2 8 0 0.005437 99816 require_once 1 /home/httpd/pres2/XML_Presentation.php /ho
3 9 0 0.010074 143232 require_once 1 /usr/local/lib/php/XML/Parser.php /home/h
4 10 0 0.014375 250952 require_once 1 /usr/local/lib/php/PEAR.php /usr/local/lib/
5 11 0 0.014486 250992 define 0 /usr/local/lib/php/PEAR.php 25
5 11 1 0.014576 250992
/tmp/trace.2043925204.xt [R0] 1,1 Top
```

```
xdebug.auto_trace=1 ; enable tracing
xdebug.trace_format=1 ; selects computerized format
xdebug.trace_options=0 ; sets extra option (1 = append)
```

# Function trace

## Other functionality

- HTML traces
- Tracing only parts of an application with `xdebug_start_trace()` and `xdebug_stop_trace()`.
- Fetching the trace file name that is being used with `xdebug_get_tracefile_name()`.
- Changing how much data is shown with `xdebug.var_display_max_children`, `xdebug.var_display_max_data` and `xdebug.var_display_max_depth`.

# Practical

## Basic functions and Function traces

- `xdebug.default_enable / html_errors`
- `xdebug.max_nesting_level`
- `xdebug.file_link_format=gvim://%f@%l`
- `xdebug.collect_includes / xdebug.collect_params /  
xdebug.collect_return /  
xdebug.collect_assignments`
- `xdebug.var_display_max_data /  
xdebug.var_display_max_depth /  
xdebug.var_display_max_children`
- `xdebug.dump.*`
- `xdebug.show_local_vars`
- `xdebug.auto_trace`
- `xdebug.trace_output_dir /  
xdebug.trace_output_name`
- `xdebug.trace_output_format`

# Profiling

- Uses the "computerized" trace files
- A script is provided in the Xdebug source dir: contrib/tracefile-analyser.php:

```
php tracefile-analyser.php /tmp/trace.124683842.xt [sortkey] [element count]
```

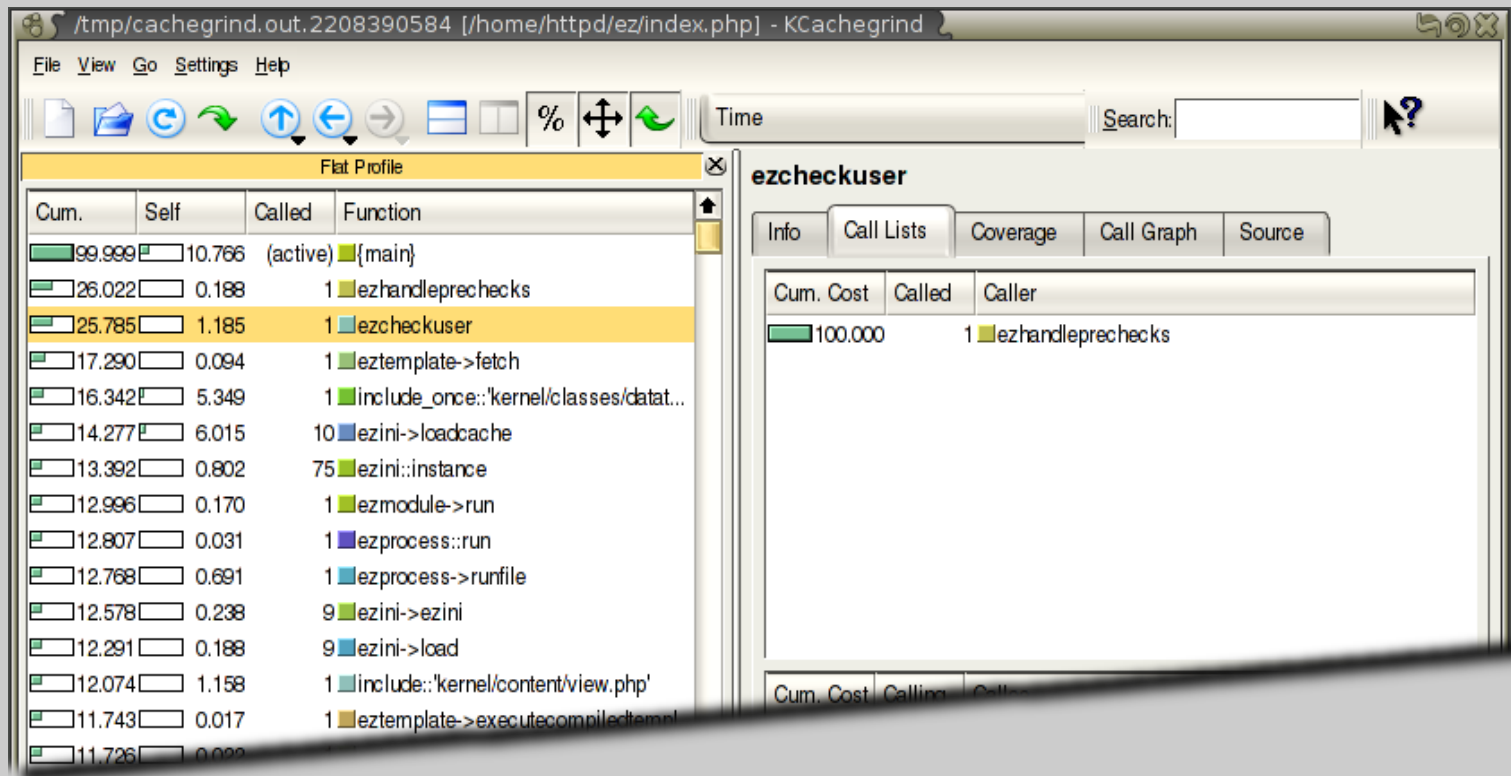
- sort keys: calls, time-inclusive, memory-inclusive, time-own, memory-own

Showing the 10 most costly calls sorted by 'time-inclusive'.

function	#calls	Inclusive time	memory	Own time	memory
{main}	1	0.0894	1025656	0.0006	5672
Presentation->display	1	0.0723	408848	0.0004	-11528
ezcTemplate->process	10	0.0685	243320	0.0005	-2976
ezcTemplateCompiledCode->execute	10	0.0627	14144	0.0009	1672
include	10	0.0612	12352	0.0009	-4392
__autoload	10	0.0176	781904	0.0006	1080
ezcBase::autoload	10	0.0170	780824	0.0017	-80728
ezcTemplateCompiledCode::findCompiled	10	0.0094	38776	0.0024	12888
ezcTemplateCompiledCode->checkRequirements	10	0.0094	2968	0.0029	2968
ezcBase::loadFile	10	0.0093	642320	0.0070	625520

# Profiling

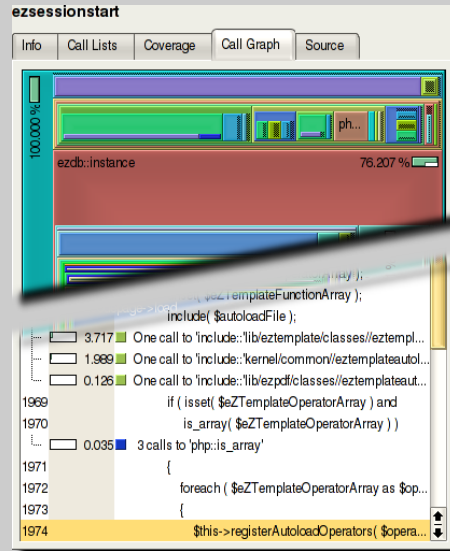
## KCacheGrind's Flat Profile and Call List



```
xdebug.profiler_enable=1 ; enable profiler
xdebug.profiler_output_dir=/tmp ; output directory
xdebug.profiler_output_name=cachegrind.out.%p
```

# Profiling

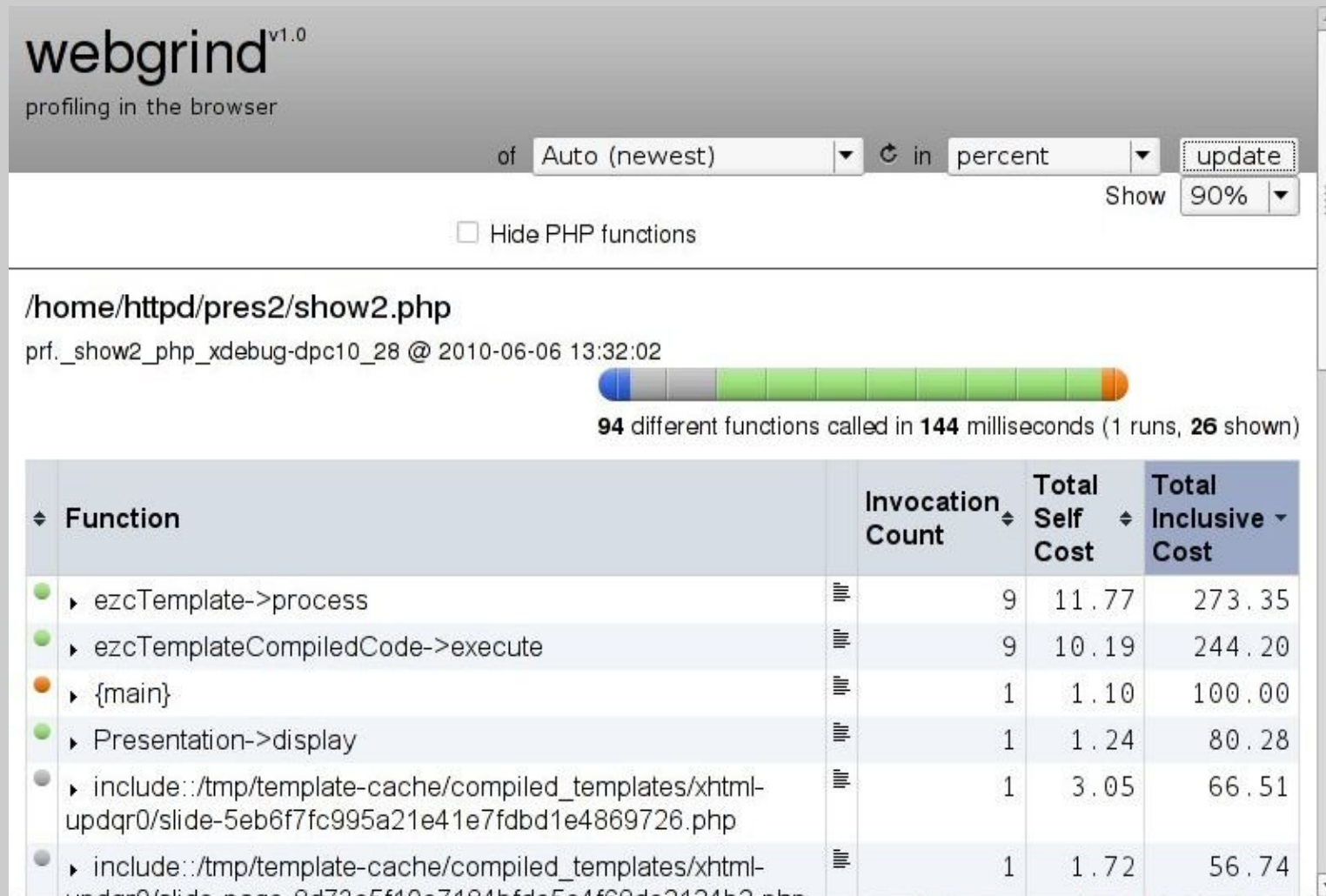
## KCacheGrind's Call Graph and Source Annotations



- Call graph
- Area shows time spend
- Stacked to show callees
- Source annotations
- Number of calls
- Total time per function

demo

- Easy to install: extract in a directory in the HTTP root and run



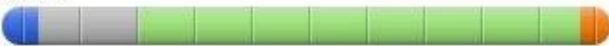
webgrind<sup>v1.0</sup>  
profiling in the browser

of  in

Show

Hide PHP functions

**/home/httpd/pres2/show2.php**  
prf.\_show2\_php\_xdebug-dpc10\_28 @ 2010-06-06 13:32:02



94 different functions called in 144 milliseconds (1 runs, 26 shown)

Function	Invocation Count	Total Self Cost	Total Inclusive Cost
▶ ezcTemplate->process	9	11.77	273.35
▶ ezcTemplateCompiledCode->execute	9	10.19	244.20
▶ {main}	1	1.10	100.00
▶ Presentation->display	1	1.24	80.28
▶ include::/tmp/template-cache/compiled_templates/xhtmll-updqr0/slide-5eb6f7fc995a21e41e7fdbd1e4869726.php	1	3.05	66.51
▶ include::/tmp/template-cache/compiled_templates/xhtmll-updqr0/slide-page-8d72e5f10e7194bfde5e4f69de2124b2.php	1	1.72	56.74

### Settings:

- `xdebug.profiler_enable / xdebug.profiler_enable_trigger`

### Tools:

- KCacheGrind (Linux)
- WinCacheGrind (Windows)
- maccallgrind (Mac, non-free)
- webgrind (PHP script, multi-platform)

### Browser extensions:

easy Xdebug:

Xdebug Helper:

# What Code Do I Use

```
34     public function __construct()  
35     {  
36         $args = func_get_args();  
37         parent::__construct( array() );  
38         foreach ( $args as $part )  
39         {  
40             if ( $part instanceof ezcMailPart )  
41             {  
42                 $this->parts[] = $part;  
43             }  
44             elseif( is_array( $part ) ) // add each  
45             {  
46                 foreach ( $part as $array_part )  
47                 {
```

## Available functions:

```
xdebug_start_code_coverage();  
xdebug_get_code_coverage();  
xdebug_stop_code_coverage();
```

### Functions for obtaining coverage information:

```
xdebug_start_code_coverage();  
xdebug_get_code_coverage();  
xdebug_stop_code_coverage();
```

Options to `xdebug_start_code_coverage()`, both only work for files parsed from this moment:

- `XDEBUG_CC_UNUSED`: Enables scanning of code to figure out which line has executable code.
- `XDEBUG_CC_DEAD_CODE`: Enables branch analyzation to figure out whether code can be executed.
- <http://kossu/coverage/index.html>

play time

# Debugging

# Analyzing Running Scripts

- DBGp, common Debugging protocol
- Cross-language: PHP, Python, Perl...
- Supported in Xdebug 2

## Clients:

- Komodo (Linux, Windows, Mac — commercial)
- Eclipse/PDT (Java based — free) L
- Zend Studio for Eclipse (Eclipse-based — commercial)
- Netbeans (Java-based — free) LW
- PHPStorm (Java-based — commercial) LW
- (and many others: )

# Activating the Remote Debugger

## php.ini settings:

```
xdebug.remote_enable=1  
xdebug.remote_host=localhost  
xdebug.remote_port=9000
```

## On the shell:

```
export XDEBUG_CONFIG="idekey=xdebugrocks"
```

## With a browser:

```
http://pres/show.php?
```

```
XDEBUG_SESSION_START=xdebugrocks
```

## With browser extensions:

easy Xdebug (FireFox):

Xdebug Helper (Chrome):

# Debugging Gotchas

- Xdebug connects to the client
- Remote vs Local debugging
- Debugging proxy: (Komodo-PythonRemoteDebugging)
- `xdebug.remote_connect_back`

- Input commands as "command line parameters"
- `property_get -n $foo -d 2 -i 81`
- Output data as documented XML:

```
data_length
```

```
[NULL]
```

```
<response command="command_name" transaction_id="transaction_id"/>
```

```
[NULL]
```

- Cross-language: PHP, Python, Perl...
- Supported in Xdebug 2

demo

### Settings:

- `xdebug.remote_enable`
- `xdebug.remote_host` / `xdebug.remote_port`
- `xdebug.remote_autostart`
- `xdebug.remote_log`

### Browser extensions:

- easy Xdebug:
- Xdebug Helper:



- Xdebug site: <http://xdebug.org>
- Xdebug documentation:  
<http://xdebug.org/docs.php>
- DBGp specification: <http://xdebug.org/docs-dbgp.php>
- If you like Xdebug: <http://xdebug.org/donate.php>
- These slides: <http://dorickrothmans.nl/talks.php>