PHP_Reflect User Guide

Laurent Laville

PHP_Reflect User Guide

Laurent Laville

Table of Contents

List of Tables

Chapter 1. Introduction

PHP_Reflect is a PHP Library that adds the ability to reverse-engineer classes, interfaces, functions, constants, namespaces and more.

This manual documents the final stable version 1.4.2

Chapter 2. Features

- Uses containers that may be changed to store scanning results
- Select properties you want to retrieve (file, start and end lines, docblock, namespace …)
- Ability to extend the parsing level: connect user callbacks to handle more tokens

Chapter 3. System Requirements

Mandatory resources :

- PHP [http://www.php.net] 5.2.0 or newer
- pcre [http://www.php.net/manual/en/book.pcre.php] extension
- SPL [http://www.php.net/manual/en/book.spl.php] extension
- tokenizer [http://www.php.net/manual/en/book.tokenizer.php] extension

Chapter 4. Installing PHP_Reflect

The current version of PHP_Reflect requires **PHP 5.2.0 or newer** to run. If you don't already have an up-to-date version of PHP installed it can be downloaded from the official PHP website http://www.php.net/.

4.1. Using PEAR installer

PHP_Reflect should be installed using the PEAR Installer [http://pear.php.net/]. This installer is the backbone of PEAR, which provides a distribution system for PHP packages, and is shipped with every release of PHP since version 4.3.0.

Registering the channel:

```
pear channel-discover bartlett.laurent-laville.org
```

Installing the latest version available:

```
pear install bartlett/PHP_Reflect
```

Installing a specific version:

```
pear install bartlett/PHP_Reflect-1.2.0
```

4.2. Install manually

Do the following:

1. Download a release archive from http://bartlett.laurent-laville.org/
2. Extract it to a directory that is listed in the include_path of your php.ini configuration file

Chapter 5. Getting Started with PHP_Reflect

5.1. Parse your first PHP source file

Parsing with default options.

```
<?php
require_once 'Bartlett/PHP/Reflect/Autoload.php';

$source = '/path/to/source_file.php';

try {
    $reflect = new PHP_Reflect();
    $reflect->scan($source);

    $classes    = $reflect->getClasses();
    // OR
    $classes    = $reflect['classes'];

    $interfaces = $reflect->getInterfaces();
    // OR
    $interfaces = $reflect['interfaces'];

    $traits     = $reflect->getTraits();
    // OR
    $traits     = $reflect['traits'];

    $functions  = $reflect->getFunctions();
    // OR
    $functions  = $reflect['functions'];

    $constants  = $reflect->getConstants();
    // OR
    $constants  = $reflect['constants'];

    $namespaces = $reflect->getNamespaces();
    // OR
    $namespaces = $reflect['namespaces'];

} catch (Exception $e) {
    echo 'Caught exception: ' . $e->getMessage() . PHP_EOL;
?>
```

Chapter 6. Configure

6.1. Containers

You can change default containers (*namespaces*, *traits*, *interfaces*, *classes*, *functions*, *includes*, *globals*), and then have different getting methods (rather than standard *getInterfaces*, …).

Table 6.1. Containers options

key code	container default name
use	namespaces
namespace	namespaces
trait	traits
interface	interfaces
class	classes
function	functions
require_once	includes
require	includes
include_once	includes
include	includes
variable	globals

Example.

```
<?php
require_once 'Bartlett/PHP/Reflect/Autoload.php';

$source = '/path/to/source_file.php';

try {
    $options = array('containers' => array('function' => 'userFunctions');

    $reflect = new PHP_Reflect($options);
    $reflect->scan($source);

    $functions = $reflect->getUserFunctions();
    // OR
    $functions = $reflect['userFunctions'];

} catch (Exception $e) {
    echo 'Caught exception: ' . $e->getMessage() . PHP_EOL;
?>
```

6.2. Properties

You can choose what information to retrieve depending of element (key code).

Table 6.2. Properties options

key code	default property
use	file, startEndLines, docblock, alias
namespace	file, startEndLines, docblock, alias
trait	file, startEndLines, docblock, namespace, parent, methods
interface	file, startEndLines, docblock, namespace, keywords, parent, methods
class	file, startEndLines, docblock, namespace, keywords, parent, methods, interfaces, package
function	file, startEndLines, docblock, namespace, keywords, signature, arguments, ccn
require_once	file, startEndLines, docblock, namespace
require	file, startEndLines, docblock, namespace
include_once	file, startEndLines, docblock, namespace
include	file, startEndLines, docblock, namespace
variable	file, startEndLines, docblock, namespace

Example 1.

```
<?php
require_once 'Bartlett/PHP/Reflect/Autoload.php';

$source = '/path/to/PEAR-1.9.2/PEAR.php';

try {
    $options = array(
        'properties' => array(
            'interface' => array(
                'keywords', 'parent', 'methods'
            ),
            'class' => array(
                'keywords', 'parent', 'methods', 'interfaces', 'package'
            ),
            'function' => array(
                'keywords', 'signature'
            ),
        )
    );
    $reflect = new PHP_Reflect($options);
    $reflect->scan($source);

    $classes = $reflect->getClasses();

    print_r($classes['\\']['PEAR_Error']['docblock']);
    print_r($classes['\\']['PEAR_Error']['methods']);

} catch (Exception $e) {
    echo 'Caught exception: ' . $e->getMessage() . PHP_EOL;
}
?>
```

Here we got only keywords and signature of each class methods. Class docblock is NULL (first print_r result) because we have decided NOT to get this property as file, startLine + endLine (startEndLines).

```
<?php

Array
(
    [PEAR_Error] => Array
        (
            [keywords] =>
            [signature] => PEAR_Error($message = 'unknown error', $code = null,
                        $mode = null, $options = null, $userinfo = null)
        )

    [getMode] => Array
        (
            [keywords] =>
            [signature] => getMode()
        )

    [getCallback] => Array
        (
            [keywords] =>
            [signature] => getCallback()
        )

    [getMessage] => Array
        (
            [keywords] =>
            [signature] => getMessage()
        )

    [getCode] => Array
        (
            [keywords] =>
            [signature] => getCode()
        )

    [getType] => Array
        (
            [keywords] =>
            [signature] => getType()
        )

    [getUserInfo] => Array
        (
            [keywords] =>
            [signature] => getUserInfo()
        )

    [getDebugInfo] => Array
        (
            [keywords] =>
            [signature] => getDebugInfo()
        )

    [getBacktrace] => Array
        (
```

```
            [keywords] =>
            [signature] => getBacktrace($frame = null)
        )

    [addUserInfo] => Array
        (
            [keywords] =>
            [signature] => addUserInfo($info)
        )

    [__toString] => Array
        (
            [keywords] =>
            [signature] => __toString()
        )

    [toString] => Array
        (
            [keywords] =>
            [signature] => toString()
        )
)
```

Example 2.

```
<?php
require_once 'Bartlett/PHP/Reflect/Autoload.php';

$source = '/path/to/PHPUnit/Framework/TestCase.php';

try {
    $options = array(
        'properties' => array(
            'function' => array(
                'signature', 'arguments'
            ),
        )
    );
    $reflect = new PHP_Reflect($options);
    $reflect->scan($source);

    $classes = $reflect->getClasses();

    print_r($classes['\\']['PHPUnit_Framework_TestCase']['methods']);

} catch (Exception $e) {
    echo 'Caught exception: ' . $e->getMessage() . PHP_EOL;
}
?>
```

And here are results we got on STDOUT :

```
<?php

Array
(
    [methods] => Array
        (
            [__construct] => Array
                (
```

```
                    [signature] => __construct($name = NULL, array $data = array(), $data
                    [arguments] => Array
                        (
                            [0] => Array
                                (
                                    [name] => $name
                                    [defaultValue] => NULL
                                )

                            [1] => Array
                                (
                                    [typeHint] => array
                                    [name] => $data
                                    [defaultValue] => array()
                                )

                            [2] => Array
                                (
                                    [name] => $dataName
                                    [defaultValue] => ''
                                )

                        )

                )


    ( ... more lines again ... )


            [prepareTemplate] => Array
                (
                    [signature] => prepareTemplate(Text_Template $template)
                    [arguments] => Array
                        (
                            [0] => Array
                                (
                                    [typeHint] => Text_Template
                                    [name] => $template
                                )

                        )

                )

        )
)
```

Example 3. With this code below (filename reference is */tmp/ns_sample.php*), we want to get namespaces :

- without imports
- only imports
- all at once

```
<?php
```

```
namespace Doctrine\Common\Cache;

use \Memcache;
use My\Full\Classname as Another;

class MemcacheCache extends CacheProvider
{
    public function setMemcache(Memcache $memcache)
    {
    }

}
```

```
<?php
require_once 'Bartlett/PHP/Reflect/Autoload.php';

$source = '/tmp/ns_sample.php';

try {
    $reflect = new PHP_Reflect();
    $reflect->scan($source);

    $namespaces_without_import = $reflect->getNamespaces();
    // OR
    $namespaces_without_import = $reflect->getNamespaces(PHP_Reflect::NAMESPACES_WITHOUT_

    $namespaces_only_import    = $reflect->getNamespaces(PHP_Reflect::NAMESPACES_ONLY_IM

    $namespaces_all            = $reflect->getNamespaces(PHP_Reflect::NAMESPACES_ALL);

    print_r($namespaces_without_import);
    print_r($namespaces_only_import);
    print_r($namespaces_all);

} catch (Exception $e) {
    echo 'Caught exception: ' . $e->getMessage() . PHP_EOL;
}
?>
```

And here are results we got on STDOUT :

```
<?php
Array
(
    [Doctrine\Common\Cache] => Array
        (
            [startLine] => 2
            [endLine] => 13
            [file] => /tmp/ns_sample.php
            [docblock] =>
            [alias] => Cache
            [import] =>
        )

)
Array
(
    [\Memcache] => Array
        (
            [startLine] => 4
```

```
            [endLine] => 4
            [file] => /tmp/ns_sample.php
            [docblock] =>
            [alias] => Memcache
            [import] => 1
        )

    [My\Full\Classname] => Array
        (
            [startLine] => 5
            [endLine] => 5
            [file] => /tmp/ns_sample.php
            [docblock] =>
            [alias] => Another
            [import] => 1
        )

)
Array
(
    [Doctrine\Common\Cache] => Array
        (
            [startLine] => 2
            [endLine] => 13
            [file] => /tmp/ns_sample.php
            [docblock] =>
            [alias] => Cache
            [import] =>
        )

    [\Memcache] => Array
        (
            [startLine] => 4
            [endLine] => 4
            [file] => /tmp/ns_sample.php
            [docblock] =>
            [alias] => Memcache
            [import] => 1
        )

    [My\Full\Classname] => Array
        (
            [startLine] => 5
            [endLine] => 5
            [file] => /tmp/ns_sample.php
            [docblock] =>
            [alias] => Another
            [import] => 1
        )

)
```

Chapter 7. Connect other tokens

Extends PHP_Reflect core, to match user needs, is possible with usage of the `connect()` function.

Take a real case. *PHP_CompatInfo v2* need to know what are the internal functions used in a source code before to give the minimal php version requires.

To do so, we need to connect a new parser element to token T_STRING. I won't describe the code of this T_STRING parser here: it's not the purpose. But we will see how to do.

Main script.

```
<?php
require_once 'Bartlett/PHP/Reflect/Autoload.php';

$source = '/path/to/PEAR-1.9.2/PEAR.php';

try {
    $options = array('containers' => array('core' => 'internalFunctions'));  ❶

    $reflect = new PHP_Reflect($options);
    $reflect->connect(
        'T_STRING', 'PHP_CompatInfo_Token_STRING',                            ❷
        array('PHP_CompatInfo_TokenParser', 'parseTokenString')               ❸
    );
    $reflect->scan($source);                                                  ❹

    $functions = $reflect->getInternalFunctions();                            ❺
    // OR
    $functions = $reflect['internalFunctions'];                               ❻

    print_r($functions);

} catch (Exception $e) {
    echo 'Caught exception: ' . $e->getMessage() . PHP_EOL;
}
?>
```

❶ we add a new container to store internal PHP functions that will be scanned by our new parser element

❷❸ we connect a user callback that have logic to parse the new token T_STRING

❹ we scan the source file

❺❻ we get the results (OOP or array access interface)

Synopsis of our new token T_STRING.

```
<?php

class PHP_CompatInfo_Token_STRING extends PHP_Reflect_Token_STRING
{
    public function getName()
    {
    }

    public function getType()
    {
```

```
    }

    // private methods ...
}
```

Synopsis of our callback (param #3 of `connect` function).

```
<?php
class PHP_CompatInfo_TokenParser
{
    public static function parseTokenString()
    {
        list($subject, $context, $token) = func_get_args();
        extract($context);

        // ... more
    }
}
```

Final result gave list of 42 PHP functions used in PEAR.php script of PEAR 1.9.2

```
<?php

Array
(
    [0] => define
    [1] => function_exists
    [2] => version_compare
    [3] => zend_version
    [4] => substr
    [5] => ini_set
    [6] => strtolower
    [7] => get_class
    [8] => strcasecmp
    [9] => method_exists
    [10] => register_shutdown_function
    [11] => get_parent_class
    [12] => printf
    [13] => array_key_exists
    [14] => is_a
    [15] => is_null
    [16] => is_string
    [17] => is_callable
    [18] => trigger_error
    [19] => is_array
    [20] => array_push
    [21] => count
    [22] => array_pop
    [23] => in_array
    [24] => array_search
    [25] => is_object
    [26] => end
    [27] => is_int
    [28] => reset
    [29] => intval
    [30] => sizeof
    [31] => extension_loaded
    [32] => ini_get
    [33] => dl
    [34] => array_reverse
```

```
    [35] => each
    [36] => call_user_func_array
    [37] => debug_backtrace
    [38] => sprintf
    [39] => call_user_func
    [40] => defined
    [41] => implode
)
```

PHP_CompatInfo v2 is an excellent exemple of what is possible to do by connecting other tokens.

Appendix A. Copyright

This work is licensed under the BSD License.

The full legal text of the license is given below.

```
Copyright (c) 2011-2012, Laurent Laville <pear@laurent-laville.org>

Credits to Sebastian Bergmann on base concept from phpunit/PHP_Token_Stream

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.
    * Neither the name of the authors nor the names of its contributors
      may be used to endorse or promote products derived from this software
      without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```