Advanced PHP debugger

Introduction

APD is the Advanced PHP Debugger. It was written to provide profiling and debugging capabilities for PHP code, as well as to provide the ability to print out a full stack backtrace. APD supports interactive debugging, but by default it writes data to trace files. It also offers event based logging so that varying levels of information (including function calls, arguments passed, timings, etc.) can be turned on or off for individual scripts.

Caution
APD is a Zend Extension, modifying the way the internals of PHP handle function calls, and thus may or may not be compatible with other Zend Extensions (for example Zend Optimizer).

Installing/Configuring

Requirements

No external libraries are needed to build this extension.

Installation

APD is currently available as a PECL extension from » http://pecl.php.net/package/apd. Make sure you have installed the CGI version of PHP and it is available in your current path along with the phpize script.

Run the following command to download, build, and install the latest stable version of APD:

```
pear install apd
```

This automatically installs the APD Zend module into your PHP extensions directory. It is not mandatory to keep it there; you can store the module in any directory PHP can read as long as you set the zend_extension parameter accordingly.

Windows users will enable *php_apd.dll* inside of *php.ini* in order to use these functions. The DLL for this PECL extension may be downloaded from either the » PHP Downloads page or from » http://pecl4win.php.net/

In your INI file, add the following lines:

```
zend_extension = /absolute/path/to/apd.so
apd.dumpdir = /absolute/path/to/trace/directory
apd.statement_tracing = 0
```

Depending on your PHP build, the zend_extension directive can be one of the following:

```
zend_extension              (non ZTS, non debug build)
zend_extension_ts           (    ZTS, non debug build)
zend_extension_debug        (non ZTS,     debug build)
zend_extension_debug_ts     (    ZTS,     debug build)
```

Building on Win32

To build APD under Windows you need a working PHP compilation environment as described on http://php.net/ -- basically, it requires you to have Microsoft Visual C++, win32build.zip, bison/flex, and some know how to get it to work. Also ensure that adp.dsp

has DOS line endings; if it has unix line endings, Microsoft Visual C++ will complain about it.

Runtime Configuration

The behaviour of these functions is affected by settings in *php.ini*.

APD Configuration Options

Name	Default	Changeable	Changelog
apd.dumpdir	NULL	PHP_INI_ALL	
apd.statement_tracing	"0"	PHP_INI_ALL	Available since apd 0.9.

For further details and definitions of the PHP_INI_* constants, see the php.ini directives.

Here's a short explanation of the configuration directives.

apd.dumpdir string
: Sets the directory in which APD writes profile dump files. You can specify an absolute path or a relative path. You can specify a different directory as an argument to apd_set_pprof_trace().

apd.statement_tracing boolean
: Specfies whether or not to do per-line tracings. Turning this on (1) will impact the performance of your application.

Resource Types

This extension has no resource types defined.

Predefined Constants

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

APD constants

Constant	Value	Description
FUNCTION_TRACE (integer)	1	
ARGS_TRACE (integer)	2	
ASSIGNMENT_TRACE (integer)	4	
STATEMENT_TRACE (integer)	8	
MEMORY_TRACE (integer)	16	
TIMING_TRACE (integer)	32	
SUMMARY_TRACE (integer)	64	
ERROR_TRACE (integer)	128	
PROF_TRACE (integer)	256	
APD_VERSION (string)	example: *1.0.2-dev*	

Examples

How to use PHP-APD in your scripts

1. As the first line of your PHP script, call the apd_set_pprof_trace() function to start the trace:
   ```
   apd_set_pprof_trace();
   ```
   You can insert the line anywhere in your script, but if you do not start tracing at the beginning of your script you discard profile data that might otherwise lead you to a performance bottleneck.
2. Now run your script. The dump output will be written to *apd.dumpdir/pprof_pid.ext*.

Tip
If you're running the CGI version of PHP, you will need to add the '-e' flag to enable extended information for apd to work properly. For example: **php -e -f script.php**

3. To display formatted profile data, issue the *pprofp* command with the sort and display options of your choice. The formatted output will look something like:
   ```
   bash-2.05b$ pprofp -R /tmp/pprof.22141.0

   Trace for /home/dan/testapd.php
   Total Elapsed Time = 0.00
   Total System Time  = 0.00
   Total User Time    = 0.00


   Real         User        System             secs/    cumm
   %Time (excl/cumm)  (excl/cumm)  (excl/cumm) Calls    call    s/call  Memory
   Usage Name
   --------------------------------------------------------------------------
   ----------
   100.0 0.00 0.00  0.00 0.00  0.00 0.00     1  0.0000   0.0009            0
   main
   56.9 0.00 0.00  0.00 0.00  0.00 0.00     1  0.0005   0.0005            0
   apd_set_pprof_trace
   28.0 0.00 0.00  0.00 0.00  0.00 0.00    10  0.0000   0.0000            0
   preg_replace
   14.3 0.00 0.00  0.00 0.00  0.00 0.00    10  0.0000   0.0000            0
   str_replace
   ```
   The -R option used in this example sorts the profile table by the amount of real time the script spent executing a given function. The "cumm call" column reveals how many times each function was called, and the "s/call" column reveals how many seconds each call to the function required, on average.
4. To generate a calltree file that you can import into the KCacheGrind profile analysis application, issue the *pprof2calltree* comand.

APD Functions

Contact information

If you have comments, bugfixes, enhancements or want to help developing this beast, you can send an mail to » apd@mail.communityconnect.com. Any help is very welcome.

apd_breakpoint

apd_breakpoint -- Stops the interpreter and waits on a CR from the socket

Description

bool **apd_breakpoint** (int `$debug_level`)

This can be used to stop the running of your script, and await responses on the connected socket. To step the program, just send enter (a blank line), or enter a php command to be executed.

Parameters

`debug_level`
: An integer which is formed by adding together the *XXX_TRACE* constants. It is not recommended to use **MEMORY_TRACE**. It is very slow and does not appear to be accurate. **ASSIGNMENT_TRACE** is not implemented yet. To turn on all functional traces (TIMING, FUNCTIONS, ARGS SUMMARY (like strace -c)) use the value 99

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #1 - Typical session using tcplisten

```
bash#tcplisten localhost 7777

APD - Advanced PHP Debugger Trace File
-------------------------------------------------------------------------
Process Pid (6118)
Trace Begun at Sun Mar 10 23:13:12 2002
-------------------------------------------------------------------------
(  0.000000): apd_set_session_trace called at
/home/alan/Projects/project2/test.
php:5
(  0.074824): apd_set_session_trace_socket() at
/home/alan/Projects/project2/tes
t.php:5 returned.  Elapsed (0.074824)
(  0.074918): apd_breakpoint() /home/alan/Projects/project2/test.php:7
            ++ argv[0] $(??) = 9
apd_breakpoint() at /home/alan/Projects/project2/test.php:7 returned.
Elapsed (
-2089521468.1073275368)
>\n
```

```
statement: /home/alan/Projects/project2/test.php:8
>\n
statement: /home/alan/Projects/project2/test.php:8
>\n
statement: /home/alan/Projects/project2/test.php:10
>apd_echo($i);
EXEC: apd_echo($i);
0
>apd_echo(serialize(apd_get_active_symbols()));
EXEC:  apd_echo(serialize(apd_get_active_symbols()));
a:47:{i:0;s:4:"PWD";i:1;s:10:"COLORFGBG";i:2;s:11:"XAUTHORITY";i:3;s:14:"
COLORTERM_BCE";i:4;s:9:"WINDOWID";i:5;s:14:"ETERM_VERSION";i:6;s:16:"SE
SSION_MANAGER";i:7;s:4:"PS1";i:8;s:11:"GDMSESSION";i:9;s:5:"USER";i:10;s:5:"
MAIL";i:11;s:7:"OLDPWD";i:12;s:5:"LANG";i:13;s:10:"COLORTERM";i:14;s:8:"DISP
LAY";i:15;s:8:"LOGNAME";i:16;s:6:"
>apd_echo(system('ls /home/mydir'));
........
>apd_continue(0);
```

apd_callstack

apd_callstack -- Returns the current call stack as an array

Description

array **apd_callstack** (void)

Returns the current call stack as an array

Return Values

An array containing the current call stack.

Examples

Example #2 - apd_callstack() example

```
<?php
print_r(apd_callstack());
?>
```

apd_clunk

apd_clunk -- Throw a warning and a callstack

Description

void **apd_clunk** (string `$warning` [, string `$delimiter`])

Behaves like perl's Carp::cluck. Throw a warning and a callstack.

Parameters

`warning`
The warning to throw.

`delimiter`
The delimiter. Default to <*BR /*>.

Return Values

No value is returned.

Examples

Example #3 - apd_clunk() example

```
<?php
apd_clunk("Some Warning", "<br/>");
?>
```

See Also

- apd_croak()

apd_continue

apd_continue -- Restarts the interpreter

Description

bool **apd_continue** (int `$debug_level`)

Usually sent via the socket to restart the interpreter.

Parameters

`debug_level`
: An integer which is formed by adding together the *XXX_TRACE* constants. It is not recommended to use **MEMORY_TRACE**. It is very slow and does not appear to be accurate. **ASSIGNMENT_TRACE** is not implemented yet. To turn on all functional traces (TIMING, FUNCTIONS, ARGS SUMMARY (like strace -c)) use the value 99

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #4 - apd_continue() example

```
<?php
apd_continue(0);
?>
```

apd_croak

apd_croak -- Throw an error, a callstack and then exit

Description

void **apd_croak** (string `$warning` [, string `$delimiter`])

Behaves like perl's Carp::croak. Throw an error, a callstack and then exit.

Parameters

`warning`
The warning to throw.

`delimiter`
The delimiter. Default to <*BR* />.

Return Values

No value is returned.

Examples

Example #5 - apd_croak() example

```
<?php
apd_croak("Some Warning","<P>");
?>
```

See Also

- apd_clunk()

apd_dump_function_table

apd_dump_function_table -- Outputs the current function table

Description

void **apd_dump_function_table** (void)

Outputs the current function table.

Return Values

No value is returned.

Examples

Example #6 - apd_dump_function_table() example

```
<?php
apd_dump_function_table();
?>
```

apd_dump_persistent_resources

apd_dump_persistent_resources -- Return all persistent resources as an array

Description

array **apd_dump_persistent_resources** (void)

Return all persistent resources as an array.

Return Values

An array containing the current call stack.

Examples

Example #7 - apd_dump_persistent_resources() **example**

```
<?php
print_r(apd_dump_persistent_resources());
?>
```

See Also

- **apd_dump_regular()**

apd_dump_regular_resources

apd_dump_regular_resources -- Return all current regular resources as an array

Description

array **apd_dump_regular_resources** (void)

Return all current regular resources as an array.

Return Values

An array containing the current regular resources.

Examples

Example #8 - apd_dump_regular_resources() **example**

```
<?php
print_r(apd_dump_regular_resources());
?>
```

See Also

- apd_dump_persistent_resources()

apd_echo

apd_echo -- Echo to the debugging socket

Description

bool **apd_echo** (string `$output`)

Usually sent via the socket to request information about the running script.

Parameters

`output`
The debugged variable.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #9 - apd_echo() **example**

```
<?php
apd_echo($i);
?>
```

apd_get_active_symbols

apd_get_active_symbols -- Get an array of the current variables names in the local scope

Description

array **apd_get_active_symbols** (void)

Returns the names of all the variables defined in the active scope, (not their values).

Return Values

A multidimensional array with all the variables.

Examples

Example #10 - apd_get_active_symbols() **example**

```
<?php
apd_echo(apd_get_active_symbols());
?>
```

apd_set_pprof_trace

apd_set_pprof_trace -- Starts the session debugging

Description

string **apd_set_pprof_trace** ([string `$dump_directory` [, string `$fragment`]])

Starts debugging to *pprof_{process_id}* in the dump directory.

Parameters

`dump_directory`
: The directory in which the profile dump file is written. If not set, the apd.dumpdir setting from the *php.ini* file is used.

`fragment`

Return Values

Returns path of the destination file.

Examples

Example #11 - apd_set_pprof_trace() example

```
<?php
apd_set_pprof_trace();
?>
```

See Also

- apd_set_session_trace()

apd_set_session_trace

apd_set_session_trace -- Starts the session debugging

Description

void **apd_set_session_trace** (int `$debug_level` [, string `$dump_directory`])

Starts debugging to *apd_dump_{process_id}* in the dump directory.

Parameters

`debug_level`
: An integer which is formed by adding together the *XXX_TRACE* constants. It is not recommended to use **MEMORY_TRACE**. It is very slow and does not appear to be accurate. **ASSIGNMENT_TRACE** is not implemented yet. To turn on all functional traces (TIMING, FUNCTIONS, ARGS SUMMARY (like strace -c)) use the value 99

`dump_directory`
: The directory in which the profile dump file is written. If not set, the apd.dumpdir setting from the *php.ini* file is used.

Return Values

No value is returned.

Examples

Example #12 - apd_set_session_trace() example

```
<?php
apd_set_session_trace(99);
?>
```

apd_set_session

apd_set_session -- Changes or sets the current debugging level

Description

void **apd_set_session** (int `$debug_level`)

This can be used to increase or decrease debugging in a different area of your application.

Parameters

`debug_level`
: An integer which is formed by adding together the *XXX_TRACE* constants. It is not recommended to use **MEMORY_TRACE**. It is very slow and does not appear to be accurate. **ASSIGNMENT_TRACE** is not implemented yet. To turn on all functional traces (TIMING, FUNCTIONS, ARGS SUMMARY (like strace -c)) use the value 99

Return Values

No value is returned.

Examples

Example #13 - apd_set_session() example

```
<?php
apd_set_session(9);
?>
```

apd_set_socket_session_trace

apd_set_socket_session_trace -- Starts the remote session debugging

Description

bool **apd_set_socket_session_trace** (string `$tcp_server`, int `$socket_type`, int `$port`, int `$debug_level`)

Connects to the specified *`tcp_server`* (eg. tcplisten) and sends debugging data to the socket.

Parameters

`tcp_server`
IP or Unix Domain socket (like a file) of the TCP server.

`socket_type`
Can be **AF_UNIX** for file based sockets or **APD_AF_INET** for standard tcp/ip.

`port`
You can use any port, but higher numbers are better as most of the lower numbers may be used by other system services.

`debug_level`
An integer which is formed by adding together the *XXX_TRACE* constants. It is not recommended to use **MEMORY_TRACE**. It is very slow and does not appear to be accurate. **ASSIGNMENT_TRACE** is not implemented yet. To turn on all functional traces (TIMING, FUNCTIONS, ARGS SUMMARY (like strace -c)) use the value 99

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #14 - apd_set_socket_session_trace() **example**

```
<?php
 apd_set_socket_session_trace("127.0.0.1",APD_AF_INET,7112,0);
?>
```

override_function

override_function -- Overrides built-in functions

Description

bool **override_function** (string `$function_name`, string `$function_args`, string `$function_code`)

Overrides built-in functions by replacing them in the symbol table.

Parameters

`function_name`
The function to override.

`function_args`
The function arguments, as a coma separated string. Usually you will want to pass this parameter, as well as the *`function_code`* parameter, as a single quote delimited string. The reason for using single quoted strings, is to protect the variable names from parsing, otherwise, if you use double quotes there will be a need to escape the variable names, e.g. \ *$your_var*.

`function_code`
The new code for the function.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #15 - override_function() **example**

```
<?php
override_function('test', '$a,$b', 'echo "DOING TEST"; return $a * $b;');
?>
```

rename_function

rename_function -- Renames orig_name to new_name in the global function table

Description

bool **rename_function** (string `$original_name`, string `$new_name`)

Renames a orig_name to new_name in the global function table. Useful for temporarily overriding built-in functions.

Parameters

`original_name`
: The original function name.

`new_name`
: The new name for the *`original_name`* function.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #16 - rename_function() **example**

```
<?php
rename_function('mysql_connect', 'debug_mysql_connect' );
?>
```