PHP CompatInfo Book

Laurent Laville

PHP CompatInfo Book

Laurent Laville

Table of Contents

List of Examples

This complete guide documents PHP CompatInfo 3.7.4, published on 2015-04-15.

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 CompatInfo bundled in a single file.

Download the latest version [http://bartlett.laurent-laville.org/get/phpcompatinfo-3.7.4.phar]

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

```
$ chmod +x phpcompatinfo-3.7.4.phar
$ mv phpcompatinfo-3.7.4.phar /usr/local/bin/phpcompatinfo

$ phpcompatinfo --version
```

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

```
$ wget http://bartlett.laurent-laville.org/get/phpcompatinfo-3.7.4.phar
$ php phpcompatinfo-3.7.4.phar --version
```

With both methods then you have this output :

```
phpCompatInfo version 3.7.4
```

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 `phpcompatinfo.json`.

```
{
    "source-providers": [
        {
            "in": ". as current",
            "name": "/\\.(php|inc|phtml)$/"
        }
    ],
    "plugins": [
        {
            "name": "Analyser",
            "class": "Bartlett\\Reflect\\Plugin\\Analyser\\AnalyserPlugin"
        }
    ],
    "analysers" : [
        {
            "name": "Namespace",
            "class": "Bartlett\\CompatInfo\\Analyser\\NamespaceAnalyser"
        },
        {
            "name": "Extension",
            "class": "Bartlett\\CompatInfo\\Analyser\\ExtensionAnalyser"
        },
        {
            "name": "Interface",
            "class": "Bartlett\\CompatInfo\\Analyser\\InterfaceAnalyser"
        },
        {
            "name": "Trait",
            "class": "Bartlett\\CompatInfo\\Analyser\\TraitAnalyser"
        },
        {
            "name": "Class",
            "class": "Bartlett\\CompatInfo\\Analyser\\ClassAnalyser"
        },
        {
            "name": "Function",
            "class": "Bartlett\\CompatInfo\\Analyser\\FunctionAnalyser"
        },
        {
            "name": "Constant",
            "class": "Bartlett\\CompatInfo\\Analyser\\ConstantAnalyser"
        },
        {
            "name": "Summary",
            "class": "Bartlett\\CompatInfo\\Analyser\\SummaryAnalyser"
        },
        {
            "name": "Condition",
            "class": "Bartlett\\CompatInfo\\Analyser\\CodeCondAnalyser"
        }
    ]
}
```

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.

Chapter 3. Structure

source-providers
: this entry provide list of your data sources to parse.

Like CompatInfo always needs a JSON file to run, Remi Collet shared [https://github.com/llaville/php-compat-info/issues/120#issuecomment-46188900] a workaround solution exposed here that allows to run :

```
$ phpcompatinfo analyser:run .
```

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

Don’t forget to add at least this default content, else the `analyser:run` and `analyser:list` commands wouldn’t be available.

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

Don’t miss it, else you could not run the `analyser:run` command.

Chapter 4. Execution

With a default `phpcompatinfo.json` as detailed above, put in the `src/` folder of the CompatInfo source code, and invoke the following command :

```
$ phpcompatinfo analyser:run .
```

and you should obtain something like this :

```
Data Source Analysed

Directories                                          7
Files                                              127

Global Analysis

                    Count Cond PHP min Elements highlight
 Extensions         11    4    5.3.0   Core
 Namespaces         14    0    5.3.0   Bartlett\CompatInfo\Reference
 Interfaces         3     0    5.3.0   Bartlett\CompatInfo\Reference\ReferenceInterface
 Traits             0     0
 Classes            140   1    5.3.0   Bartlett\CompatInfo\Reference\Extension\RarExtens
 User Functions     5     0    5.3.0   Bartlett\CompatInfo\Reference\closure-335-372
 Internal Functions 47    2    5.2.0   json_encode
 Constants          13    3    5.3.0   __DIR__
 Total                    5.3.0
```

Let’s explain the results. We have 12 extensions, 4 optionals due to conditional code, with `Core` that requires at least PHP 5.3.0, and so on for all others components.

Since release 3.3.0 only, if you want to have a full report of all elements without to specify each analyser, just enter the following command :

```
$ phpcompatinfo analyser:run . --php
```

Since release 3.4.0, the `condition` analyser results are also included.

To see details of conditional code (`Cond` column in summary report), run following command :

```
$ phpcompatinfo analyser:run . condition
```

and you should obtain something like this :

```
Data Source Analysed

Directories                                          7
Files                                              127

Conditional Code Analysis

   Condition                          REF      EXT min/Max PHP min/Max
   class_exists(\PHP_Timer)           user                 4.0.0
   defined(INTL_ICU_VERSION)          intl     2.0.0b1     5.3.7
   defined(OPENSSL_VERSION_NUMBER)    openssl  5.2.0       5.2.0
   defined(OPENSSL_VERSION_TEXT)      openssl  5.2.0       5.2.0
   function_exists(curl_version)      curl     4.0.2       4.0.2
   function_exists(event_priority_set) libevent 0.0.5      5.3.0
```

```
Total [6]                                                    5.3.7
```

Chapter 5. Summary

Let’s review what we’ve done :

- downloaded the latest stable PHAR version.
- prepared your JSON configuration file that is required to run CompatInfo commands.
- executed your first parse on the CompatInfo 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 CompatInfo an easy tool without to write a line of PHP code, have a look on Part II, “User Guide”
- Want to learn more about CompatInfo architecture and /or you want to extends it to match your needs, have a look on Part IV, “Developer Guide”
- You are a user of previous version 2.26 that is really different, and want to upgrade to the new major version 3, and keep your old environment still running, have a look on Part III, “Migration Guide”

Part II. User Guide

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

1. Installing all necessary CompatInfo 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 2.x easily.

See Part III, “Migration Guide”

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

See Part IV, “Developer Guide”

Chapter 7. Installation

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

Please read the Part III, “Migration Guide” in case you are upgrading from a version 2.x of PHP CompatInfo.

7.1. Requirements

Before you install PHP CompatInfo, you will need an operating system with PHP [http://www.php.net] 5.3.0 or later installed,

CompatInfo requires the json [http://www.php.net/manual/en/book.json.php], libxml [http://www.php.net/manual/en/book.libxml.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-compatinfo": "~3.7"
    }
}
```

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

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

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 CompatInfo 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-compatinfo /path/to 3.7.4
```

Where `/path/to` 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 CompatInfo bundled in a single file.

```
$ wget http://bartlett.laurent-laville.org/get/phpcompatinfo-3.7.4.phar
```

```
$ chmod +x phpcompatinfo-3.7.4.phar
$ mv phpcompatinfo-3.7.4.phar /usr/local/bin/phpcompatinfo
$ phpcompatinfo
```

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

```
$ wget http://bartlett.laurent-laville.org/get/phpcompatinfo-3.7.4.phar
$ php phpcompatinfo-3.7.4.phar
```

Chapter 8. The Json Configuration File

CompatInfo always needs a file in JSON [http://json.org/] format to run. It should be found either in the current, `$HOME/.config/`, or `/etc` directory.

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

E.g: `COMPATINFO=my-phpcompatinfo.json`

The minimalist JSON file `phpcompatinfo.json` is :

```
{
    "source-providers": [
        {
            "in": ". as current",
            "name": "/\\.(php|inc|phtml)$/"
        }
    ],
    "plugins": [
        {
            "name": "Analyser",
            "class": "Bartlett\\Reflect\\Plugin\\Analyser\\AnalyserPlugin"
        }
    ],
    "analysers" : [
        {
            "name": "Namespace",
            "class": "Bartlett\\CompatInfo\\Analyser\\NamespaceAnalyser"
        },
        {
            "name": "Extension",
            "class": "Bartlett\\CompatInfo\\Analyser\\ExtensionAnalyser"
        },
        {
            "name": "Interface",
            "class": "Bartlett\\CompatInfo\\Analyser\\InterfaceAnalyser"
        },
        {
            "name": "Trait",
            "class": "Bartlett\\CompatInfo\\Analyser\\TraitAnalyser"
        },
        {
            "name": "Class",
            "class": "Bartlett\\CompatInfo\\Analyser\\ClassAnalyser"
        },
        {
            "name": "Function",
            "class": "Bartlett\\CompatInfo\\Analyser\\FunctionAnalyser"
        },
        {
            "name": "Constant",
            "class": "Bartlett\\CompatInfo\\Analyser\\ConstantAnalyser"
        },
        {
            "name": "Summary",
            "class": "Bartlett\\CompatInfo\\Analyser\\SummaryAnalyser"
```

```
        },
        {
            "name": "Condition",
            "class": "Bartlett\\CompatInfo\\Analyser\\CodeCondAnalyser"
        },
        {
            "name": "Structure",
            "class": "Bartlett\\Reflect\\Analyser\\StructureAnalyser"
        },
        {
            "name": "Composer",
            "class": "Bartlett\\CompatInfo\\Analyser\\ComposerAnalyser"
        }
    ]
}
```

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 CompatInfo.

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

8.1. 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. Plugins

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

The `Analyser` is the only mandatory plugin you should add to parse your data source.

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

```
{
    "name": "Analyser",
    "class": "Bartlett\\Reflect\\Plugin\\Analyser\\AnalyserPlugin"
}
```

The `Analyser` plugin is a component from Reflect, and not from CompatInfo. Take care of namespace !

The `name` key identify the namespace of optional commands the plugin may provide.

Each `name` key must be unique to avoid conflicts.

The `class` key identify the name of the class that implement the plugin features.

8.2.1. Cache Plugin

Available only since version 3.3.0

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

```
{
    "name": "Cache",
    "class": "Bartlett\\Reflect\\Plugin\\Cache\\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 3.3.0, the `HOME` syntax is compatible Linux/Windows.

Take care to use the same configuration as in PHP Reflect, or you should not share the cache results.

In previous configuration we used the Doctrine Cache adapter and its File system backend. See the same configuration applied with other SAPI, in Section 16.3, “File cache”

8.2.2. Log Plugin

Available only since version 3.4.0

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

```
{
    "name": "Log",
    "class": "Bartlett\\Reflect\\Plugin\\Log\\LogPlugin",
    "options": {
        "logger": {
            "class": "YourLogger"
        },
        "conf": []
    }
}
```

Where `YourLogger` identify your class logger (fully qualified. E.g `YourNamespace\YourLogger`).

See the Developer Guide for definition examples of some loggers Section 17.2, “Using your private logger” or Section 17.3, “Using Monolog”

And `conf` entry are custom plugin options to apply.

For example, to suppress logging of `reflect.success` event, and remove contextual data on `reflect.complete` event, modify your `phpreflect.json` configuration file as follow :

```
{
    "name": "Log",
    "class": "Bartlett\\Reflect\\Plugin\\Log\\LogPlugin",
    "options": {
        "logger": {
            "class": "YourLogger"
        },
        "conf": {
            "reflect.success": false,
            "reflect.complete": {
                "context": false
            }
        }
    }
}
```

All configuration options are available, see Section 16.1, “Register Plugin”

8.3. Analysers

There are a number of optional analysers you can use along with the Reflect Analyser Plugin.

The `Summary` is the default analyser you should add to obtain results when you parse your data source.

In your `phpcompatinfo.json` configuration file, add in `analysers` section the following entry:

```
    {
        "name": "Structure",
        "class": "Bartlett\\CompatInfo\\Analyser\\SummaryAnalyser"
    }
```

The `name` key identify the name you can optionally invoke with the `analyser:run` command.

The two following commands do the same:

Used implicitly the summary analyser (default behavior).

```
$ phpcompatinfo analyser:run .
```

Named explicitly the summary analyser.

```
$ phpcompatinfo analyser:run . summary
```

Each `name` key must be unique to avoid conflicts.

The `class` key identify the name of the class that implement the analyser features.

8.3.1. Conditional Code

Available only since version 3.4.0

CompatInfo is now capable to detect conditional code, mark each element in all analysers reporting, and exclude their versions from PHP min/max final results.

Let’s see with CompatInfo source code, and explain results.

The new condition analyser (`CodeCondAnalyser`), is able to produce such kind of report :

```
Data Source Analysed

Directories                                        7
Files                                            127

Conditional Code Analysis

   Condition                           REF      EXT min/Max PHP min/Max
   class_exists(\PHP_Timer)            user                 4.0.0
   defined(INTL_ICU_VERSION)           intl     2.0.0b1     5.3.7
   defined(OPENSSL_VERSION_NUMBER)     openssl  5.2.0       5.2.0
   defined(OPENSSL_VERSION_TEXT)       openssl  5.2.0       5.2.0
   function_exists(curl_version)       curl     4.0.2       4.0.2
   function_exists(event_priority_set) libevent 0.0.5       5.3.0
   Total [6]                                                5.3.7
```

That meens CompatInfo need at least PHP 5.3.7 if you used all conditional features. But remember, that the minimum version is only PHP 5.3.0 (see summary report).

```
Data Source Analysed
```

```
Directories                                       7
Files                                           127

Global Analysis

                    Count Cond PHP min Elements highlight
 Extensions         11    4    5.3.0   Core
 Namespaces         14    0    5.3.0   Bartlett\CompatInfo\Reference
 Interfaces         3     0    5.3.0   Bartlett\CompatInfo\Reference\ReferenceInterface
 Traits             0     0
 Classes            140   1    5.3.0   Bartlett\CompatInfo\Reference\Extension\RarExtens
 User Functions     5     0    5.3.0   Bartlett\CompatInfo\Reference\closure-335-372
 Internal Functions 47    2    5.2.0   json_encode
 Constants          13    3    5.3.0   __DIR__
 Total                    5.3.0
```

The new `Cond` column tell you how many condition where found in each category. And of course the condition analyser give you the details.

And finally, all other analysers show you with a `C` mark in front of element name when it's excluded from PHP min/max versions, and considered as conditional code.

Chapter 9. The Command-Line

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

For all users.

9.1. Command-Line Options

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

```
phpCompatInfo version 3.7.0

Usage:
  [options] command [arguments]

Options:
  --help           -h Display this help message.
  --quiet          -q Do not output any message.
  --verbose        -v|vv|vvv Increase the verbosity of messages: 1 for normal output, 2 f
  --version        -V Display this application version.
  --ansi              Force ANSI output.
  --no-ansi           Disable ANSI output.
  --no-interaction -n Do not ask any interactive question.
  --profile           Display timing and memory usage information.

Available commands:
  help               Displays help for a command
  list               Lists commands
plugin
  plugin:list        List all plugins installed.
provider
  provider:display   Show source of a file in a data source.
  provider:list      List all data source providers.
  provider:show      Show list of files in a data source.
reference
  reference:list     List all references supported.
  reference:show     Show information about a reference.
```

plugin:list

List all plugins configured (and correctly installed) in `plugins` section of your `phpcompatinfo.json` config file.

Without plugins, you will get.

```
$ phpcompatinfo plugin:list
```

```
 [Json Configuration]
No plugins detected.
```

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

```
$ phpcompatinfo plugin:list
```

```
 Plugin Name Plugin Class                                          Events Subscrib
```

```
 Analyser    Bartlett\Reflect\Plugin\Analyser\AnalyserPlugin reflect.complet
```

provider:list — List all data source providers configured in `source-providers` section of your `phpcompatinfo.json` config file.

Result may vary depending of your current directory, but you will get something like.

```
$ phpcompatinfo provider:list
```

```
 Source Alias   Files
 .      current     46
```

provider:show — Show list of files corresponding to the (Symfony) Finder rules defined.

With Reflect source files.

```
$ phpcompatinfo provider:show .
```

Possible alternative.

```
$ phpcompatinfo provider:show --alias current
```

```
Source                                                          Files
.                                                               46
Relative Path Name                                              Date
Bartlett\Reflect\Analyser\AbstractAnalyser.php                  2014-02-03T17:25:07
Bartlett\Reflect\Analyser\AnalyserInterface.php                 2014-02-03T17:26:50
Bartlett\Reflect\Analyser\StructureAnalyser.php                 2014-02-23T17:31:16

<... more lines ...>
```

provider:display — Show source code of a file in one of the data source identified.

With `Bartlett\Reflect.php` in the *current* data source.

```
$ phpcompatinfo provider:display . Bartlett\Reflect.php
```

Possible alternative.

```
$ phpcompatinfo provider:display --alias current Bartlett\Reflect.php
```

```
Source
.
Relative Path Name Date Size
Id    Token                           Line  Text
    0 T_OPEN_TAG                         1 <?php
    1 T_DOC_COMMENT                      2 /** * Reflect * Reverse-engineer
    2 T_WHITESPACE                      15
    3 T_NAMESPACE                       17 namespace
    4 T_WHITESPACE                      17
    5 T_STRING                          17 Bartlett
    6 T_SEMICOLON                       17 ;

<... more lines ...>
```

reference:list — Show the list of all references supported in this version.

```
$ phpcompatinfo reference:list
```

```
 References     Version     Loaded
```

```
amqp           1.4.0       1.4.0
apc            3.1.13      4.0.6
apcu           4.0.4       4.0.6

<... more lines ...>

Zend OPcache   7.0.4-devFE 7.0.4-devFE
zip            1.12.4      1.12.4
zlib           2.0         2.0
Total [100]
```

- Loaded version is referenced only if the corresponding extension is loaded from your `php.ini` configuration file.
- For PHP extensions that are always enabled by default, the versions (current and loaded) depends of your platform.

reference:show Introspection of a reference (case insensitive)

For example, with apc reference.

```
$ phpcompatinfo reference:show apc
```

List all elements of apc extension.

```
IniEntries                  REF EXT min/Max PHP min/Max
apc.cache_by_default            3.0.0       4.3.0
apc.canonicalize                3.1.1       5.1.0
apc.coredump_unmap              3.0.19      4.3.0
apc.enable_cli                  3.0.11      4.3.0
apc.enabled                     2.0.0       4.0.0
apc.file_md5                    3.1.1       5.1.0
apc.file_update_protection      3.0.11      4.3.0
apc.filters                     2.0.0       4.0.0
apc.gc_ttl                      2.0.0       4.0.0
apc.include_once_override       3.0.19      4.3.0
apc.lazy_classes                3.1.12      5.1.0
apc.lazy_functions              3.1.12      5.1.0
apc.max_file_size               3.0.11      4.3.0
apc.mmap_file_mask              2.0.0       4.0.0
apc.num_files_hint              2.0.0       4.0.0
apc.optimization                2.0.0       4.0.0
apc.preload_path                3.1.1       5.1.0
apc.report_autofilter           3.0.11      4.3.0
apc.rfc1867                     3.0.19      4.3.0
apc.rfc1867_freq                3.0.19      4.3.0
apc.rfc1867_name                3.0.19      4.3.0
apc.rfc1867_prefix              3.0.19      4.3.0
apc.rfc1867_ttl                 3.1.1       5.1.0
apc.serializer                  3.1.12      5.1.0
apc.shm_segments                2.0.0       4.0.0
apc.shm_size                    2.0.0       4.0.0
apc.shm_strings_buffer          3.1.12      5.1.0
apc.slam_defense                3.0.0       4.3.0
apc.stat                        3.0.11      4.3.0
apc.stat_ctime                  3.0.19      4.3.0
apc.ttl                         3.0.0       4.3.0
apc.use_request_time            3.1.12      5.1.0
apc.user_entries_hint           3.0.0       4.3.0
```

```
apc.user_ttl                    3.0.0       4.3.0
apc.write_lock                  3.0.11      4.3.0
Total [35]
Constants             REF EXT min/Max PHP min/Max
APC_BIN_VERIFY_CRC32      3.1.4       5.1.0
APC_BIN_VERIFY_MD5        3.1.4       5.1.0
APC_ITER_ALL              3.1.1       5.1.0
APC_ITER_ATIME            3.1.1       5.1.0
APC_ITER_CTIME            3.1.1       5.1.0
APC_ITER_DEVICE           3.1.1       5.1.0
APC_ITER_DTIME            3.1.1       5.1.0
APC_ITER_FILENAME         3.1.1       5.1.0
APC_ITER_INODE            3.1.1       5.1.0
APC_ITER_KEY              3.1.1       5.1.0
APC_ITER_MD5              3.1.1       5.1.0
APC_ITER_MEM_SIZE         3.1.1       5.1.0
APC_ITER_MTIME            3.1.1       5.1.0
APC_ITER_NONE             3.1.1       5.1.0
APC_ITER_NUM_HITS         3.1.1       5.1.0
APC_ITER_REFCOUNT         3.1.1       5.1.0
APC_ITER_TTL              3.1.1       5.1.0
APC_ITER_TYPE             3.1.1       5.1.0
APC_ITER_VALUE            3.1.1       5.1.0
APC_LIST_ACTIVE           3.1.1       5.1.0
APC_LIST_DELETED          3.1.1       5.1.0
Total [21]
Functions             REF EXT min/Max PHP min/Max
apc_add                   3.0.13      4.3.0
apc_bin_dump              3.1.4       5.1.0
apc_bin_dumpfile          3.1.4       5.1.0
apc_bin_load              3.1.4       5.1.0
apc_bin_loadfile          3.1.4       5.1.0
apc_cache_info            2.0.0       4.0.0
apc_cas                   3.1.1       5.1.0
apc_clear_cache           2.0.0       4.0.0
apc_compile_file          3.0.13      4.3.0
apc_dec                   3.1.1       5.1.0
apc_define_constants      3.0.0       4.3.0
apc_delete                3.0.0       4.3.0
apc_delete_file           3.1.1       5.1.0
apc_exists                3.1.4       5.1.0
apc_fetch                 3.0.0       4.3.0
apc_inc                   3.1.1       5.1.0
apc_load_constants        3.0.0       4.3.0
apc_sma_info              2.0.0       4.0.0
apc_store                 3.0.0       4.3.0
Total [19]
Classes     REF EXT min/Max PHP min/Max
APCIterator     3.1.1       5.1.0
Total [1]
```

List all elements of apc extension that requires at least PHP 5.1.

```
$ phpcompatinfo reference:show --php=">= 5.1.0" apc
```

```
IniEntries              REF EXT min/Max PHP min/Max
apc.canonicalize            3.1.1       5.1.0
apc.file_md5                3.1.1       5.1.0
apc.lazy_classes            3.1.12      5.1.0
```

```
apc.lazy_functions          3.1.12      5.1.0
apc.preload_path            3.1.1       5.1.0
apc.rfc1867_ttl             3.1.1       5.1.0
apc.serializer              3.1.12      5.1.0
apc.shm_strings_buffer      3.1.12      5.1.0
apc.use_request_time        3.1.12      5.1.0
Total [9/35]
Constants             REF EXT min/Max PHP min/Max
APC_BIN_VERIFY_CRC32      3.1.4       5.1.0
APC_BIN_VERIFY_MD5        3.1.4       5.1.0
APC_ITER_ALL              3.1.1       5.1.0
APC_ITER_ATIME            3.1.1       5.1.0
APC_ITER_CTIME            3.1.1       5.1.0
APC_ITER_DEVICE           3.1.1       5.1.0
APC_ITER_DTIME            3.1.1       5.1.0
APC_ITER_FILENAME         3.1.1       5.1.0
APC_ITER_INODE            3.1.1       5.1.0
APC_ITER_KEY              3.1.1       5.1.0
APC_ITER_MD5              3.1.1       5.1.0
APC_ITER_MEM_SIZE         3.1.1       5.1.0
APC_ITER_MTIME            3.1.1       5.1.0
APC_ITER_NONE             3.1.1       5.1.0
APC_ITER_NUM_HITS         3.1.1       5.1.0
APC_ITER_REFCOUNT         3.1.1       5.1.0
APC_ITER_TTL              3.1.1       5.1.0
APC_ITER_TYPE             3.1.1       5.1.0
APC_ITER_VALUE            3.1.1       5.1.0
APC_LIST_ACTIVE           3.1.1       5.1.0
APC_LIST_DELETED          3.1.1       5.1.0
Total [21/21]
Functions         REF EXT min/Max PHP min/Max
apc_bin_dump          3.1.4       5.1.0
apc_bin_dumpfile      3.1.4       5.1.0
apc_bin_load          3.1.4       5.1.0
apc_bin_loadfile      3.1.4       5.1.0
apc_cas               3.1.1       5.1.0
apc_dec               3.1.1       5.1.0
apc_delete_file       3.1.1       5.1.0
apc_exists            3.1.4       5.1.0
apc_inc               3.1.1       5.1.0
Total [9/19]
Classes     REF EXT min/Max PHP min/Max
APCIterator     3.1.1       5.1.0
Total [1/1]
```

List ini directives of apc extension, that require only PHP 4.3.

```
$ phpcompatinfo reference:show --ini --php="== 4.3.0" apc
```

```
IniEntries                  REF EXT min/Max PHP min/Max
apc.cache_by_default            3.0.0       4.3.0
apc.coredump_unmap              3.0.19      4.3.0
apc.enable_cli                  3.0.11      4.3.0
apc.file_update_protection      3.0.11      4.3.0
apc.include_once_override       3.0.19      4.3.0
apc.max_file_size               3.0.11      4.3.0
apc.report_autofilter           3.0.11      4.3.0
apc.rfc1867                     3.0.19      4.3.0
apc.rfc1867_freq                3.0.19      4.3.0
```

```
apc.rfc1867_name                3.0.19      4.3.0
apc.rfc1867_prefix              3.0.19      4.3.0
apc.slam_defense                3.0.0       4.3.0
apc.stat                        3.0.11      4.3.0
apc.stat_ctime                  3.0.19      4.3.0
apc.ttl                         3.0.0       4.3.0
apc.user_entries_hint           3.0.0       4.3.0
apc.user_ttl                    3.0.0       4.3.0
apc.write_lock                  3.0.11      4.3.0
Total [18/35]
```

The possible operators are: <, <=, >, >=, ==, =, !=, <>.

When the `Analyser` plugin is installed, following lines added into `analysers` section

```
    {
        "name": "Analyser",
        "class": "Bartlett\\Reflect\\Plugin\\Analyser\\AnalyserPlugin"
    }
```

you will get two additionnal commands.

analyser:list — List all analysers configured in `analysers` section of your `phpcompatinfo.json` config file.

Without analysers, you will get.

```
$ phpcompatinfo analyser:list
```

```
[Json Configuration]
No analysers detected.
```

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

```
$ phpcompatinfo analyser:list
```

```
Analyser Name Analyser Class
Class         Bartlett\CompatInfo\Analyser\ClassAnalyser
Composer      Bartlett\CompatInfo\Analyser\ComposerAnalyser
Condition     Bartlett\CompatInfo\Analyser\CodeCondAnalyser
Constant      Bartlett\CompatInfo\Analyser\ConstantAnalyser
Extension     Bartlett\CompatInfo\Analyser\ExtensionAnalyser
Function      Bartlett\CompatInfo\Analyser\FunctionAnalyser
Interface     Bartlett\CompatInfo\Analyser\InterfaceAnalyser
Namespace     Bartlett\CompatInfo\Analyser\NamespaceAnalyser
Structure     Bartlett\Reflect\Analyser\StructureAnalyser
Summary       Bartlett\CompatInfo\Analyser\SummaryAnalyser
Trait         Bartlett\CompatInfo\Analyser\TraitAnalyser
```

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

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

```
$ phpcompatinfo analyser:run .
```

Possible alternative.

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

```
Data Source Analysed

Directories                                             7
Files                                                 127

Global Analysis

                    Count Cond PHP min Elements highlight
 Extensions         11    4    5.3.0   Core
 Namespaces         14    0    5.3.0   Bartlett\CompatInfo\Reference
 Interfaces         3     0    5.3.0   Bartlett\CompatInfo\Reference\Referen
 Traits             0     0
 Classes            140   1    5.3.0   Bartlett\CompatInfo\Reference\Extensi
 User Functions     5     0    5.3.0   Bartlett\CompatInfo\Reference\closure
 Internal Functions 47    2    5.2.0   json_encode
 Constants          13    3    5.3.0   __DIR__
 Total                      5.3.0
```

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.
- You can examine inside a reference and filters elements.

Part III. Migration Guide

 **CLI**

Begin first with the commands of CompatInfo in CLI mode.

See Chapter 11, *CLI*

 **XML**

How you can customize CompatInfo in CLI mode only.

See Chapter 12, *Configuration file*

 **API**

Due to namespaces, API are incompatible in versions 2 and 3. Here are code to do the same things.

See Chapter 13, *Server API*

Chapter 11. CLI

11.1. Progress Bar

Since version 2.12, there is a new progress bar (ALA PHPUnit).

```
PHP_CompatInfo 2.12.0 by Laurent Laville

....CC.................................................  60 / 128 ( 46%)
....C..C.....................................C....C.... 120 / 128 ( 93%)
.C.....C
```

- each `C` character tell us that there is a source file with conditional code.

With version 3.2, the progress bar (ALA Symfony Console Progress Helper) is activated with the first verbose level (`-v`). For example :

```
$ phpcompatinfo -v analyser:run <SOURCE>
```

Renders

```
    3/128 [>---------------------------]   0% Elapsed: 2 secs
```

References

If we want to display list of references supported, here is how to do with both versions :

Version 2.26.

```
$ phpcompatinfo list-references
```

Version 3.2.

```
$ phpcompatinfo reference:list
```

Now, if you want to details each reference one by one identified by its name (`<REF>` in following examples), and got :

- Extensions

Version 2.26.

```
$ phpcompatinfo list --reference=ALL extensions
or
$ phpcompatinfo list-extensions --reference=ALL
```

Version 3.2.

```
$ phpcompatinfo reference:list
```

- Interfaces

Version 2.26.

```
$ phpcompatinfo list --reference=ALL interfaces <REF>
or
$ phpcompatinfo list-interfaces --reference=ALL <REF>
```

Version 3.2.

```
$ phpcompatinfo reference:show --interfaces <REF>
```

- Classes

Version 2.26.

```
$ phpcompatinfo list --reference=ALL classes <REF>
or
$ phpcompatinfo list-classes --reference=ALL <REF>
```

Version 3.2.

```
$ phpcompatinfo reference:show --classes <REF>
```

- Functions

Version 2.26.

```
$ phpcompatinfo list --reference=ALL functions <REF>
or
$ phpcompatinfo list-functions --reference=ALL <REF>
```

Version 3.2.

```
$ phpcompatinfo reference:show --functions <REF>
```

- Constants

Version 2.26.

```
$ phpcompatinfo list --reference=ALL constants <REF>
or
$ phpcompatinfo list-constants --reference=ALL <REF>
```

Version 3.2.

```
$ phpcompatinfo reference:show --constants <REF>
```

- INI entries

Feature not provided by version 2.26

Version 3.2.

```
$ phpcompatinfo reference:show --ini <REF>
```

And if you want to filter results on PHP version, do for example :

Version 2.26.

```
$ phpcompatinfo list --reference=ALL classes <REF> --filter-version="php_5.1.0" --filter
```

Version 3.2.

```
$ phpcompatinfo reference:show --classes <REF> --php=">= 5.1.0"
```

11.2. Print parses results

Where <SOURCE> identify the data source, directly in CompatInfo 2.26, and via the JSON configuration file in version 3.2

Version 2.26.

```
$ phpcompatinfo print --reference=ALL --recursive <SOURCE>
or
$ phpcompatinfo print --reference=ALL --recursive --report summary <SOURCE>
```

Version 3.2.

```
$ phpcompatinfo analyser:run <SOURCE>
or
$ phpcompatinfo analyser:run <SOURCE> summary
```

And with additionals reports :

- extension

Version 2.26.

```
$ phpcompatinfo print --reference=ALL --recursive --report extension <SOURCE>
```

Version 3.2.

```
$ phpcompatinfo analyser:run <SOURCE> extension
```

- namespace

Version 2.26.

```
$ phpcompatinfo print --reference=ALL --recursive --report namespace <SOURCE>
```

Version 3.2.

```
$ phpcompatinfo analyser:run <SOURCE> namespace
```

- trait

Version 2.26.

```
$ phpcompatinfo print --reference=ALL --recursive --report trait <SOURCE>
```

Version 3.2.

```
$ phpcompatinfo analyser:run <SOURCE> trait
```

- interface

Version 2.26.

```
$ phpcompatinfo print --reference=ALL --recursive --report interface <SOURCE>
```

Version 3.2.

```
$ phpcompatinfo analyser:run <SOURCE> interface
```

- class

Version 2.26.

```
$ phpcompatinfo print --reference=ALL --recursive --report class <SOURCE>
```

Version 3.2.

```
$ phpcompatinfo analyser:run <SOURCE> class
```

- function

Version 2.26.

```
$ phpcompatinfo print --reference=ALL --recursive --report function <SOURCE>
```

Version 3.2.

```
$ phpcompatinfo analyser:run <SOURCE> function
```

- constant

Version 2.26.

```
$ phpcompatinfo print --reference=ALL --recursive --report constant <SOURCE>
```

Version 3.2.

```
$ phpcompatinfo analyser:run <SOURCE> constant
```

- global

Version 2.26.

```
$ phpcompatinfo print --reference=ALL --recursive --report global <SOURCE>
```

Feature not provided by version 3.2

- condition

Version 2.26.

```
$ phpcompatinfo print --reference=ALL --recursive --report condition <SOURCE>
```

Version 3.4.

```
$ phpcompatinfo analyser:run <SOURCE> condition
```

- token

Version 2.26.

```
$ phpcompatinfo print --reference=ALL --recursive --report token <SOURCE>
```

Feature not provided by version 3.2

- xml

Version 2.26.

```
$ phpcompatinfo print --reference=ALL --recursive --report xml <SOURCE>
```

Feature not provided by version 3.2

- source

Version 2.26.

```
$ phpcompatinfo -v print --reference=ALL --report source <SOURCE>
```

Version 3.2.

```
$ phpcompatinfo provider:display <SOURCE>
```

11.3. Caching results

- Version 2.26 may cache results to speed-up later analysis.
- Version 3.2 is able (optionally) to cache results only with other SAPI than CLI.
- Version 3.3 is able to cache results on all API including CLI.

Chapter 12. Configuration file

12.1. Global options

- File extensions was restricted by default in both versions to `php`, `inc` and `html`.

In CompatInfo 3.2, the Finder recursively traverse directories, while it's not true in version 2.26

XML configuration 2.26.

```
<?xml version="1.0" encoding="utf-8" ?>
<phpcompatinfo
    recursive="false"
    fileExtensions="php, inc, phtml"
    >

    <!-- ... -->
</phpcompatinfo>
```

JSON configuration 3.2.

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

- Progress bar

XML configuration 2.26.

```
<?xml version="1.0" encoding="utf-8" ?>
<phpcompatinfo
    consoleProgress="true"
    verbose="false"
    >

    <!-- ... -->
</phpcompatinfo>
```

Use the first verbose level (`-v`) with `phpcompatinfo` while running the `analyser:run` command.

- Caching results

XML configuration 2.26.

```
<?xml version="1.0" encoding="utf-8" ?>
<phpcompatinfo
    cacheDriver="file"
    >
```

```
    <!-- ... -->
</phpcompatinfo>
```

Version 3.2 is able to cache parsing results only with other SAPI than CLI. See the Developer Guide.

12.2. Cache options

XML configuration 2.26.

```
<?xml version="1.0" encoding="utf-8" ?>
<phpcompatinfo>

    <cache id="file">
        <options>
            <gc_probability>1</gc_probability>
            <gc_maxlifetime>86400</gc_maxlifetime>
        </options>
    </cache>

</phpcompatinfo>
```

Version 3.2 does not provide yet ability to cache parsing results in CLI mode.

12.3. References options

XML configuration 2.26.

```
<?xml version="1.0" encoding="utf-8" ?>
<phpcompatinfo>

    <references>
        <reference name="Core" />
        <reference name="standard" />
    </references>

</phpcompatinfo>
```

Version 3.2 does not provide ability to load reference depending of rules in the configuration file.

All references are either pre-loaded (Prefetch Strategy) or loaded only when detected (AutoDiscover Strategy).

See Chapter 19, *References included*

12.4. PHP settings

XML configuration 2.26.

```
<?xml version="1.0" encoding="utf-8" ?>
<phpcompatinfo>

    <php>
        <ini name="memory_limit" value="140M" />
        <ini name="short_open_tag" />
        <ini name="zend.ze1_compatibility_mode" value="false" />
    </php>

</phpcompatinfo>
```

Version 3.2 does not provide ability to change PHP settings at run-time.

12.5. Excluding Files or Elements from parsing

XML configuration 2.26.

```
<?xml version="1.0" encoding="utf-8" ?>
<phpcompatinfo>

    <excludes>
        <exclude id="demo">
            <directory name=".*\/Zend\/.*" />
            <file name=".*\.php5" />
            <extension name="xdebug" />
            <interface name="SplSubject" />
            <trait name="^S" />
            <class name=".*Compat.*" />
            <function name="ereg.*" />
            <function name="debug_print_backtrace" />
            <constant name="T_USE" />
        </exclude>
    </excludes>

</phpcompatinfo>
```

Version 3.2 does not provide ability to exclude elements (class, trait, …), but you can exclude files or directories with the Finder. See `source-providers` in the JSON configuration file. See Chapter 8, *The Json Configuration File*

12.6. Listeners

XML configuration 2.26.

```
<?xml version="1.0" encoding="utf-8" ?>
<phpcompatinfo>

    <listeners>
        <listener class="className" file="/path/to/filename">
            <arguments>
            </arguments>
```

```
        </listener>
    </listeners>

</phpcompatinfo>
```

Version 3.2 provide this feature with the Symfony EventDispatcher component.

See Section 8.2, “Plugins”

12.7. Plugins options

XML configuration 2.26.

```
<?xml version="1.0" encoding="utf-8" ?>
<phpcompatinfo>

    <plugins>
        <reference name="MyReference"
            class="PEAR_CompatInfo"
            file="/path/to/PEARCompatInfo.php">
            <arguments>
            </arguments>
        </reference>
    </plugins>

</phpcompatinfo>
```

Version 3.2 does not provide ability to select a custom References list.

All references are either pre-loaded (Prefetch Strategy) or loaded only when detected (AutoDiscover Strategy).

See Chapter 19, *References included*

Chapter 13. Server API

13.1. Parsing recursive directories

By default,

- `recursive` option is set to `false` in CompatInfo 2.26, while the Finder recursively traverse directories in version 3.2
- `cacheDriver` option is set to `file` in CompatInfo 2.26, while version 3.2 did not added the cache plugin.

Version 2.26.

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

$source  = '/path/to/source';
$options = array(
    'cacheDriver' => 'null',
    'recursive'   => true
);

$compatinfo = new PHP_CompatInfo($options);
$compatinfo->parse($source);
```

Version 3.2.

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

use Bartlett\CompatInfo;

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

use Symfony\Component\Finder\Finder;

$dirs = '/path/to/source';

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

$provider = new SymfonyFinderProvider($finder);

$pm = new ProviderManager;
$pm->set('dataSourceIdent', $provider);

$compatinfo = new CompatInfo;
$compatinfo->setProviderManager($pm);
$compatinfo->parse();
```

13.2. Using cache feature

Version 2.26.

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

$source  = '/path/to/source';
$options = array(
    'cacheDriver' => 'file',
    'recursive'   => true
);

$compatinfo = new PHP_CompatInfo($options);
$compatinfo->parse($source);
```

It’s not mandatory to specify `cacheDriver` option what is by default set to false.

Version 3.2.

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

use Bartlett\CompatInfo;

use Bartlett\Reflect\ProviderManager;
use Bartlett\Reflect\Provider\SymfonyFinderProvider;
use Bartlett\Reflect\Plugin\Cache\CachePlugin;
use Bartlett\Reflect\Plugin\Cache\DefaultCacheStorage;
use Bartlett\Reflect\Cache\DoctrineCacheAdapter;

use Doctrine\Common\Cache\FilesystemCache;

use Symfony\Component\Finder\Finder;

$dirs = '/path/to/source';

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

$provider = new SymfonyFinderProvider($finder);

$pm = new ProviderManager;
$pm->set('dataSourceIdent', $provider);

$backend = new FilesystemCache(sys_get_temp_dir() . '/phpcompatinfo');
$doctrineCache = new DoctrineCacheAdapter($backend);
$cache = new DefaultCacheStorage($doctrineCache);

$compatinfo = new CompatInfo;
$compatinfo->setProviderManager($pm);
$compatinfo->addPlugin( new CachePlugin($cache) );
$compatinfo->parse();
```

13.3. Listeners

While CompatInfo 2.26 audit all events (does not provide ability to filter them, unless by writing a new listener), version 3.2 let you choose and connect a function by event. See Chapter 15, *Build your Plugins*

Version 2.26.

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

$source  = '/path/to/source';
$options = array(
    'cacheDriver' => 'null',
    'recursive'   => true
);

$fileListener = new PHP_CompatInfo_Listener_File();

$compatinfo = new PHP_CompatInfo($options);
$compatinfo->attach($fileListener);
$compatinfo->parse($source);
```

Version 3.2.

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

use Bartlett\CompatInfo;

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

use Symfony\Component\Finder\Finder;

$dirs = '/path/to/source';

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

$provider = new SymfonyFinderProvider($finder);

$pm = new ProviderManager;
$pm->set('dataSourceIdent', $provider);

$compatinfo = new CompatInfo;
$compatinfo->setProviderManager($pm);

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

```
        );
    }
);
$compatinfo->parse();
```

13.4. Exploring parsing results

Version 2.26.

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

$source  = '/path/to/source';
$options = array(
    'cacheDriver' => 'null',
    'recursive'   => true
);

$compatinfo = new PHP_CompatInfo($options);
$compatinfo->parse($source);

$versions   = $compatinfo->getVersions();
$classes    = $compatinfo->getClasses();
$functions  = $compatinfo->getFunctions();
$extensions = $compatinfo->getExtensions();
```

Version 3.2.

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

use Bartlett\CompatInfo;
use Bartlett\CompatInfo\Analyser;

use Bartlett\Reflect\ProviderManager;
use Bartlett\Reflect\Provider\SymfonyFinderProvider;
use Bartlett\Reflect\Plugin\Analyser\AnalyserPlugin;

use Symfony\Component\Finder\Finder;

$dirs = '/path/to/source';

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

$provider = new SymfonyFinderProvider($finder);

$sourceId = 'dataSourceIdent';

$pm = new ProviderManager;
$pm->set($sourceId, $provider);

$compatinfo = new CompatInfo;
$compatinfo->setProviderManager($pm);
$compatinfo->addPlugin(
    new AnalyserPlugin(
```

```
        array(
            new Analyser\SummaryAnalyser(),
        )
    )
);
$compatinfo->parse();

$metrics = $compatinfo->getMetrics();

$versions   = $metrics[$sourceId]['sa.versions'];
$classes    = $metrics[$sourceId]['sa.classes'];
$functions  = $metrics[$sourceId]['sa.functions'];
$extensions = $metrics[$sourceId]['sa.extensions'];
```

`sa.` prefix corresponds to class constant METRICS_PREFIX of SummaryAnalyser.

Part IV. Developer Guide

 **API**

CompatInfo comes with a complete API.

See Chapter 14, *API*

 **Plugins**

CompatInfo 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 15, *Build your Plugins*

 **Analysers**

CompatInfo 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 Chapter 18, *Build your Analysers*

Chapter 14. API

14.1. Data Source Identification

Identify the Data Source with the Symfony Finder [http://symfony.com/doc/current/components/finder.html] Component.

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.

CompatInfo uses the Reflect data source provider mechanism. You may either use the basic Symfony Finder, what we will do next, or use your own.

The `Provider` is a component from Reflect, and not from CompatInfo. Take care of namespace !

14.1.1. Basic Provider with Symfony Finder

```
<?php

use Bartlett\Reflect\Provider\SymfonyFinderProvider;
use Symfony\Component\Finder\Finder;

$dirs = dirname(__DIR__) . '/sources';

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

$provider = new SymfonyFinderProvider($finder);
```

At this step, we have created a data source provider that is allowed to retrieve each element to parse.

CompatInfo need to know it. We attach then the previous provider instance to a **Provider Manager**, with a label (e.g: `Single`) to identify it easily.

Reflect Provider Manager with a unique data source.

```
<?php

use Bartlett\Reflect\ProviderManager;

$pm = new ProviderManager;
$pm->set('Single', $provider);
```

A **Provider Manager** may provide one or more data source identifications. Equivalent to the `source-providers` section of the `phpcompatinfo.json` configuration file for the command-line interface.

Reflect Provider Manager with multiple data sources.

```
<?php

use Bartlett\Reflect\ProviderManager;
use Bartlett\Reflect\Provider\SymfonyFinderProvider;
use Symfony\Component\Finder\Finder;

$pm = new ProviderManager;

// -- source 1
$source1 = dirname(__DIR__) . '/sources/';

$finder1 = new Finder();
$finder1->files()
    ->name('sample1.php')
    ->in($source1);

$pm->set('Sample', new SymfonyFinderProvider($finder1));

// -- source 2
$pharFile = dirname(__DIR__) . '/sources/pirus.phar';
$source2  = 'phar://' . $pharFile;

$finder2 = new Finder();
$finder2->files()
    ->path('/Pirus/')
    ->name('*.php')
    ->in($source2);

$pm->set('Pirus', new SymfonyFinderProvider($finder2));
```

On this example CompatInfo is able to parse contents of two data sources: `Sample` and `Pirus`, all at once (default behavior) or individually.

14.2. Parse elements of the provider

We reuse the provider manager instance (`$pm`) seen above (unique data source named `Single`). Then we ask CompatInfo to parse its full contents.

```
<?php

use Bartlett\CompatInfo;

$compatinfo = new CompatInfo;
$compatinfo->setProviderManager($pm);
$compatinfo->parse();
```

In case of multiple data sources, when you want to parse it individually rather than fully, use the following statements.

Parse only Data Source named `Pirus`.

```
<?php

use Bartlett\CompatInfo;

$compatinfo = new CompatInfo;
$compatinfo->setProviderManager($pm);
```

```
$compatinfo->parse(array('Pirus'));
```

`Pirus` is the data source label used on `$pm#set()` statement.

You have identified data sources and parsed its full contents. Now you are ready to handle the results.

14.3. Using analysers

To explore and exploit results, we need first to parse the data source, and connect at least one analyser.

On following example, we will use the standard `SummaryAnalyser`, but you are free to use any other analysers available and even your owns.

Each analyser, should at least, implement the `Bartlett\Reflect\Analyser\AnalyserInterface` and provide a `getMetrics()` method (to return results).

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

use Bartlett\CompatInfo;
use Bartlett\CompatInfo\Analyser;

use Bartlett\Reflect\ProviderManager;
use Bartlett\Reflect\Provider\SymfonyFinderProvider;
use Bartlett\Reflect\Plugin\Analyser\AnalyserPlugin;

use Symfony\Component\Finder\Finder;

$dirs = '/path/to/source';

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

$provider = new SymfonyFinderProvider($finder);

$sourceId = 'dataSourceIdent';

$pm = new ProviderManager;
$pm->set($sourceId, $provider);

$compatinfo = new CompatInfo;
$compatinfo->setProviderManager($pm);
$compatinfo->addPlugin(
    new AnalyserPlugin(
        array(
            new Analyser\SummaryAnalyser(),
        )
    )
);
$compatinfo->parse();

$metrics = $compatinfo->getMetrics();
```

```
$versions   = $metrics[$sourceId]['sa.versions'];
$classes    = $metrics[$sourceId]['sa.classes'];
$functions  = $metrics[$sourceId]['sa.functions'];
$extensions = $metrics[$sourceId]['sa.extensions'];
```

`sa.` prefix corresponds to class constant METRICS_PREFIX of SummaryAnalyser.

Chapter 15. Build your Plugins

15.1. Events

CompatInfo 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 and CompatInfo components to communicate with each other by dispatching events and listening to them.

15.1.1. Event Dispatcher

CompatInfo 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.

15.1.2. 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\CompatInfo;

$compatinfo = new CompatInfo;

$ed = $compatinfo->getEventDispatcher();
```

15.1.3. Adding Event Listeners

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

Example 15.1. Add a listener that will echo out files when they are parsed

```
<?php

use Bartlett\CompatInfo;
use Symfony\Component\EventDispatcher\GenericEvent;

$compatinfo = new CompatInfo;

$compatinfo->getEventDispatcher()->addListener(
    'reflect.progress',
```

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

Example 15.2. Add a listener that will exploit each AST [http://en.wikipedia.org/wiki/Abstract_syntax_tree] of file parsed

```
<?php

use Bartlett\CompatInfo;
use Symfony\Component\EventDispatcher\GenericEvent;

$compatinfo = new CompatInfo;

$compatinfo->getEventDispatcher()->addListener(
    'reflect.success',
    function (GenericEvent $e) {
        $ast = unserialize($e['ast']);
        printf(
            'Parsing Data source "%s", file "%s". AST = %s' . PHP_EOL,
            $e['source'],
            $e['file']->getPathname(),
            print_r($ast, true)
        );
    }
);
```

15.1.4. 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.

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

```
<?php
class AnalyserPlugin implements EventSubscriberInterface
{
    public static function getSubscribedEvents()
    {
        return array(
            'reflect.complete' => 'onReflectComplete',
        );
    }
}
```

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

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

15.1.5. 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 `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.cache	A previous cached request was found and return the AST.	`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

15.2. Console Commands

If your plugin should be accessible on the command line, and provides some new commands, you have to register them with the static `getCommands()` method.

Have a look on AnalyserPlugin, that provide two new commands: `analyser:list` and `analyser:run`.

```
<?php
```

```
class AnalyserPlugin implements EventSubscriberInterface
{
    public static function getCommands()
    {
        $commands   = array();
        $commands[] = new AnalyserListCommand;
        $commands[] = new AnalyserRunCommand;

        return $commands;
    }
```

If your plugin must not provide console command, your `getCommands()` static method should return an empty php array.

Chapter 16. Cache Plugin

16.1. Register Plugin

You can used this plugin (from PHP Reflect) only since CompatInfo 3.2

```
<?php

use Bartlett\CompatInfo;

use Bartlett\Reflect\Plugin\Cache\CachePlugin;

$compatinfo = new CompatInfo;
$compatinfo->addPlugin( new CachePlugin($cache) );
```

Where `$cache` is an instance of object that must implement interface `Bartlett\Reflect\Plugin\Cache\CacheStorageInterface`.

Use the `Bartlett\Reflect\Plugin\Cache\DefaultCacheStorage` object unless you want to change the cache storage behavior.

16.2. Doctrine Adapter

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

```
<?php

use Bartlett\CompatInfo;

use Bartlett\Reflect\Plugin\Cache\CachePlugin;
use Bartlett\Reflect\Plugin\Cache\DefaultCacheStorage;
use Bartlett\Reflect\Cache\DoctrineCacheAdapter;

use Doctrine\Common\Cache\FilesystemCache;

$doctrineCache = new DoctrineCacheAdapter($backend);

$cache = new DefaultCacheStorage($doctrineCache);

$compatinfo = new CompatInfo;
$compatinfo->addPlugin( new CachePlugin($cache) );
```

Where `$backend` is an instance of object that must implement interface `Doctrine\Common\Cache\CacheProvider`.

16.3. File cache

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

```
<?php

use Bartlett\CompatInfo;

use Bartlett\Reflect\Plugin\Cache\CachePlugin;
use Bartlett\Reflect\Plugin\Cache\DefaultCacheStorage;
use Bartlett\Reflect\Cache\DoctrineCacheAdapter;

use Doctrine\Common\Cache\FilesystemCache;

$backend = new FilesystemCache(sys_get_temp_dir() . '/bartlett/cache');

$doctrineCache = new DoctrineCacheAdapter($backend);

$cache = new DefaultCacheStorage($doctrineCache);

$compatinfo = new CompatInfo;
$compatinfo->addPlugin( new CachePlugin($cache) );
```

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).

Chapter 17. Log Plugin

17.1. Register Plugin

You can used this plugin (from PHP Reflect) only since CompatInfo 3.4

```
<?php

use Bartlett\CompatInfo;
use Bartlett\Reflect\Plugin\Log\LogPlugin;

// Optional plugin configuration
$opt = array();

$compatinfo = new CompatInfo;
$compatinfo->addPlugin( new LogPlugin($logger, $opt) );
```

Where `$logger` is an instance of object that must implement interface `Psr\Log\LoggerInterface` (PSR-3 [http://www.php-fig.org/psr/psr-3/]).

And `$opt` is an array to configure what events and its details you would like to have.

Event	Log Level	Message Template (with/ without placeholders)
reflect.progess	Psr3\Log\Level\LogLevel::INFO	Parsing file "{file}" in progress.
reflect.success	Psr3\Log\Level\LogLevel::INFO	AST built.
reflect.cache	Psr3\Log\Level\LogLevel::INFO	AST built by a previous request.
reflect.error	Psr3\Log\Level\LogLevel::ERROR	Parser has detected an error on file "{file}". "{error}".
reflect.complete	Psr3\Log\Level\LogLevel::NOTICE	Parsing data source "{source}" completed.

For example, if you want to deactivate logging on `reflect.success` event, then give the following options :

```
<?php

$opt = array(
    'reflect.success' => false,
);
```

Or, if you don’t want to have contextual data sent to logger :

```
<?php
```

```
$opt = array(
    'reflect.success' => array(
        'level'    => LogLevel::INFO,
        'template' => 'AST built.',
        'context'  => false,
    ),
);
```

17.2. Using your private logger

Use your own logger, that must be compatible PSR-3.

```
<?php

use Bartlett\CompatInfo;
use Bartlett\Reflect\Plugin\Log\LogPlugin;

use Psr\Log\AbstractLogger;

class YourLogger extends AbstractLogger
{
    private $channel;

    public function __construct($name = 'YourLoggerChannel')
    {
        $this->channel = $name;
    }

    public function log($level, $message, array $context = array())
    {
        error_log(
            sprintf(
                '%s.%s: %s',
                $this->channel,
                strtoupper($level),
                $this->interpolate($message, $context)
            )
        );
    }

    protected function interpolate($message, array $context = array())
    {
        // build a replacement array with braces around the context keys
        $replace = array();
        foreach ($context as $key => $val) {
            $replace['{' . $key . '}'] = $val;
        }

        // interpolate replacement values into the message and return
        return strtr($message, $replace);
    }
}

// Create the main logger
$logger = new YourLogger('CompatInfo');

// Optional plugin configuration
$opt = array();
```

```
$compatinfo = new CompatInfo;
$compatinfo->addPlugin( new LogPlugin($logger, $opt) );
```

17.3. Using Monolog

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

```
<?php

use Bartlett\CompatInfo;
use Bartlett\Reflect\Plugin\Log\LogPlugin;

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

// Create some handlers
$stream = new StreamHandler('/var/logs/phpcompatinfo.log');

// Create the main logger
$logger = new Logger('CompatInfo');
$logger->pushHandler($stream);

// Optional plugin configuration
$opt = array();

$compatinfo = new CompatInfo;
$compatinfo->addPlugin( new LogPlugin($logger, $opt) );
```

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

```
<?php

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

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

Chapter 18. 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.

18.1. Visitor pattern

Each `Analyser` class must implement interface `Bartlett\Reflect\Visitor\VisitorInterface`.

Abstract visitor is a component of Reflect, and not CompatInfo. Take care of namespace !

```
<?php

namespace Bartlett\Reflect\Visitor;

use Bartlett\Reflect\Model\Visitable;

interface VisitorInterface
{
    public function visit(Visitable $visitable);
}
```

Each element that need to be explored by your analyser should have a visit method accordingly.

- For packages, we need to implement a **visitPackageModel** method.
- For classes, we need to implement a **visitClassModel** method.
- For properties, we need to implement a **visitPropertyModel** method.
- For methods, we need to implement a **visitMethodModel** method.
- For functions, we need to implement a **visitFunctionModel** method.
- For constants, we need to implement a **visitConstantModel** method.
- For includes, we need to implement a **visitIncludeModel** method.
- For dependencies, we need to implement a **visitDependencyModel** method.

Abstract class `Bartlett\Reflect\Visitor\AbstractVisitor`, that implement interface `Bartlett\Reflect\Visitor\VisitorInterface`, holds a basic visitor.

```
<?php

use Bartlett\Reflect\Visitor\AbstractVisitor;

class Analyser extends AbstractVisitor
{
    public function visitPackageModel($package)
```

```
    {
    }

    public function visitClassModel($class)
    {
    }

    public function visitMethodModel($method)
    {
    }

    public function visitPropertyModel($property)
    {
    }

    public function visitFunctionModel($function)
    {
    }

    public function visitConstantModel($constant)
    {
    }

    public function visitIncludeModel($include)
    {
    }

    public function visitDependencyModel($dependency)
    {
    }
}
```

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

use Bartlett\Reflect\Analyser\AbstractAnalyser;

class Analyser extends AbstractAnalyser
{
}
```

18.2. Print results

Once you have used visit methods to explore parsing results, you will need to return formatted data ready to be print.

To do so, you should implement the `render()` method of `Bartlett\Reflect\Analyser\AnalyserInterface`. :leveloffset: 0

Chapter 19. References included

Statistics v2

- 2.0.0 support 61 references
- 2.1.0 support 63 references
- 2.2.0 support 65 references
- 2.3.0 support 67 references
- 2.5.0 support 75 references
- 2.8.0 support 80 references
- 2.10.0 support 83 references
- 2.13.0 support 84 references
- 2.15.0 support 86 references
- 2.16.0 support 95 references
- 2.23.0 support 98 references
- 2.25.0 support 99 references
- 2.26.0 support 100 references

Statistics v3

- 3.0.0 support 100 references
- 3.3.0 support 102 references

Prefetch	AutoDiscover	Reference	CompatInfo
☐	☑	amqp 1.4.0 stable	2.8.0
☐	☑	apc 3.1.13 beta	2.0.0
☑	☐	apcu 4.0.7 beta	2.16.0
☑	☐	bcmath php5	2.0.0
☑	☐	bz2 php5	2.0.0

Prefetch	AutoDiscover	Reference	CompatInfo
☑	☐	calendar php5	2.0.0
☑	☐	Core php5	2.0.0
☐	☑	ctype php5	2.0.0
☐	☑	curl php5	2.0.0
☑	☐	date php5	2.0.0
☑	☐	dom 20031129 stable	2.0.0
☐	☑	enchant 1.1.0 stable	2.0.0
☑	☐	ereg php5	2.0.0
☐	☑	exif php5	2.5.0
☑	☐	fileinfo 1.0.5 stable	2.0.0
☑	☐	filter 0.11.0 stable	2.0.0
☐	☑	ftp php5	2.0.0
☑	☐	gd php5	2.0.0
☐	☑	gender 1.0.0 stable	2.16.0
☐	☑	geoip 1.1.0 beta	2.8.0
☑	☐	gettext php5	2.0.0

Prefetch	AutoDiscover	Reference	CompatInfo
☐	☑	gmp php5	2.0.0
☐	☑	haru 1.0.4 stable	2.16.0
☐	☑	hash php5	2.0.0
☐	☑	htscanner 1.0.1 stable	2.23.0
☐	☑	http 2.1.4 stable	2.16.0
☐	☑	iconv php5	2.0.0
☐	☑	igbinary 1.2.1 stable	2.10.0
☐	☑	imagick 3.1.2 stable	2.10.0
☑	☐	imap php5	2.0.0
☐	☑	inclued 0.1.3 beta	2.8.0
☑	☐	intl php5	2.0.0
☐	☑	jsmin 1.0.0 stable	2.25.0
☐	☑	json 1.2.1 stable	2.0.0
☐	☑	ldap php5	2.2.0
☑	☐	libevent 0.1.0 beta	2.16.0
☐	☑	libxml php5	2.0.0

Prefetch	AutoDiscover	Reference	CompatInfo
☐	☑	lzf 1.6.2 stable	2.5.0
☑	☐	mailparse 2.1.6 stable	2.5.0
☑	☐	mbstring php5	2.0.0
☑	☐	mcrypt php5	2.0.0
☐	☑	memcached 2.2.0 stable	2.1.0
☐	☑	memcache 3.0.8 beta	2.1.0
☐	☑	mhash php5	2.0.0
☑	☐	mongo 1.5.8 stable	2.8.0
☑	☐	msgpack 0.5.5 beta	2.16.0
☑	☐	mssql php5	2.5.0
☐	☑	mysql 1.0 stable	2.0.0
☐	☑	mysqli 0.1 stable	2.0.0
☐	☑	OAuth 1.2.3 stable	2.2.0
☑	☐	odbc php5	2.10.0
☑	☐	openssl php5	2.0.0
☑	☐	pcntl php5	2.0.0

Prefetch	AutoDiscover	Reference	CompatInfo
☑	☐	pcre php5	2.0.0
☑	☐	pdflib 3.0.4 stable	2.23.0
☐	☑	PDO 1.0.4dev stable	2.0.0
☑	☐	pgsql php5	2.0.0
☐	☑	phar 2.0.2 stable	2.0.0
☐	☑	posix php5	2.0.0
☑	☐	pthreads 2.0.10 stable	2.16.0
☐	☑	rar 3.0.2 stable	2.23.0
☑	☐	readline 2.0.1 stable	2.0.0
☐	☑	recode 2.0.1 stable	2.0.0
☐	☑	Reflection php5	2.3.0
☐	☑	riak 1.2.0 stable	2.26.0
☐	☑	session php5	2.0.0
☐	☑	shmop php5	2.0.0
☐	☑	SimpleXML 0.1 stable	2.0.0
☐	☑	snmp php5	2.0.0

Prefetch	AutoDiscover	Reference	CompatInfo
☑	☐	soap php5	2.0.0
☑	☐	sockets php5	2.0.0
☐	☑	solr 2.0.0 stable	2.5.0
☑	☐	sphinx 1.3.2 stable	2.5.0
☑	☐	spl 0.2 stable	2.0.0
☐	☑	sqlite3 0.7-dev stable	2.0.0
☐	☑	sqlite 2.0-dev stable	2.0.0
☐	☑	ssh2 0.12 beta	2.0.0
☑	☐	standard php5	2.0.0
☐	☑	stomp 1.0.6 stable	2.16.0
☐	☑	svn 1.0.2 stable	2.13.0
☐	☑	sync 1.0.1 stable	3.3.0
☑	☐	sysvmsg php5	2.0.0
☑	☐	sysvsem php5	2.0.0
☑	☐	sysvshm php5	2.0.0
☐	☑	tidy 2.0 stable	2.0.0

Prefetch	AutoDiscover	Reference	CompatInfo
☑	☐	tokenizer 0.1 stable	2.0.0
☐	☑	uploadprogress 1.0.3.1 stable	2.16.0
☐	☑	varnish 1.1.1 stable	2.15.0
☐	☑	wddx php5	2.0.0
☑	☐	XCache 3.2.0 stable	2.8.0
☐	☑	xdebug 2.2.6 stable	2.0.0
☐	☑	xhprof 0.9.4 beta	2.5.0
☑	☐	xml php5	2.0.0
☐	☑	xmlreader 0.1 stable	2.0.0
☐	☑	xmlrpc 0.51 stable	2.0.0
☐	☑	xmlwriter 0.1 stable	2.0.0
☑	☐	xsl 0.1 stable	2.0.0
☐	☑	yac 0.9.2 beta	3.3.0
☐	☑	yaml 1.1.1 stable	2.5.0
☑	☐	Zend OPcache 7.0.4-devFE beta	2.15.0
☐	☑	zip 1.12.4 stable	2.3.0

Prefetch	AutoDiscover	Reference	CompatInfo
☑	☐	zlib 2.0 stable	2.0.0