PHP Reflect Book

Laurent Laville

PHP Reflect Book

Laurent Laville

Table of Contents

List of Examples

This complete guide documents PHP Reflect 4.0.2, published on 2016-09-22.

Part I. Getting Started

Chapter 1. Download

We distribute a PHP Archive [http://www.php.net/phar] (PHAR) that contains all required dependencies of PHP Reflect bundled in a single file.

Download the latest version [http://bartlett.laurent-laville.org/get/phpreflect-4.0.2.phar]

Make it executable, and put it into your `$PATH`.

```
$ chmod +x phpreflect-4.0.2.phar
$ mv phpreflect-4.0.2.phar /usr/local/bin/phpreflect

$ phpreflect --version
```

You can also immediately use the PHAR after you have downloaded it.

```
$ wget http://bartlett.laurent-laville.org/get/phpreflect-4.0.2.phar
$ php phpreflect-4.0.2.phar --version
```

With both methods then you have this output :

```
phpReflect version 4.0.2
```

Other alternative installations are possible. Please refer to the Chapter 7, *Installation* for details on how to do this.

Chapter 2. Configuration

With the minimalist JSON file `phpreflect.json`.

```
{
    "source-providers": [
        {
            "in": ". as current",
            "name": "/\\.(php|inc|phtml)$/"
        }
    ],
    "plugins": [
    ],
    "analysers": [
    ],
    "services": [
    ]
}
```

Put it in your project's folder. Alternative locations are possible. Please refer to the Chapter 8, *The Json Configuration File* for details on how to do this.

The JSON configuration file is no more required for basic usage. Reserved to advanced users.

Chapter 3. Structure

source-providers

this entry provide list of your data sources to parse.

plugins

this entry list all plugins added to the core base code of PHP Reflect.

analysers

this entry list all analysers that may be used with the `analyser:run` command.

services

this entry list all services that may be used with this application.

Chapter 4. Execution

With the Reflect source code, invoke the following command :

```
$ phpreflect analyser:run .
```

and you should obtain something like this :

```
Data Source Analysed

Directories                                    22
Files                                          77

Structure
  Namespaces                                   22
  Interfaces                                   10
  Traits                                        0
  Classes                                      67
    Abstract Classes                            8 (11.94%)
    Concrete Classes                           59 (88.06%)
  Methods                                     312
    Scope
      Non-Static Methods                      299 (95.83%)
      Static Methods                           13 (4.17%)
    Visibility
      Public Method                           268 (85.90%)
      Protected Method                         35 (11.22%)
      Private Method                            9 (2.88%)
  Functions                                    11
    Named Functions                             0 (0.00%)
    Anonymous Functions                        11 (100.00%)
  Constants                                    21
    Global Constants                            0 (0.00%)
    Magic Constants                             3 (14.29%)
    Class Constants                            18 (85.71%)
  Tests
    Classes                                     0
    Methods                                     0
```

Another analyser (`loc`) is also available, and can be combined or not with `structure` the default analyser.

With the Reflect source code, invoke the following command :

```
$ phpreflect analyser:run . loc
```

and you should obtain something like this :

```
Data Source Analysed

Directories                                    22
Files                                          77

Size
  Lines of Code (LOC)                        3832
  Comment Lines of Code (CLOC)                137 (3.58%)
  Non-Comment Lines of Code (NCLOC)          3695 (96.42%)
```

```
  Logical Lines of Code (LLOC)                    1210 (31.58%)
    Classes                                       1142 (94.38%)
      Average Class Length                          17
      Average Method Length                          3
    Functions                                       68 (5.62%)
      Average Function Length                       6
    Not in classes or functions                     0 (0.00%)

Complexity
  Cyclomatic Complexity / LLOC                    0.53
  Cyclomatic Complexity / Number of Methods       2.73
```

Chapter 5. Summary

Let’s review what we’ve done :

- downloaded the latest stable PHAR version.
- prepared a minimalist JSON configuration file that is OPTIONAL to run Reflect commands.
- executed your first parse on the Reflect data source.

Chapter 6. Next

Choose your way depending of your skill level.

Read more

- Want to learn more about the command line interpreter (CLI) version, interface that do Reflect an easy tool without to write a line of PHP code, have a look on Part II, “User Guide”
- Want to learn more about Reflect architecture and /or you want to extends it to match your needs, have a look on Part V, “Developer Guide”
- You are a user of previous version 1.9 that is really different, and want to upgrade to the new major version 2, and keep your old environment still running, have a look on ???

Part II. User Guide

First visit, you are highly recommended to follow chapters in following order.

1. Installing all necessary Reflect components. See Chapter 7, *Installation*
2. Configuring your project and get ready for your first parsing. See Chapter 8, *The Json Configuration File*
3. Running your first parses with the Command-Line interface. See Chapter 9, *The Command-Line*

All you have to know if you want to upgrade from a previous version 1.x easily.

See ???

Basic Reflect features does not match your needs. Learn how to extend or change some features/behaviors.

See Part V, “Developer Guide”

Chapter 7. Installation

Reflect may be installed in several ways, choose your favorite.

Please read the ??? in case you are upgrading from a version 1.x of PHP Reflect.

7.1. Requirements

Before you install PHP Reflect, you will need an operating system with PHP [http://www.php.net] 5.4.0 or later installed,

Reflect requires the date [http://www.php.net/manual/en/book.datetime.php], json [http://www.php.net/manual/en/book.json.php], reflection [http://www.php.net/manual/en/book.reflection.php], tokenizer [http://www.php.net/manual/en/book.tokenizer.php], phar [http://www.php.net/manual/en/book.phar.php], pcre [http://www.php.net/manual/en/book.pcre.php], and spl [http://www.php.net/manual/en/book.spl.php] extensions. These extensions are usually compiled and enabled by default.

7.2. Composer

Put a file named composer.json at the root of your project, with the content below:

```
{
    "require": {
        "bartlett/php-reflect": "4.0.2"
    }
}
```

And ask Composer [http://getcomposer.org/] to install the dependencies:

```
$ php composer.phar install
```

Or just invoke Composer to install the latest version:

```
$ php composer.phar require bartlett/php-reflect
```

With composer `install` or `create-project` commands, if you want to disable installation of require-dev packages (`doctrine/cache`, `psr/log`, `monolog/monolog`, `bartlett/phpunit-loggertestlistener`), don't forget to specify the `--no-dev` option.

You can also use Composer to create a new project from an existing Reflect package. This is the equivalent of doing a git clone checkout followed by a composer install of the vendors.

```
$ php composer.phar create-project bartlett/php-reflect /path/to/install 4.0.2
```

Where `/path/to/install` is your install directory.

7.3. PHAR

The recommended way for newbies, or just to have a look on features of this library, is to download a PHP Archive that contain all required dependencies of PHP Reflect bundled in a single file.

```
$ wget http://bartlett.laurent-laville.org/get/phpreflect-4.0.2.phar
$ chmod +x phpreflect-4.0.2.phar
$ mv phpreflect-4.0.2.phar /usr/local/bin/phpreflect
$ phpreflect
```

You can also immediately use the PHAR after you have downloaded it.

```
$ wget http://bartlett.laurent-laville.org/get/phpreflect-4.0.2.phar
$ php phpreflect-4.0.2.phar
```

Chapter 8. The Json Configuration File

Reflect may used an optional config file in JSON [http://json.org/] format. It could be found either in the current, `$HOME/.config/`, or `/etc` directory.

By setting the `BARTLETTRC` environment variable it is possible to set the filename of `phpreflect.json` to something else.

E.g: `BARTLETTRC=my-phpreflect.json`

And by setting the `BARTLETT_SCAN_DIR` environment variable it is possible to change directories where to search for the json config file.

E.g: `BARTLETT_SCAN_DIR=.:/var/configs:/tmp/bartlett` (for Linux)

E.g: `BARTLETT_SCAN_DIR=.;\var\configs;\tmp\bartlett` (for Windows)

Take care of different PATH_SEPARATOR and DIRECTORY_SEPARATOR in each platform.

The minimalist JSON file `phpreflect.json` is :

```
{
    "source-providers": [
        {
            "in": ". as current",
            "name": "/\\.(php|inc|phtml)$/"
        }
    ],
    "plugins": [
    ],
    "analysers": [
    ],
    "services": [
    ]
}
```

source-providers

this entry provide list of your data sources to parse.

plugins

this entry list all plugins added to the core base code of PHP Reflect.

analysers

this entry list all analysers that may be used with the `analyser:run` command.

services

this entry list all services that may be used with this application.

8.1. section Source Providers

There are lot of way to filter your data source. Each rule follow the syntax of Symfony Finder [http://symfony.com/doc/current/components/finder.html] Component.

The **Location** is the only mandatory criteria. It tells the Finder which directory to use for the search.

In a simple directory.

```
{
    "in": ". as current"
}
```

If you want to identify a data source easily by a short name, the alias (right of `as`) is compared with the `--alias` option constraint.

Search in several locations.

```
{
    "in": ". as current",
    "in": "src/"
}
```

Use wildcard characters to search in the directories matching a pattern:

```
{
    "in": "src/Bartlett/R*"
}
```

Search directly in archives (phar, zip, tar) with the `phar://` protocol.

```
{
    "in": "phar://path/to/archive.zip"
}
```

Restrict files by name and/or extension.

```
{
    "in": "phar://path/to/archive.zip",
    "name": "*.php"
}
```

Restrict files by size.

```
{
    "in": "phar://path/to/archive.zip",
    "name": "*.php",
    "size": "< 10K"
}
```

Restrict files by last modified dates.

```
{
    "in": ". as current",
    "date": "since yesterday"
}
```

By default, the Finder recursively traverse directories.

Restrict the depth of traversing.

```
{
    "in": ". as current",
    "depth": "< 3"
}
```

Restrict location by only one directory.

```
{
    "in": ". as current",
    "exclude": "vendor"
}
```

Restrict location by 1 or more directories.

```
{
    "in": ". as current",
    "exclude": ["vendor", "tests"]
}
```

8.2. section Plugins

There are a number of optional plugins you can use along with Reflect to add more capabilities.

Take an example with the `Logger` plugin.

In your `phpreflect.json` configuration file, add in `plugins` section the following entry:

```
{
    "name": "Logger",
    "class": "Bartlett\\Reflect\\Plugin\\LogPlugin"
}
```

- The `name` key is (since version 3.0.0-alpha1) comment only.
- The `class` key identify the name of the class that implement the plugin features (must be fully qualified).

The `LogPlugin` used by default the `Bartlett\Reflect\Plugin\Log\DefaultLogger` class that write results to error_log

8.2.1. Cache Plugin

Available since version 2.3.0, but location changed since version 3.0.0-alpha1

In your `phpreflect.json` configuration file, add in `plugins` section the following entry:

```
{
```

```
        "name": "Cache",
        "class": "Bartlett\\Reflect\\Plugin\\CachePlugin",
        "options": {
            "adapter": "DoctrineCacheAdapter",
            "backend": {
                "class": "Doctrine\\Common\\Cache\\FilesystemCache",
                "args": [
                    "%{TEMP}/bartlett/cache"
                ]
            }
        }
    }
```

You may use any environment variable that will be replaced, at run-time, by their value. E.g: `TEMP`, `HOME`

Since release 2.3.0, the `HOME` syntax is compatible Linux/Windows.

If you want to used the same options (Doctrine adapter with file cache) as above, you can used shortcut syntax like this.

```
    {
        "name": "Cache",
        "class": "Bartlett\\Reflect\\Plugin\\CachePlugin",
        "options": []
    }
```

In previous configuration we used the Doctrine Cache adapter and its File system backend. See the same configuration applied with other SAPI, in Section 19.3, “File cache”

8.2.2. Log Plugin

Available since version 2.4.0, but location and options changed since version 3.0.0-alpha1

In your `phpreflect.json` configuration file, add in `plugins` section the following entry:

```
    {
        "name": "Log",
        "class": "Bartlett\\Reflect\\Plugin\\LogPlugin"
    }
```

Where `options` key identify an optional class logger (fully qualified. E.g `YourNamespace\YourLogger`).

When `options` key is not provided, log plugin used the default Reflect logger bundled with distribution. See `Bartlett\Reflect\Plugin\Log\DefaultLogger` that write results to the error log system.

See the Developer Guide for definition examples of some loggers Section 20.3, “Using your private logger” or Section 20.4, “Using Monolog”

8.3. section Analysers

There are two default analysers you can use, but you are free to add your owns.

In your `phpreflect.json` configuration file, add in `analysers` section (for example) the following entry:

```
    {
        "name": "MyAnalyser",
        "class": "Your\\Analysers\\MyAnalyser"
    }
```

- The `name` key is (since version 3.0.0-alpha1) comment only.
- The `class` key identify the name of the class that implement your analyser (must be fully qualified).

Your analyser should implement both interfaces `Bartlett\Reflect\Analyser\AnalyserInterface` and `PhpParser\NodeVisitor`.

Then to use it in command line :

```
$ phpreflect analyser:run /path/to/datasource my
```

`my` identify your analyser (prefix in lower case of `MyAnalyser` class)

Chapter 9. The Command-Line

The command-line interface is the easiest way to try and learn the basic Reflect features.

For all users.

9.1. Command-Line Options

Without `plugins` and `analysers` sections in your `phpreflect.json` configuration file, when you invoke the `phpreflect` command, you should obtain the following commands and options :

```
       _               ____       __ _           _
  _ __ | |__  _ __ |  _ \ ___ / _| | ___  ___| |_
 | '_ \| '_ \| '_ \| |_) / _ \ |_| |/ _ \/ __| __|
 | |_) | | | | |_) |  _ <  __/  _| |  __/ (__| |_
 | .__/|_| |_| .__/|_| \_\___|_| |_|\___|\___|\__|
 |_|         |_|

phpReflect version 4.0.2

Usage:
 command [options] [arguments]

Options:
  -h, --help            Display this help message
  -q, --quiet           Do not output any message
  -V, --version         Display this application version
      --ansi            Force ANSI output
      --no-ansi         Disable ANSI output
  -n, --no-interaction  Do not ask any interactive question
      --no-plugins      Disables all plugins.
      --profile         Display timing and memory usage information.
      --progress        Show progress bar.
      --output=OUTPUT   Write results to file or URL.
      --manifest        Show which versions of dependencies are bundled.
  -v|vv|vvv, --verbose  Increase the verbosity of messages: 1 for normal output, 2 for m

Available commands:
 help                  Displays help for a command
 list                  Lists commands
analyser
 analyser:list         List all analysers available.
 analyser:run          Analyse a data source and display results.
cache
 cache:clear           Clear cache (any adapter and backend).
config
 config:validate       Validates a JSON configuration file.
diagnose
 diagnose:run          Diagnoses the system to identify common errors.
diagram
 diagram:class         Generates diagram about a user class present in a data source.
 diagram:package       Generates diagram about namespaces in a data source.
plugin
```

```
 plugin:list           List all plugins installed.
reflection
 reflection:class      Reports information about a user class present in a data source.
 reflection:function   Reports information about a user function present in a data sourc
```

config:validate Validates an optional JSON config file.

```
$ phpreflect config:validate
```

```
"/etc/phpreflect.json" config file is valid.
```

diagnose:run Diagnoses the system to identify common errors.

```
$ phpreflect diagnose:run
```

```
Checking php settings:
- Requires PHP 5.4.0 or better OK
- php.ini file loaded C:\UwAmp\bin\php\php-5.6.23\php.ini
- date extension loaded YES
- json extension loaded YES
- pcre extension loaded YES
- phar extension loaded YES
- reflection extension loaded YES
- spl extension loaded YES
- tokenizer extension loaded YES
```

Use verbose level 2 for more details, and level 3 to get raw response

diagram:class Generates diagram about a user class present in a data source.

```
$ phpreflect diagram:class Bartlett\Reflect\Plugin\LogPlugin ./src
```

diagram:package Generates diagram about namespaces in a data source.

```
$ phpreflect diagram:package --engine=graphviz ./src
```

plugin:list List all plugins configured (and correctly installed) in `plugins` section of your `phpreflect.json` config file.

Without plugins, you will get.

```
$ phpreflect plugin:list
```

```
No plugin installed.
```

With only `Cache` plugin configured, you will get.

```
$ phpreflect plugin:list
```

```
Plugin Class                              Events Subscribed
Bartlett\Reflect\Plugin\CachePlugin       reflect.progress
                                          reflect.success
                                          reflect.complete
```

analyser:list List all analysers configured in `analysers` section of your `phpreflect.json` config file, and available by default.

With only default analysers, you will get.

```
$ phpreflect analyser:list
```

```
 Analyser Name Analyser Class
 loc           Bartlett\Reflect\Analyser\LocAnalyser
 reflection    Bartlett\Reflect\Analyser\ReflectionAnalyser
 structure     Bartlett\Reflect\Analyser\StructureAnalyser
```

analyser:run Parse a data source and display results. May vary depending of the data source and analyser used.

With `structure` analyser and the Reflect source code, you will get something like.

```
$ phpreflect analyser:run .
```

Possible alternative (if you use the default json config file).

```
$ phpreflect analyser:run --alias current
```

```
Data Source Analysed

Directories                                          22
Files                                                77

Structure
  Namespaces                                         22
  Interfaces                                         10
  Traits                                              0
  Classes                                            67
    Abstract Classes                                  8 (11.94%)
    Concrete Classes                                 59 (88.06%)
  Methods                                           312
    Scope
      Non-Static Methods                            299 (95.83%)
      Static Methods                                 13 (4.17%)
    Visibility
      Public Method                                 268 (85.90%)
      Protected Method                               35 (11.22%)
      Private Method                                  9 (2.88%)
  Functions                                          11
    Named Functions                                   0 (0.00%)
    Anonymous Functions                              11 (100.00%)
  Constants                                          21
    Global Constants                                  0 (0.00%)
    Magic Constants                                   3 (14.29%)
    Class Constants                                  18 (85.71%)
  Tests
    Classes                                           0
    Methods                                           0
```

Filter results with a closure (available since version 3.1.0).

```
$ phpreflect analyser:run --filter=YourFilters.php .
```

```
<?php
$closure = function ($data) {
    $filterOnKeys = array(
        'namespaces',
        'interfaces',
        'traits',
        'classes', 'abstractClasses', 'concreteClasses',
```

```
        'functions', 'namedFunctions', 'anonymousFunctions',
        'classConstants', 'globalConstants', 'magicConstants',
    );

    foreach ($data as $title => &$keys) {
        if (strpos($title, 'StructureAnalyser') === false) {
            continue;
        }
        // looking into Structure Analyser metrics only
        foreach ($keys as $key => $val) {
            if (!in_array($key, $filterOnKeys)) {
                unset($keys[$key]);  // "removed" unsolicited values
                continue;
            }
        }
    }
    return $data;
};

return $closure;
```

```
Data Source Analysed

Directories                                          22
Files                                                77

Structure
  Namespaces                                         22
  Interfaces                                         10
  Traits                                              0
  Classes                                            67
    Abstract Classes                                  8 (11.94%)
    Concrete Classes                                 59 (88.06%)
  Functions                                          11
    Named Functions                                   0 (0.00%)
    Anonymous Functions                              11 (100.00%)
  Constants                                          21
    Global Constants                                  0 (0.00%)
    Magic Constants                                   3 (14.29%)
    Class Constants                                  18 (85.71%)
```

The filter's file that host the `$closure`, must be resolvable through the include_path.

reflection:class Reports information about a user class present in a data source.

With the Reflect source code (`./src`), and `Bartlett\Reflect` class.

```
$ phpreflect reflection:class Bartlett\Reflect ./src
```

```
Class [ <user> class Bartlett\Reflect extends Bartlett\Reflect\Event\Abstrac
  @@ C:\home\github\php-reflect\src\Bartlett\Reflect.php 45 - 340

  - Constants [0] {
  }

  - Properties [2] {
    Property [ private $analysers ]
```

```
    Property [ private $dataSourceId ]
  }

  - Methods [6] {
    Method [ <user> public method __construct ] {
      @@ C:\home\github\php-reflect\src\Bartlett\Reflect.php 53 - 57

      - Parameters [0] {
      }
    }

    Method [ <user> public method addAnalyser ] {
      @@ C:\home\github\php-reflect\src\Bartlett\Reflect.php 66 - 71

      - Parameters [1] {
        Parameter #0 [ <required> PhpParser\NodeVisitor $analyser ]
      }
    }

    Method [ <user> public method getAnalysers ] {
      @@ C:\home\github\php-reflect\src\Bartlett\Reflect.php 78 - 81

      - Parameters [0] {
      }
    }

    Method [ <user> public method setDataSourceId ] {
      @@ C:\home\github\php-reflect\src\Bartlett\Reflect.php 90 - 94

      - Parameters [1] {
        Parameter #0 [ <required> $id ]
      }
    }

    Method [ <user> public method getDataSourceId ] {
      @@ C:\home\github\php-reflect\src\Bartlett\Reflect.php 101 - 104

      - Parameters [0] {
      }
    }

    Method [ <user> public method parse ] {
      @@ C:\home\github\php-reflect\src\Bartlett\Reflect.php 113 - 339

      - Parameters [1] {
        Parameter #0 [ <required> Symfony\Component\Finder\Finder $finder ]
      }
    }
  }
}
```

Chapter 10. Summary

Let’s review what we’ve learned about the command-line interface :

- It’s a Symfony Console Component [http://symfony.com/doc/current/components/console/index.html] that can be extended to infinite via plugins and analysers.
- Default analyser produced results such as PHPLoc [https://github.com/sebastianbergmann/phploc] by Sebastian Bergmann.

Part III. Migration Guide v2

Because the version 2 is a full API rewrites, and used namespaces, your old code cannot migrate without a little change.

We will try to explain how to do in few steps.

Chapter 11. Containers

Version 1.x used customizable containers [http://php5.laurent-laville.org/reflect/manual/1.9/en/configure.html#containers] feature to store parsing results.

For example, in **version 1.x** when we wanted to retrieve user functions, we could either do :

Example 11.1. With standard container

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

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

$options = array();

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

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

Example 11.2. With a non standard container

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

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

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

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

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

In **version 2.x**, we have collections of data models that we can enumerate and exploit.

See API in developer's guide v2.6

- Namespaces [http://php5.laurent-laville.org/reflect/manual/current/../2.6/en/developer-guide--api.html#_packages_or_namespaces] collection
- Classes [http://php5.laurent-laville.org/reflect/manual/current/../2.6/en/developer-guide--api.html#_classes] collection
- Interfaces [http://php5.laurent-laville.org/reflect/manual/current/../2.6/en/developer-guide--api.html#_interfaces] collection
- Traits [http://php5.laurent-laville.org/reflect/manual/current/../2.6/en/developer-guide--api.html#_traits] collection
- Functions [http://php5.laurent-laville.org/reflect/manual/current/../2.6/en/developer-guide--api.html#_functions] collection
- Constants [http://php5.laurent-laville.org/reflect/manual/current/../2.6/en/developer-guide--api.html#_constants] collection
- Includes [http://php5.laurent-laville.org/reflect/manual/current/../2.6/en/developer-guide--api.html#_includes] collection
- Dependencies [http://php5.laurent-laville.org/reflect/manual/current/../2.6/en/developer-guide--api.html#_dependencies] collection

Example 11.3. Enumerate each user functions

```
<?php
require_once 'vendor/autoload.php';

use Bartlett\Reflect;
use Bartlett\Reflect\ProviderManager;
use Bartlett\Reflect\Provider\SymfonyFinderProvider;

use Symfony\Component\Finder\Finder;

$finder = new Finder();
$finder->files()
    ->name('source_file.php')
    ->in('/path/to/');

// Identify Data Source
$pm = new ProviderManager;
$pm->set('Sample', new SymfonyFinderProvider($finder));

$reflect = new Reflect;
$reflect->setProviderManager($pm);
$reflect->parse();

// Exploit results
foreach ($reflect->getPackages() as $package) {
```

```
    $functions = $package->getFunctions();
}
```

Chapter 12. Properties

Version 1.x may provide a variable properties [http://php5.laurent-laville.org/reflect/manual/1.9/en/configure.html#properties] list. Version 2.x provides all properties anytime. It’s up to you to decide to use them or not.

For example, in **version 1.x** when we wanted to retrieve only keywords and signature of each class methods of a data source.

Example 12.1. Configure interface, class and method properties

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

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

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

$classes = $reflect->getClasses();

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

Script output.

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

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

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

    [getMessage] => Array
```

```
        (
            [signature] => getMessage()
        )

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

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

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

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

    [getBacktrace] => Array
        (
            [signature] => getBacktrace($frame = null)
        )

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

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

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

In **version 2.x**, when we did the same.

Example 12.2. Properties on demand

```
<?php
require_once 'vendor/autoload.php';

use Bartlett\Reflect;
use Bartlett\Reflect\ProviderManager;
use Bartlett\Reflect\Provider\SymfonyFinderProvider;

use Symfony\Component\Finder\Finder;

$finder = new Finder();
```

```
$finder->files()
    ->name('PEAR.php')
    ->in('/path/to/PEAR-1.9.2/');

// Identify Data Source
$pm = new ProviderManager;
$pm->set('PEAR192', new SymfonyFinderProvider($finder));

$reflect = new Reflect;
$reflect->setProviderManager($pm);
$reflect->parse();

// Exploit results
$out = array();

foreach ($reflect->getPackages() as $package) {
    foreach ($package->getClasses() as $class) {

        if ($class->getShortName() !== 'PEAR_Error') {
            continue;
        }

        foreach ($class->getMethods() as $method) {

            if ($method->isPrivate()) {
                $visibility = 'private';
            } elseif ($method->isProtected()) {
                $visibility = 'protected';
            } else {
                $visibility = 'public';
            }

            $name = $method->getShortName();

            $parameters = $method->getParameters();
            $args       = array();

            foreach ($parameters as $parameter) {

                $args[] = sprintf(
                    '%s%s%s',
                    $parameter->isPassedByReference() ? '&' : '',
                    '$' . $parameter->getName(),
                    $parameter->isDefaultValueAvailable() ? ' = ' . $parameter->getDefau
                );
            }

            $out[$name] = array(
                'signature' => sprintf('%s %s(%s)', $visibility, $name, implode(',', $arg
            );
        }
    }
}
print_r($out);
```

Script output.

```
Array
(
```

```
    [PEAR_Error] => Array
        (
            [signature] => public PEAR_Error($message = 'unknown error',$code = null,$mo
        )

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

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

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

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

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

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

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

    [getBacktrace] => Array
        (
            [signature] => public getBacktrace($frame = null)
        )

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

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

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

```
)
```

Chapter 13. Summary

Let’s review what we’ve did :

- Compared **Containers** configuration solutions, and how to do it with both versions **1.x** and **2.x**
- Compared **Properties** configuration solutions, and how to do it with both versions **1.x** and **2.x**
- Used some methods of the new **API 2.x**, to enumerate and exploit parsing results.

Part IV. Migration Guide v3

Because the version 3 is a full API rewrites, and used namespaces, your old code cannot migrate without a little change.

We will try to explain how to do in few steps.

Chapter 14. Collections

Version 2.x used collections of data models that we can enumerate and exploit. Version 3.x return only a single data model that match object to reflect.

In **version 2.x**, we have collections of data models that we can enumerate and exploit.

Example 14.1. Exploit model collections to print methods of a user class

```
<?php
require_once 'vendor/autoload.php';

use Bartlett\Reflect;
use Bartlett\Reflect\ProviderManager;
use Bartlett\Reflect\Provider\SymfonyFinderProvider;

use Symfony\Component\Finder\Finder;

$finder = new Finder();
$finder->files()
    ->name('*.php')
    ->in('/path/to/');

// Identify Data Source
$pm = new ProviderManager;
$pm->set('Sample', new SymfonyFinderProvider($finder));

$reflect = new Reflect;
$reflect->setProviderManager($pm);
$reflect->parse();

// Exploit results
foreach ($reflect->getPackages() as $package) {
    foreach ($package->getClasses() as $class) {
        if ('VendorNamespace\\VendorClass' === $class->getName()) {
            $methods = array();

            foreach ($class->getMethods() as $method) {
                $methods[] = $method->getShortName();
            }
            printf( 'Methods are : %s' . PHP_EOL, print_r($methods, true) );
        }
    }
}
```

In **version 3.x**, we have a single data model corresponding to a user class or function.

Example 14.2. Print methods of a user class

```
<?php
require_once 'vendor/autoload.php';

use Bartlett\Reflect\Client;

// creates an instance of client
$client = new Client();
```

```
// request for a Bartlett\Reflect\Api\Reflection
$api = $client->api('reflection');

// perform request, on a data source
$dataSource = '/path/to/';

// equivalent to CLI command `phpreflect reflection:class VendorNamespace\VendorClass /pa
$class = $api->class('VendorNamespace\\VendorClass', $dataSource);

$methods = array();

foreach ($class->getMethods() as $method) {
    $methods[] = $method->getShortName();
}
printf( 'Methods are : %s' . PHP_EOL, print_r($methods, true) );
```

Chapter 15. Summary

Let’s review what we’ve did :

- Analysed a data source
- Exploited `Bartlett\Reflect\Model\ClassModel` object from both versions **2.x** and **3.x**

Part V. Developer Guide

 **API**

Reflect comes with a complete reflection API, almost equivalent to PHP5 reflection [http://www.php.net/manual/en/book.reflection.php].

See Chapter 17, *API*

 **Plugins**

Reflect uses a Symfony EventDispatcher [http://symfony.com/doc/current/components/event_dispatcher/index.html] Component to allow you to easily extend the features list.

See Chapter 18, *Plugins*

- Use cache plugin to speed up future data source parsing.

 **Analysers**

Reflect uses analysers that implements the Visitor [http://en.wikipedia.org/wiki/Visitor_pattern] pattern in a simple and effective way to make the render of your results truly customizable.

See ???

 **Filters**

CompatInfo can filter results, since version 4.2, to make the render of your results truly customizable.

See Chapter 23, *Build your Filters*

Chapter 16. API compared

16.1. Class Reflection

PHP5	Reflect	Description
Class reports information about a class http://www.php.net/manual/en/class.reflectionclass.php `Bartlett\Reflect\Model\ClassModel`		
☑	☑	**__construct** - Constructs a ReflectionClass
☑	☑	**__toString** - Returns the string representation of the ReflectionClass object
☑	☐	**export** - Exports a class
☑	☑	**getConstant** - Gets defined constant
☑	☑	**getConstants** - Gets constants
☑	☐	**getConstructor** - Gets the constructor of the class
☑	☐	**getDefaultProperties** - Gets default properties
☑	☑	**getDocComment** - Gets doc comments
☑	☑	**getEndLine** - Gets end line
☑	☑	**getExtension** - Gets a ReflectionExtension object for the extension which defined the class
☑	☑	**getExtensionName** - Gets the name of

PHP5	Reflect	Description
		the extension which defined the class
☑	☑	**getFileName** - Gets the filename of the file in which the class has been defined
☑	☑	**getInterfaceNames** - Gets the interface names
☑	☑	**getInterfaces** - Gets the interfaces
☑	☑	**getMethod** - Gets a ReflectionMethod for a class method
☑	☑	**getMethods** - Gets an array of methods
☑	☐	**getModifiers** - Gets modifiers
☑	☑	**getName** - Gets class name
☑	☑	**getNamespaceName** - Gets namespace name
☑	☐	**getParentClass** - Gets parent class
☐	☑	**getParentClassName** - Gets parent class name
☑	☑	**getProperties** - Gets properties
☑	☑	**getProperty** - Gets a ReflectionProperty for a class's property
☑	☑	**getShortName** - Gets short name
☑	☑	**getStartLine** - Gets starting line number

PHP5	Reflect	Description
☑	☑	**getStaticProperties** - Gets static properties
☑	☑	**getStaticPropertyValue** - Gets static property value
☑	☐	**getTraitAliases** - Returns an array of trait aliases
☑	☐	**getTraitNames** - Returns an array of names of traits used by this class
☑	☐	**getTraits** - Returns an array of traits used by this class
☑	☑	**hasConstant** - Checks if constant is defined
☑	☑	**hasMethod** - Checks if method is defined
☑	☑	**hasProperty** - Checks if property is defined
☑	☐	**implementsInterface** - Implements interface
☑	☑	**inNamespace** - Checks if class in namespace
☑	☑	**isAbstract** - Checks if class is abstract
☑	☑	**isCloneable** - Returns whether this class is cloneable
☑	☑	**isFinal** - Checks if class is final
☑	☐	**isInstance** - Checks class for instance

PHP5	Reflect	Description
☑	☑	**isInstantiable** - Checks if the class is instantiable
☑	☑	**isInterface** - Checks if the class is an interface
☑	☐	**isInternal** - Checks if class is defined internally by an extension, or the core
☑	☑	**isIterateable** - Checks if iterateable
☑	☑	**isSubclassOf** - Checks if a subclass
☑	☑	**isTrait** - Returns whether this is a trait
☑	☑	**isUserDefined** - Checks if user defined
☑	☐	**newInstance** - Creates a new class instance from given arguments
☑	☐	**newInstanceArgs** - Creates a new class instance from given arguments
☑	☐	**newInstanceWithoutConstr** - Creates a new class instance without invoking the constructor
☑	☐	**setStaticPropertyValue** - Sets static property value

16.2. Constant Reflection

Does not exist in PHP5 Reflection API

PHP5	Reflect	Description
Class reports information about a constant `Bartlett\Reflect\Model\ConstantModel`		
☐	☑	**__construct** - Constructs a Reflection Constant
☐	☑	**__toString** - Returns the string representation of the Reflection Constant object
☐	☑	**getDocComment** - Gets doc comments
☐	☑	**getExtension** - Gets a ReflectionExtension object for the extension which defined the constant
☐	☑	**getExtensionName** - Gets the name of the extension which defined the constant
☐	☑	**getFileName** - Gets the filename of the file in which the constant has been defined
☐	☑	**getName** - Gets constant name
☐	☑	**getNamespaceName** - Gets namespace name
☐	☑	**getShortName** - Gets short name
☐	☑	**getValue** - Gets value
☐	☑	**inNamespace** - Checks if in namespace

PHP5	Reflect	Description
☐	☑	**isInternal** - Checks if constant is defined internally by an extension, or the core
☐	☑	**isMagic** - Checks whether it's a magic constant

16.3. Function Reflection

PHP5	Reflect	Description
Class reports information about a function http://www.php.net/manual/en/class.reflectionfunction.php `Bartlett\Reflect\Model\FunctionModel`		
☑	☑	**__construct** - Constructs a ReflectionFunction
☑	☑	**__toString** - Returns the string representation of the ReflectionFunction object
☑	☐	**export** - Exports a function
☑	☐	**getClosure** - Returns a dynamically created closure for the function
☑	☐	**invoke** - Invokes function
☑	☐	**invokeArgs** - Invokes function with args
☑	☐	**isDisabled** - Checks if function is disabled

16.4. Function Abstract Reflection

PHP5	Reflect	Description
A parent class to ReflectionFunction http://www.php.net/manual/en/class.reflectionfunctionabstract.php		

PHP5	Reflect	Description
`Bartlett\Reflect\Model\AbstractFunctionModel`		
☑	☐	**__clone** - Clones function
☑	☐	**getClosureScopeClass** - Returns the scope associated to the closure
☑	☐	**getClosureThis** - Returns this pointer bound to closure
☑	☑	**getDocComment** - Gets doc comments
☑	☑	**getEndLine** - Gets end line
☑	☑	**getExtension** - Gets a ReflectionExtension object for the extension which defined the function
☑	☑	**getExtensionName** - Gets the name of the extension which defined the function
☑	☑	**getFileName** - Gets the filename of the file in which the function has been defined
☑	☑	**getName** - Gets function name
☑	☑	**getNamespaceName** - Gets namespace name
☑	☑	**getNumberOfParameters** - Gets number of parameters
☑	☑	**getNumberOfRequiredPara** - Gets number of required parameters

PHP5	Reflect	Description
☑	☑	**getParameters** - Gets parameters
☑	☑	**getShortName** - Gets function short name
☑	☑	**getStartLine** - Gets starting line number
☑	☐	**getStaticVariables** - Gets static variables
☑	☑	**inNamespace** - Checks if function in namespace
☑	☑	**isClosure** - Checks if closure
☑	☐	**isDeprecated** - Checks if function deprecated
☑	☐	**isGenerator** - Returns whether this function is a generator
☑	☑	**isInternal** - Checks if function is defined internally by an extension, or the core
☑	☐	**isUserDefined** - Checks if user defined
☑	☐	**returnsReference** - Checks if returns reference

16.5. Method Reflection

PHP5	Reflect	Description
Class reports information about a method http://www.php.net/manual/en/class.reflectionmethod.php `Bartlett\Reflect\Model\MethodModel`		

PHP5	Reflect	Description
☑	☑	**__construct** - Constructs a ReflectionMethod
☑	☑	**__toString** - Returns the string representation of the ReflectionMethod object
☑	☐	**export** - Exports a method
☑	☐	**getClosure** - Returns a dynamically created closure for the method
☑	☐	**getDeclaringClass** - Gets declaring class for the reflected method
☑	☐	**getModifiers** - Gets the method modifiers
☑	☐	**getPrototype** - Gets the method prototype
☑	☐	**invoke** - Invokes method
☑	☐	**invokeArgs** - Invokes method with args
☑	☑	**isAbstract** - Checks if method is abstract
☑	☑	**isConstructor** - Checks if method is a constructor
☑	☑	**isDestructor** - Checks if method is a destructor
☑	☑	**isFinal** - Checks if method is final

PHP5	Reflect	Description
☑	☑	**isPrivate** - Checks if method is private
☑	☑	**isProtected** - Checks if method is protected
☑	☑	**isPublic** - Checks if method is public
☑	☑	**isStatic** - Checks if method is static
☑	☐	**setAccessible** - Set method accessibility

16.6. Parameter Reflection

PHP5	Reflect	Description
Class reports information about a parameter http://www.php.net/manual/en/class.reflectionparameter.php `Bartlett\Reflect\Model\ParameterModel`		
☑	☑	**__construct** - Constructs a ReflectionParameter
☑	☑	**__toString** - Returns the string representation of the ReflectionParameter object
☑	☐	**__clone** - Clones parameter
☑	☑	**allowsNull** - Checks if null is allowed
☑	☐	**canBePassedByValue** - Returns whether this parameter can be passed by value
☑	☐	**export** - Exports a parameter
☑	☐	**getClass** - Gets class

PHP5	Reflect	Description
☐	☑	**getTypeHint** - Gets the type of the parameter (callable, array, class name, or none)
☑	☐	**getDeclaringClass** - Gets declaring class for the reflected parameter
☑	☐	**getDeclaringFunction** - Gets declaring function for the reflected parameter
☑	☑	**getDefaultValue** - Gets default parameter value
☑	☐	**getDefaultValueConstantNa** - Returns the default value's constant name if default value is constant or null
☑	☑	**getName** - Gets parameter name
☑	☑	**getPosition** - Gets parameter position
☑	☑	**isArray** - Checks if parameter expects an array
☑	☐	**isCallable** - Returns whether parameter MUST be callable
☑	☑	**isDefaultValueAvailable** - Checks if a default value is available
☑	☐	**isDefaultValueConstant** - Returns whether the default value of this parameter is constant
☑	☑	**isOptional** - Checks if the parameter is optional

PHP5	Reflect	Description
☑	☑	**isPassedByReference** - Checks if the parameter is passed in by reference

16.7. Property Reflection

PHP5	Reflect	Description
Class reports information about classes properties http://www.php.net/manual/en/class.reflectionproperty.php `Bartlett\Reflect\Model\PropertyModel`		
☑	☑	**__construct** - Constructs a ReflectionProperty
☑	☑	**__toString** - Returns the string representation of the ReflectionProperty object
☑	☐	**__clone** - Clones property
☑	☐	**export** - Exports a property
☐	☑	**getClassName** - Gets class name of the reflected property
☑	☐	**getDeclaringClass** - Gets declaring class for the reflected property
☑	☑	**getDocComment** - Gets doc comments from a property
☑	☐	**getModifiers** - Gets modifiers
☑	☑	**getName** - Gets property name
☑	☑	**getValue** - Gets property value

PHP5	Reflect	Description
☑	☑	**isDefault** - Checks if default value
☑	☑	**isPrivate** - Checks if property is private
☑	☑	**isProtected** - Checks if property is protected
☑	☑	**isPublic** - Checks if property is public
☑	☑	**isStatic** - Checks if property is static
☑	☐	**setAccessible** - Set property accessibility
☑	☐	**setValue** - Set property value

Chapter 17. API

17.1. Data Source Identification

Basic or Complex Strategy to identify the Data Source.

Now, and for the following chapters, we will not mention how you load the classes. Depending of the install strategy you've adopted, Composer or other, don't forget to load your autoloader.

Compare to version 2, Reflect 3 offers two simple strategies to identify the data source.

First, is to give the relative or absolute path to file or directory to parse (without limitation).

Second, is to specify options to customize parsing process, to the Symfony Finder [http://symfony.com/doc/current/components/finder.html] Component.

17.1.1. Basic Strategy

With all SAPI, no JSON config file is required (as it was for Reflect 2). You have just to give the relative or absolute path to file or directory to parse.

It's also possible to specify any archive (phar, zip, tar, tgz, gz, rar) as file source.

Example with a simple file or directory (absolute path).

```
$ phpreflect analyser:run /absolute/path/to/source
```

Example with a simple file or directory (relative path).

```
$ phpreflect analyser:run ./relative/path/to/source
```

17.1.2. Complex Strategy

Still as it was with Reflect 2, you will need to configure your data source in a JSON file.

Syntax is closed to the Symfony Finder Component that is used to limit data source contents to parse.

Example to parse an archive.

```
{
    "source-providers": [
        {
            "in": "phar:///var/dist/owncloud-7.0.2.tar as owncloud7",
            "name": "*.php",
            "exclude": ["3rdparty"]
        }
    ],
    "plugins": [
    ],
    "analysers" : [
    ]
}
```

Do not forget the `phar://` protocol in front of archive identification.

Use alias named here `owncloud7` to identify data source entry in the JSON config file, rather than the full path `phar:///var/dist/owncloud-7.0.2.tar`.

Example to parse a directory.

```
{
    "source-providers": [
        {
            "in": "/home/github/phing/ as phing2",
            "path": ["bin", "classes"],
            "exclude": ["test"],
            "name": "*.php"
        }
    ],
    "plugins": [
    ],
    "analysers" : [
    ]
}
```

Learn more about directives, see Section 8.1, “section Source Providers”

Whatever SAPI you use, all metrics (for each analysers asked) are available at end of parse, in the same format.

With CLI, and Reflect source code, to get a structure report, you have to invoke the following command :

```
$ phpreflect analyser:run /home/github/php-reflect/src
```

With others SAPI, use example https://raw.githubusercontent.com/llaville/php-reflect/master/examples/api_analyser_run.php

and you should obtain something like this :

```
Data Source Analysed

Directories                                      22
Files                                            77

Structure
  Namespaces                                     22
  Interfaces                                     10
  Traits                                          0
  Classes                                        67
    Abstract Classes                              8 (11.94%)
    Concrete Classes                             59 (88.06%)
  Methods                                       312
    Scope
      Non-Static Methods                        299 (95.83%)
      Static Methods                             13 (4.17%)
    Visibility
      Public Method                             268 (85.90%)
      Protected Method                           35 (11.22%)
```

```
     Private Method                                  9 (2.88%)
  Functions                                         11
    Named Functions                                  0 (0.00%)
    Anonymous Functions                             11 (100.00%)
  Constants                                         21
    Global Constants                                 0 (0.00%)
    Magic Constants                                  3 (14.29%)
    Class Constants                                 18 (85.71%)
  Tests
    Classes                                          0
    Methods                                          0
```

This is the default render. But, if you want to compare with other SAPI, activate the debug verbose mode (`-vvv`) to get the raw response. You should obtain something like this :

```
Array
(
    [files] => Array
        (
            [0] => /home/github/php-reflect/src/Bartlett/Reflect/Analyser/AbstractAnalys
            [1] => /home/github/php-reflect/src/Bartlett/Reflect/Analyser/AbstractSniffA
            [2] => /home/github/php-reflect/src/Bartlett/Reflect/Analyser/AnalyserInterf
            [3] => /home/github/php-reflect/src/Bartlett/Reflect/Analyser/AnalyserManage
            [4] => /home/github/php-reflect/src/Bartlett/Reflect/Analyser/LocAnalyser.ph
            [5] => /home/github/php-reflect/src/Bartlett/Reflect/Analyser/ReflectionAnal
            [6] => /home/github/php-reflect/src/Bartlett/Reflect/Analyser/StructureAnaly
            [7] => /home/github/php-reflect/src/Bartlett/Reflect/Api/Analyser.php
            [8] => /home/github/php-reflect/src/Bartlett/Reflect/Api/BaseApi.php
            [9] => /home/github/php-reflect/src/Bartlett/Reflect/Api/Cache.php
            [10] => /home/github/php-reflect/src/Bartlett/Reflect/Api/Config.php
            [11] => /home/github/php-reflect/src/Bartlett/Reflect/Api/Diagnose.php
            [12] => /home/github/php-reflect/src/Bartlett/Reflect/Api/Diagram.php
            [13] => /home/github/php-reflect/src/Bartlett/Reflect/Api/Plugin.php
            [14] => /home/github/php-reflect/src/Bartlett/Reflect/Api/Reflection.php
            [15] => /home/github/php-reflect/src/Bartlett/Reflect/Api/V3/Analyser.php
            [16] => /home/github/php-reflect/src/Bartlett/Reflect/Api/V3/Cache.php
            [17] => /home/github/php-reflect/src/Bartlett/Reflect/Api/V3/Common.php
            [18] => /home/github/php-reflect/src/Bartlett/Reflect/Api/V3/Config.php
            [19] => /home/github/php-reflect/src/Bartlett/Reflect/Api/V3/Diagnose.php
            [20] => /home/github/php-reflect/src/Bartlett/Reflect/Api/V3/Diagram.php
            [21] => /home/github/php-reflect/src/Bartlett/Reflect/Api/V3/Plugin.php
            [22] => /home/github/php-reflect/src/Bartlett/Reflect/Api/V3/Reflection.php
            [23] => /home/github/php-reflect/src/Bartlett/Reflect/Client/ClientInterface
            [24] => /home/github/php-reflect/src/Bartlett/Reflect/Client/LocalClient.php
            [25] => /home/github/php-reflect/src/Bartlett/Reflect/Client.php
            [26] => /home/github/php-reflect/src/Bartlett/Reflect/Collection/ReflectionC
            [27] => /home/github/php-reflect/src/Bartlett/Reflect/Console/Application.ph
            [28] => /home/github/php-reflect/src/Bartlett/Reflect/Console/Command.php
            [29] => /home/github/php-reflect/src/Bartlett/Reflect/Console/CommandFactory
            [30] => /home/github/php-reflect/src/Bartlett/Reflect/Console/Formatter/LocO
            [31] => /home/github/php-reflect/src/Bartlett/Reflect/Console/Formatter/Outp
            [32] => /home/github/php-reflect/src/Bartlett/Reflect/Console/Formatter/Stru
            [33] => /home/github/php-reflect/src/Bartlett/Reflect/Environment.php
            [34] => /home/github/php-reflect/src/Bartlett/Reflect/Event/AbstractDispatch
            [35] => /home/github/php-reflect/src/Bartlett/Reflect/Event/CacheAwareEventD
            [36] => /home/github/php-reflect/src/Bartlett/Reflect/Event/DispatcherInterf
            [37] => /home/github/php-reflect/src/Bartlett/Reflect/Events.php
            [38] => /home/github/php-reflect/src/Bartlett/Reflect/Exception/Exception.ph
            [39] => /home/github/php-reflect/src/Bartlett/Reflect/Exception/ModelExcepti
```

```
            [40] => /home/github/php-reflect/src/Bartlett/Reflect/Model/AbstractFunctionM
            [41] => /home/github/php-reflect/src/Bartlett/Reflect/Model/AbstractModel.php
            [42] => /home/github/php-reflect/src/Bartlett/Reflect/Model/ClassModel.php
            [43] => /home/github/php-reflect/src/Bartlett/Reflect/Model/ConstantModel.php
            [44] => /home/github/php-reflect/src/Bartlett/Reflect/Model/FunctionModel.php
            [45] => /home/github/php-reflect/src/Bartlett/Reflect/Model/MethodModel.php
            [46] => /home/github/php-reflect/src/Bartlett/Reflect/Model/ParameterModel.ph
            [47] => /home/github/php-reflect/src/Bartlett/Reflect/Model/PropertyModel.php
            [48] => /home/github/php-reflect/src/Bartlett/Reflect/MonologConsoleLogger.ph
            [49] => /home/github/php-reflect/src/Bartlett/Reflect/Output/Analyser.php
            [50] => /home/github/php-reflect/src/Bartlett/Reflect/Output/Cache.php
            [51] => /home/github/php-reflect/src/Bartlett/Reflect/Output/Config.php
            [52] => /home/github/php-reflect/src/Bartlett/Reflect/Output/Diagnose.php
            [53] => /home/github/php-reflect/src/Bartlett/Reflect/Output/Diagram.php
            [54] => /home/github/php-reflect/src/Bartlett/Reflect/Output/Plugin.php
            [55] => /home/github/php-reflect/src/Bartlett/Reflect/Output/Reflection.php
            [56] => /home/github/php-reflect/src/Bartlett/Reflect/PhpParser/NodeProcesso
            [57] => /home/github/php-reflect/src/Bartlett/Reflect/PhpParser/NodeProcesso
            [58] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/Cache/CacheAdap
            [59] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/Cache/CacheStor
            [60] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/Cache/DefaultCa
            [61] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/Cache/DoctrineC
            [62] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/CachePlugin.php
            [63] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/Log/DefaultLogg
            [64] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/LogPlugin.php
            [65] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/Notifier/GrowlN
            [66] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/Notifier/Notifi
            [67] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/NotifierPlugin.
            [68] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/PluginInterface
            [69] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/PluginManager.ph
            [70] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/ProfilerPlugin.
            [71] => /home/github/php-reflect/src/Bartlett/Reflect/Sniffer/SniffAbstract.
            [72] => /home/github/php-reflect/src/Bartlett/Reflect/Sniffer/SniffInterface
            [73] => /home/github/php-reflect/src/Bartlett/Reflect/Tokenizer/DefaultToken
            [74] => /home/github/php-reflect/src/Bartlett/Reflect/Util/Timer.php
            [75] => /home/github/php-reflect/src/Bartlett/Reflect/Visitor/VisitorInterfa
            [76] => /home/github/php-reflect/src/Bartlett/Reflect.php
        )

    [Bartlett\Reflect\Analyser\StructureAnalyser] => Array
        (
            [namespaces] => 22
            [interfaces] => 10
            [traits] => 0
            [classes] => 67
            [abstractClasses] => 8
            [concreteClasses] => 59
            [functions] => 11
            [namedFunctions] => 0
            [anonymousFunctions] => 11
            [methods] => 312
            [publicMethods] => 268
            [protectedMethods] => 35
            [privateMethods] => 9
            [nonStaticMethods] => 299
            [staticMethods] => 13
            [constants] => 0
            [classConstants] => 18
            [globalConstants] => 0
```

```
            [magicConstants] => 3
            [testClasses] => 0
            [testMethods] => 0
        )

)
```

- First entry in array is the list of parsed `files`
- Second entry in array is the `structure` analyser result

Each analyser as its own data structure and results, but you will always get the fully qualified class name that identify origin of analyser used.

Example with two analysers (`structure` and `loc`).

```
$ phpreflect analyser:run /home/github/php-reflect/src structure loc
```

```
Raw response
Array
(
    [files] => Array
        (
            ...
        )

    [Bartlett\Reflect\Analyser\StructureAnalyser] => Array
        (
            ...
        )

Array
(
    [files] => Array
        (
            [0] => /home/github/php-reflect/src/Bartlett/Reflect/Analyser/AbstractAnalys
            [1] => /home/github/php-reflect/src/Bartlett/Reflect/Analyser/AbstractSniffA
            [2] => /home/github/php-reflect/src/Bartlett/Reflect/Analyser/AnalyserInterf
            [3] => /home/github/php-reflect/src/Bartlett/Reflect/Analyser/AnalyserManage
            [4] => /home/github/php-reflect/src/Bartlett/Reflect/Analyser/LocAnalyser.ph
            [5] => /home/github/php-reflect/src/Bartlett/Reflect/Analyser/ReflectionAnal
            [6] => /home/github/php-reflect/src/Bartlett/Reflect/Analyser/StructureAnaly
            [7] => /home/github/php-reflect/src/Bartlett/Reflect/Api/Analyser.php
            [8] => /home/github/php-reflect/src/Bartlett/Reflect/Api/BaseApi.php
            [9] => /home/github/php-reflect/src/Bartlett/Reflect/Api/Cache.php
            [10] => /home/github/php-reflect/src/Bartlett/Reflect/Api/Config.php
            [11] => /home/github/php-reflect/src/Bartlett/Reflect/Api/Diagnose.php
            [12] => /home/github/php-reflect/src/Bartlett/Reflect/Api/Diagram.php
            [13] => /home/github/php-reflect/src/Bartlett/Reflect/Api/Plugin.php
            [14] => /home/github/php-reflect/src/Bartlett/Reflect/Api/Reflection.php
            [15] => /home/github/php-reflect/src/Bartlett/Reflect/Api/V3/Analyser.php
            [16] => /home/github/php-reflect/src/Bartlett/Reflect/Api/V3/Cache.php
            [17] => /home/github/php-reflect/src/Bartlett/Reflect/Api/V3/Common.php
            [18] => /home/github/php-reflect/src/Bartlett/Reflect/Api/V3/Config.php
            [19] => /home/github/php-reflect/src/Bartlett/Reflect/Api/V3/Diagnose.php
            [20] => /home/github/php-reflect/src/Bartlett/Reflect/Api/V3/Diagram.php
            [21] => /home/github/php-reflect/src/Bartlett/Reflect/Api/V3/Plugin.php
            [22] => /home/github/php-reflect/src/Bartlett/Reflect/Api/V3/Reflection.php
            [23] => /home/github/php-reflect/src/Bartlett/Reflect/Client/ClientInterface
            [24] => /home/github/php-reflect/src/Bartlett/Reflect/Client/LocalClient.php
```

```
            [25] => /home/github/php-reflect/src/Bartlett/Reflect/Client.php
            [26] => /home/github/php-reflect/src/Bartlett/Reflect/Collection/ReflectionC
            [27] => /home/github/php-reflect/src/Bartlett/Reflect/Console/Application.ph
            [28] => /home/github/php-reflect/src/Bartlett/Reflect/Console/Command.php
            [29] => /home/github/php-reflect/src/Bartlett/Reflect/Console/CommandFactory
            [30] => /home/github/php-reflect/src/Bartlett/Reflect/Console/Formatter/LocO
            [31] => /home/github/php-reflect/src/Bartlett/Reflect/Console/Formatter/Outp
            [32] => /home/github/php-reflect/src/Bartlett/Reflect/Console/Formatter/Stru
            [33] => /home/github/php-reflect/src/Bartlett/Reflect/Environment.php
            [34] => /home/github/php-reflect/src/Bartlett/Reflect/Event/AbstractDispatch
            [35] => /home/github/php-reflect/src/Bartlett/Reflect/Event/CacheAwareEventD
            [36] => /home/github/php-reflect/src/Bartlett/Reflect/Event/DispatcherInterf
            [37] => /home/github/php-reflect/src/Bartlett/Reflect/Events.php
            [38] => /home/github/php-reflect/src/Bartlett/Reflect/Exception/Exception.ph
            [39] => /home/github/php-reflect/src/Bartlett/Reflect/Exception/ModelExcepti
            [40] => /home/github/php-reflect/src/Bartlett/Reflect/Model/AbstractFunction
            [41] => /home/github/php-reflect/src/Bartlett/Reflect/Model/AbstractModel.ph
            [42] => /home/github/php-reflect/src/Bartlett/Reflect/Model/ClassModel.php
            [43] => /home/github/php-reflect/src/Bartlett/Reflect/Model/ConstantModel.ph
            [44] => /home/github/php-reflect/src/Bartlett/Reflect/Model/FunctionModel.ph
            [45] => /home/github/php-reflect/src/Bartlett/Reflect/Model/MethodModel.php
            [46] => /home/github/php-reflect/src/Bartlett/Reflect/Model/ParameterModel.p
            [47] => /home/github/php-reflect/src/Bartlett/Reflect/Model/PropertyModel.ph
            [48] => /home/github/php-reflect/src/Bartlett/Reflect/MonologConsoleLogger.p
            [49] => /home/github/php-reflect/src/Bartlett/Reflect/Output/Analyser.php
            [50] => /home/github/php-reflect/src/Bartlett/Reflect/Output/Cache.php
            [51] => /home/github/php-reflect/src/Bartlett/Reflect/Output/Config.php
            [52] => /home/github/php-reflect/src/Bartlett/Reflect/Output/Diagnose.php
            [53] => /home/github/php-reflect/src/Bartlett/Reflect/Output/Diagram.php
            [54] => /home/github/php-reflect/src/Bartlett/Reflect/Output/Plugin.php
            [55] => /home/github/php-reflect/src/Bartlett/Reflect/Output/Reflection.php
            [56] => /home/github/php-reflect/src/Bartlett/Reflect/PhpParser/NodeProcesso
            [57] => /home/github/php-reflect/src/Bartlett/Reflect/PhpParser/NodeProcesso
            [58] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/Cache/CacheAdap
            [59] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/Cache/CacheStor
            [60] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/Cache/DefaultCa
            [61] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/Cache/DoctrineC
            [62] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/CachePlugin.php
            [63] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/Log/DefaultLogg
            [64] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/LogPlugin.php
            [65] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/Notifier/GrowlN
            [66] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/Notifier/Notifi
            [67] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/NotifierPlugin.
            [68] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/PluginInterface
            [69] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/PluginManager.ph
            [70] => /home/github/php-reflect/src/Bartlett/Reflect/Plugin/ProfilerPlugin.
            [71] => /home/github/php-reflect/src/Bartlett/Reflect/Sniffer/SniffAbstract.
            [72] => /home/github/php-reflect/src/Bartlett/Reflect/Sniffer/SniffInterface
            [73] => /home/github/php-reflect/src/Bartlett/Reflect/Tokenizer/DefaultToken
            [74] => /home/github/php-reflect/src/Bartlett/Reflect/Util/Timer.php
            [75] => /home/github/php-reflect/src/Bartlett/Reflect/Visitor/VisitorInterfa
            [76] => /home/github/php-reflect/src/Bartlett/Reflect.php
        )

    [Bartlett\Reflect\Analyser\LocAnalyser] => Array
        (
            [llocClasses] => 1142
            [llocByNoc] => 0
            [llocByNom] => 0
```

```
            [llocFunctions] => 68
            [llocByNof] => 0
            [llocGlobal] => 0
            [classes] => 67
            [functions] => 11
            [methods] => 350
            [cloc] => 137
            [eloc] => 3293
            [lloc] => 1210
            [wloc] => 402
            [loc] => 3832
            [ccn] => 647
            [ccnMethods] => 607
        )

)

)
```

Chapter 18. Plugins

18.1. Events

Reflect uses a Symfony EventDispatcher [http://symfony.com/doc/current/components/event_dispatcher/index.html] Component to allow you to easily extend the features list.

The EventDispatcher component allow Reflect components to communicate with each other by dispatching events and listening to them.

18.1.1. Event Dispatcher

Reflect implement interface `Bartlett\Reflect\Event\DispatcherInterface`. You can add event listeners and event subscribers to this object.

listeners	Callable functions that are registered on an event dispatcher for specific events.
subscribers	Classes that tell an event dispatcher what methods to listen to and what functions on the class to invoke when the event is triggered. Event subscribers subscribe event listeners to an event dispatcher.

18.1.2. Getting an EventDispatcher

You can get the EventDispatcher of `Bartlett\Reflect\Event\DispatcherInterface` by calling the `getEventDispatcher()` method.

Here is an example :

```
<?php

use Bartlett\Reflect\Client;

// creates an instance of client
$client = new Client();

// request for a Bartlett\Reflect\Api\Analyser
$api = $client->api('analyser');

$dispatcher = $api->getEventDispatcher();
```

18.1.3. Adding Event Listeners

After you have the event dispatcher, you can register event listeners that listen to specific events.

Example 18.1. Add a listener that will echo out files when they are parsed

```
<?php

use Bartlett\Reflect\Client;

use Symfony\Component\EventDispatcher\GenericEvent;
```

```
// creates an instance of client
$client = new Client();

// request for a Bartlett\Reflect\Api\Analyser
$api = $client->api('analyser');

$dispatcher = $api->getEventDispatcher();

$dispatcher->addListener(
    'reflect.progress',
    function (GenericEvent $e) {
        printf(
            'Parsing Data source "%s" in progress ... File "%s"' . PHP_EOL,
            $e['source'],
            $e['file']->getPathname()
        );
    }
);
```

18.1.4. Event Subscribers

Event subscribers are classes that implement interface `Symfony\Component\EventDispatcher\EventSubscriberInterface`. They are used to register one or more event listeners to methods of the class. Event subscribers tell event dispatcher exactly which events to listen to and what method to invoke on the class.

Reflect plugins follow the event subscribers behaviors. Have a look on NotifierPlugin :

```
<?php

use Bartlett\Reflect\Events;

class NotifierPlugin implements PluginInterface, EventSubscriberInterface
{
    public static function getSubscribedEvents()
    {
        $events = array(
            Events::PROGRESS => 'onNotification',
            Events::ERROR    => 'onNotification',
            Events::COMPLETE => 'onNotification',
        );
        return $events;
    }
}
```

This plugin registers event listeners to the `reflect.complete` event of a Reflect parse request.

When the `reflect.complete` event is emitted, the `onNotification` instance method of the plugin is invoked.

18.1.5. Events lifecycle

Event	Action	Informations available
reflect.progess	Before to parse a new file of the data source.	`source` data source identifier or its alias

Event	Action	Informations available
		`file` current file parsed in the data source
reflect.success	After parsing the current file (A cached request will not trigger this event)	`source` data source identifier or its alias `file` current file parsed in the data source `ast` the Abstract Syntax Tree [http://en.wikipedia.org/wiki/Abstract_syntax_tree] result of PHP-Parser [https://github.com/nikic/PHP-Parser]
reflect.error	When PHP Parser raise an error	`source` data source identifier or its alias `file` current file parsed in the data source `error` PHP Parser error message
reflect.complete	When a parse request is over.	`source` data source identifier or its alias

18.2. Register Plugins

In Reflect API 2, and other SAPI than CLI, you have to register a plugin, if you want to use it.

In Reflect API 3, it's no more necessary. All valid plugins defined in the JSON configuration file are automatically registered.

You must define environment variables `BARTLETT_SCAN_DIR` and `BARTLETTRC`, otherwise the JSON config file will not found it.

If you don't want to use any plugins, and de-activated all at once, follow this pattern.

```
<?php

use Bartlett\Reflect\Environment;
use Bartlett\Reflect\Client;

// set default values for BARTLETT_SCAN_DIR
Environment::setScanDir()

// set default value for BARTLETTRC
putenv("BARTLETTRC=phpreflect.json");

// creates an instance of client
$client = new Client();
```

```
// request for a Bartlett\Reflect\Api\Analyser
$api = $client->api('analyser');

// de activate all plugins
$api->activatePlugins(false);

// perform request, on a data source with default analyser (structure)
$dataSource = dirname(__DIR__) . '/src';
$analysers  = array('structure');

// equivalent to CLI command `phpreflect analyser:run ../src`
$metrics = $api->run($dataSource, $analysers);
```

Chapter 19. Cache Plugin

> This plugin is almost unnecessary if you unload the `xdebug` extension that slows down execution. Learn more on RFC [https://github.com/llaville/php-compat-info/issues/167].

19.1. Register Plugin

If you want to use the Doctrine cache component [https://github.com/doctrine/cache], skip this section.

If you want to use your own version of cache plugin, use following pattern.

```
<?php

namespace YourNamespace;

use Bartlett\Reflect\Plugin\CachePlugin as BaseCachePlugin;

class CachePlugin extends BaseCachePlugin
{
    // all additional code you need
}
```

And the configuration in JSON file :

```
{
    "source-providers": [
    ],
    "plugins": [
        {
            "name": "Cache",
            "class": "YourNamespace\\CachePlugin",
            "options": []
        }
    ],
    "analysers" : [
    ],
    "services": [
    ]
}
```

19.2. Doctrine Adapter

Use one of the most famous caching solution, provided by the Doctrine project.

Use this shortcut version that is strictly equivalent to next in section file cache.

```
{
    "source-providers": [
    ],
    "plugins": [
        {
```

```
            "name": "Cache",
            "class": "Bartlett\\Reflect\\Plugin\\CachePlugin",
            "options": []
        }
    ],
    "analysers" : [
    ],
    "services": [
    ]
}
```

19.3. File cache

Doctrine File backend to store your Reflect results in the local file system.

```
{
    "source-providers": [
    ],
    "plugins": [
        {
            "name": "Cache",
            "class": "Bartlett\\Reflect\\Plugin\\CachePlugin",
            "options": {
                "adapter": "DoctrineCacheAdapter",
                "backend": {
                    "class": "Doctrine\\Common\\Cache\\FilesystemCache",
                    "args": [
                        "%{TEMP}/bartlett/cache"
                    ]
                }
            }
        }
    ],
    "analysers" : [
    ],
    "services": [
    ]
}
```

In the source code above, we use the standard Doctrine File cache provider, and store results in the default system temporary directory (see php sys_get_temp_dir() [http://www.php.net/manual/en/function.sys-get-temp-dir.php] function).

19.4. SQLite3 cache

See a full example with doctrine SQLite3Cache driver at https://raw.githubusercontent.com/llaville/php-reflect/master/examples/api_analyser_run_with_cache.php

Chapter 20. Log Plugin

20.1. Register Plugin

If you want to use default logger `Bartlett\Reflect\Plugin\Log\DefaultLogger`, skip this section.

If you want to use your own version of log plugin, use following pattern.

```
<?php

namespace YourNamespace;

use Bartlett\Reflect\Plugin\LogPlugin as BaseLogPlugin;

class LogPlugin extends BaseLogPlugin
{
    // all additional code you need
}
```

And the configuration in JSON file :

```
{
    "source-providers": [
    ],
    "plugins": [
        {
            "name": "Logger",
            "class": "YourNamespace\\LogPlugin",
            "options": []
        }
    ],
    "analysers" : [
    ],
    "services": [
    ]
}
```

20.2. Default logger

Use a solution similar to ErrorLogHandler of Monolog project.

```
{
    "source-providers": [
    ],
    "plugins": [
        {
            "name": "Logger",
            "class": "Bartlett\\Reflect\\Plugin\\LogPlugin",
            "options": []
        }
    ],
    "analysers" : [
    ],
    "services": [
```

```
    ]
}
```

It logs records at `Psr\Log\LogLevel::INFO` level or highter, identified by channel name `DefaultLoggerChannel`.

20.3. Using your private logger

Use your own logger, that must be compatible PSR-3 [http://www.php-fig.org/psr/psr-3/].

```
<?php

namespace YourNamespace;

use Psr\Log\AbstractLogger;

class YourLogger extends AbstractLogger
{
    public function log($level, $message, array $context = array())
    {
    }
}
```

And identify it in the JSON config file, as follow

```
{
    "source-providers": [
    ],
    "plugins": [
        {
            "name": "Logger",
            "class": "YourNamespace\\LogPlugin"
        }
    ],
    "analysers" : [
    ],
    "services": [
    ]
}
```

Or even

```
{
    "source-providers": [
    ],
    "plugins": [
        {
            "name": "Logger",
            "class": "Bartlett\\Reflect\\Plugin\\LogPlugin",
            "options": "YourNamespace\\YourLogger"
        }
    ],
    "analysers" : [
    ],
    "services": [
    ]
}
```

See full example at https://raw.githubusercontent.com/llaville/php-reflect/master/examples/api_analyser_run_with_logger.php

20.4. Using Monolog

Use one of the most famous logging solution compatible PSR-3.

If you want to use Monolog with Reflect on CLI mode, then you should use a wrapper like this.

```
<?php

namespace YourNamespace;

use Monolog\Logger;
use Monolog\Handler\StreamHandler;

class YourLogger extends Logger
{
    public function __construct($name = 'YourLoggerChannel')
    {
        $stream = new StreamHandler('/var/logs/phpreflect.log');
        parent::__construct($name, array($stream));
    }
}
```

And with JSON config file as follow

```
{
    "source-providers": [
    ],
    "plugins": [
        {
            "name": "Logger",
            "class": "Bartlett\\Reflect\\Plugin\\LogPlugin",
            "options": "YourNamespace\\YourLogger"
        }
    ],
    "analysers" : [
    ],
    "services": [
    ]
}
```

Chapter 21. Notifier Plugin

21.1. Register Plugin

Two ways depending of SAPI used.

If you're on Windows or Mac platform, you may have Growl. If you're on Linux, the default bundled growl notifier is not for you. Skip this section.

- Growl for Windows
- Growl for Mac

You'll add to configure your plugin in your `phpreflect.json` file, as follow :

```
    {
        "name": "Notifier",
        "class": "Bartlett\\Reflect\\Plugin\\NotifierPlugin",
        "options" : "Bartlett\\CompatInfo\\Plugin\\Notifier\\GrowlNotifier"
    }
```

- The `name` key is (since version 3.0.0-alpha1) comment only.
- The `class` key identify the name of the class that implement the plugin (must be fully qualified).
- The `options` key identify the name of the class that implement the notifier (must be fully qualified).

Default behaviors are :

- do not notify `reflect.progress` and `reflect.success` events (`enabled` option set to false)
- notify `reflect.error` and `reflect.complete` events, and keep them displayed (`sticky` option set to true).
- used the new `gntp` protocol rather than `udp` basic protocol.

If one or all of those behaviors does not match your need, here is how to change it.

Creates your own growl notifier class, (E.g: `YourNamespace\MyGrowlNotifier`) and put it in somewhere in your `include_path`.

```
<?php

namespace YourNamespace;

use Bartlett\Reflect\Plugin\Notifier\GrowlNotifier as BaseGrowlNotifier;

class MyGrowlNotifier extends BaseGrowlNotifier
{
    public function __construct($application = 'myPhpReflect', $notifications = array(),
        $password = '', $options = array()
    {
```

```
        parent::__construct($application, $notifications, $password, $options);
    }
}
```

We have changed the Growl Application Name to `myPhpReflect`, and used the `udp` protocol.

See http://growl.laurent-laville.org/ to learn more about PEAR/Net_Growl package.

21.1.1. With CLI

You have just to run the `analyser:run` command, and you will be notified when parse is completed.

21.1.2. Other SAPI

This is the standard analyser run process as defined in following script. Default behavior is to activate all plugins that can be registered in the `PluginManager`.

```
<?php

use Bartlett\Reflect\Environment;
use Bartlett\Reflect\Client;

// set default values for BARTLETT_SCAN_DIR
Environment::setScanDir()

// set default value for BARTLETTRC
putenv("BARTLETTRC=phpreflect.json");

// creates an instance of client
$client = new Client();

// request for a Bartlett\Reflect\Api\Analyser
$api = $client->api('analyser');

// perform request, on a data source with default analyser (structure)
$dataSource = dirname(__DIR__) . '/src';
$analysers  = array('structure');

// equivalent to CLI command `phpreflect analyser:run ../src`
$metrics = $api->run($dataSource, $analysers);

var_export($metrics);
```

Chapter 22. Build your Analysers

Analysers implements the Visitor [http://en.wikipedia.org/wiki/Visitor_pattern] pattern in a simple and effective way to make the render of your results truly customizable.

22.1. Visitor pattern

Each `Analyser` class must implement two interfaces `Bartlett\Reflect\Analyser\AnalyserInterface`, and `PhpParser\NodeVisitor`.

Your first own analyser may start like that :

```
<?php

namespace YourNamespace;

use Bartlett\Reflect\Analyser\AnalyserInterface;
use PhpParser\NodeVisitor;

class YourAnalyser implements AnalyserInterface, NodeVisitor
{
    //
    // AnalyserInterface methods
    //

    public function setSubject(Reflect $reflect)
    {
    }

    public function setTokens(array $tokens)
    {
    }

    public function setCurrentFile($path)
    {
    }

    public function getMetrics()
    {
    }

    public function getName()
    {
    }

    public function getNamespace()
    {
    }

    public function getShortName()
    {
    }

    //
    // NodeVisitor methods
    //
```

```
    public function beforeTraverse(array $nodes)
    {
    }

    public function enterNode(Node $node)
    {
    }

    public function leaveNode(Node $node)
    {
    }

    public function afterTraverse(array $nodes)
    {
    }
}
```

An abstract class `Bartlett\Reflect\Analyser\AbstractAnalyser` that implement all required interfaces may be used to initialize common data in a simple way.

Your analyser became as simple like that:

```
<?php

namespace YourNamespace;

use Bartlett\Reflect\Analyser\AbstractAnalyser;

class YourAnalyser extends AbstractAnalyser
{
}
```

22.2. Print results

Once all nodes of AST built by PHP-Parser were traversed, you have to publish your results with the `getMetrics()` method.

Results must be organized as a `key`/`values` pair array, where `key` is the fully qualified name of your analyser (E.g: `YourNamespace\YourAnalyser`), and `values` are your metrics (free organization).

At end of API `analyser/run`, your metrics are returned, and may be exploited as you want. You are free to create a custom render or not. If no output formatter is provided in namespace `YourNamespace\Console\Formatter`, a simple PHP `print_r()` format is returned.

This is the default format in debug mode (verbose level 3 in CLI).

Here is a pattern of skeleton.

```
<?php

namespace YourNamespace\Console\Formatter;

use Symfony\Component\Console\Output\OutputInterface;
```

```
class YourAnalyserOutputFormatter
{
    // $output   === instance of console output
    // $response === your metrics returned by the +YourNamespace\YourAnalyser\getMetrics

    public function __invoke(OutputInterface $output, $response)
    {
    }
}
```

Chapter 23. Build your Filters

If you want to restrict final results to one ore more criteria, the filter feature is what you are waiting for.

This feature was introduced in Reflect 3.1.0.

23.1. Your first filter

Here, our goal is to remove some elements of the default report.

Script `YourFilters.php`.

```
<?php
$closure = function ($data) {
    $filterOnKeys = array(
        'namespaces',
        'interfaces',
        'traits',
        'classes', 'abstractClasses', 'concreteClasses',
        'functions', 'namedFunctions', 'anonymousFunctions',
        'classConstants', 'globalConstants', 'magicConstants',
    );

    foreach ($data as $title => &$keys) {
        if (strpos($title, 'StructureAnalyser') === false) {
            continue;
        }
        // looking into Structure Analyser metrics only
        foreach ($keys as $key => $val) {
            if (!in_array($key, $filterOnKeys)) {
                unset($keys[$key]);  // "removed" unsolicited values
                continue;
            }
        }
    }
    return $data;
};
return $closure;
```

The filter's file that host the `$closure`, must be resolvable through the include_path.

Be carefull, with filter source code, or unwanted results may occured.

You have ability to remove definitively (`unset`), or remove partially (`false`), values in response through the filter.

Only one filter is allowed at same run, but you can combine one or more analyser rules.

23.2. SAPI usage

On CLI, invoke the `analyser:run` command with the `--filter` option. E.g:

```
$ phpreflect analyser:run --filter=YourFilters.php src
```

On other SAPI, follow example pattern like:

```
<?php

use Bartlett\Reflect\Client;

// creates an instance of client
$client = new Client();

// request for a Bartlett\Reflect\Api\Analyser
$api = $client->api('analyser');

// perform request, on a data source with two analysers (structure, loc)
$dataSource = dirname(__DIR__) . '/src';
$analysers  = array('structure', 'loc');

// filter rules on final results
$closure = function ($data) {
    $filterOnKeys = array(
        'classes', 'abstractClasses', 'concreteClasses',
        'classConstants', 'globalConstants', 'magicConstants',
    );

    foreach ($data as $title => &$keys) {
        if (strpos($title, 'StructureAnalyser') === false) {
            continue;
        }
        // looking into Structure Analyser metrics and keep classes and constants info
        foreach ($keys as $key => $val) {
            if (!in_array($key, $filterOnKeys)) {
                unset($keys[$key]);  // "removed" unsolicited values
                continue;
            }
        }
    }
    return $data;
};

// equivalent to CLI command `phpreflect analyser:run --filter=YourFilters.php ../src st
//$metrics = $api->run($dataSource, $analysers, null, false, $closure = 'YourFilters.php

// OR with embeded $closure code
$metrics = $api->run($dataSource, $analysers, null, false, $closure);
```