# Plop

***Release***

December 28, 2014

Contents


**1 Prerequisites** **3**
1.1 Getting started . . . 3
1.2 Running Plop from a PHAR . . . 3
1.3 Validating your PHP installation . . . 4

**2 Installation** **7**
2.1 Installation using a PHAR archive . . . 7
2.2 Installation using Composer . . . 8
2.3 Installation from sources . . . 9

**3 Usage** **11**
3.1 Loading Plop's classes . . . 11
3.2 Configuring Plop . . . 11
3.3 Logging some messages . . . 14

**4 Contributing** **17**
4.1 Code/patch . . . 17

**5 Licenses** **19**
5.1 Plop . . . 19

Plop is a logging library loosely based on Python's logging module and therefore bears similarities with it. The code underwent major changes though, so as to make it easier to use with Dependency Injection Containers. The API makes heavy use of chainable method calls, making it very intuitive.

Contents:

CHAPTER 1

# Prerequisites

This page assumes that the reader has a working PHP setup (either installed using some distribution's package manager or manually) and lists the dependencies required to use Plop. In case you compiled PHP yourself, you may need to recompile it to include additional extensions (see the list of required PHP dependencies in the section entitled Getting started for more information).

Plop is known to work with most PHP versions. Plop should run correctly on both Windows (XP or later) and Linux (most distros). The code is tested using an automated process on Windows Vista (64 bits), Windows 7 (64 bits), Windows 8.1 (64 bits), Debian Stable (64 bits) and CentOS 6 (64 bits), as reflected by our Continuous Integration server.


- Getting started
- Running Plop from a PHAR
- Validating your PHP installation


## 1.1 Getting started

To use Plop in your project, you need PHP 5.3.3 or later, compiled with the following extensions:

- pcre
- sockets
- SPL

## 1.2 Running Plop from a PHAR

If you want to use Plop from a PHP ARchive (phar), the following additional extensions are required:

- openssl
- Phar
- SimpleXML

## 1.3 Validating your PHP installation

You can check whether your PHP installation satisfies all the prerequisites listed above by running the following commands, which will display information about the PHP version and list all currently enabled extensions:

```
me@home:~$ php -v # Check PHP version
PHP 5.4.33 (cli) (built: Sep 25 2014 23:41:02) (DEBUG)
Copyright (c) 1997-2014 The PHP Group
Zend Engine v2.4.0, Copyright (c) 1998-2014 Zend Technologies

me@home:~$ php -m # Check available extensions
[PHP Modules]
bcmath
bz2
calendar
Core
ctype
date
dom
ereg
gd
gettext
gmp
iconv
intl
json
libxml
mbstring
mysql
mysqli
mysqlnd
openssl
pcntl
pcre
PDO
pdo_mysql
pdo_sqlite
Phar
posix
readline
Reflection
session
SimpleXML
soap
sockets
SPL
sqlite3
standard
sysvmsg
sysvsem
sysvshm
tokenizer
xdebug
xml
xmlreader
xmlwriter
xsl
zip
```

```
zlib
```

You may also consult the output of `phpinfo()` for the same purpose.

CHAPTER 2

# Installation

This pages contains instructions on how to install Plop on your machine. There are several ways to achieve that. Each method is described below.


- Installation using a PHAR archive
- Installation using Composer
- Installation from sources


**Note:** We recommend using the PHAR installation method or the composer installation method, depending on whether your project already uses Composer or not.

## 2.1 Installation using a PHAR archive

A PHAR archive is simply a way of bundling all the necessary files in one big file.

Installing Plop as a PHAR archive only involves a few steps:

1. Make sure your installation fulfills all of the prerequisites.

   **Note:** As all of Plop's PHAR archives (core and modules) are digitally signed, you must make sure the OpenSSL extension is enabled on your PHP installation. Failure to do so will result in an error when trying to run Plop's PHAR archive.

2. Download the PHAR archive for Plop. You can grab the latest version from https://pear.erebot.net/get/Plop-latest.phar. You **MUST** also download the public signature for the archive. The signature for the latest version is available at https://pear.erebot.net/get/Plop-latest.phar.pubkey.

**Note:** The whole installation process using PHAR archives can be automated using the following commands:

```
$ wget https://packages.erebot.net/get/Plop-latest.phar      \
    https://packages.erebot.net/get/Plop-latest.phar.pubkey
```

**Warning:** Even though the command above should work on most installations, a few known problems may occur due to incompatibilities with certain PHP features and extensions. To avoid such issues, it is usually a good idea to check the following items:

- Make sure `detect_unicode` is set to `Off` in your `php.ini`. This is especially important on MacOS where this setting tends to be set to `On` for a default PHP installation.
- If you applied the Suhosin security patch to your PHP installation, make sure `phar` is listed in your `php.ini` under the `suhosin.executor.include.whitelist` directive.
- Please be aware of certain incompatibilities between the Phar extension and the ionCube Loader extension. To use Plop from a PHAR archive, you will need to remove the following line from your `php.ini`:

  ```
  zend_extension = /usr/lib/php5/20090626+lfs/ioncube_loader_lin_5.3.so
  ```

  (the path and versions may be different for your installation).

3. Check that the installation was successful by running the following command:

   ```
   $ php -f Plop-latest.phar
   ```

   (replace `Plop-latest.phar` with the actual name of the PHAR archive you just downloaded in case it was different)

   The command should return without any error. If error messages are issued, try to fix your installation using the information given by those messages.

You may now proceed to the *next step*, which makes actual use of Plop's logging capabilities.

## 2.2 Installation using Composer

Composer is a simple dependency resolver / package manager aimed at PHP 5.3.0 or later. Their website contains extensive documentation on how to use it in your project to handle dependencies.

With that in mind, using composer to install Plop is very simple and only involves the following steps:

1. Install Composer on your machine:

   ```
   me@home:~$ curl -s http://getcomposer.org/installer | php
   ```

2. Create a file named `composer.json` in your current directory.
3. Copy/paste the following snippet in that file and save:

   ```
   {
     "require": {
       "Erebot/Plop": "*"
     }
   }
   ```

4. Let composer do the rest:

   ```
   me@home:~$ php composer.phar install
   ```

You may now proceed to the *next step*, which makes actual use of Plop's logging capabilities.

## 2.3 Installation from sources

First, make sure a git client is installed on your machine. Under Linux, **from a root shell**, run the command that most closely matches the tools provided by your distribution:

```
# For apt-based distributions such as Debian or Ubuntu
$ apt-get install git

# For yum-based distributions such as Fedora / RHEL (RedHat)
$ yum install git

# For urpmi-based distributions such as MES (Mandriva)
$ urpmi git
```

**Note:** Windows users may be interested in installing Git for Windows to get an equivalent git client. Also, make sure that **git** is present on your account's `PATH`. If not, you'll have to replace **git** by the full path to `git.exe` on every invocation (e.g. **"C:Program FilesGitbingit.exe" clone ...**)

Also, make sure you have all the required dependencies installed as well. Now, retrieve Plop's code from its repository, using the following command:

```
$ git clone --recursive git://github.com/Erebot/Plop.git
```

You may now proceed to the *next step*, which makes actual use of Plop's logging capabilities.

CHAPTER 3

# Usage

Using Plop usually involves 3 steps, detailed below.


- Loading Plop's classes
- Configuring Plop
  - Loggers
  - Filters
  - Handlers
  - Formatters
- Logging some messages


TL;DR: simply read the section on Loading Plop's classes then skip all the way down to the section on Logging some messages.

## 3.1 Loading Plop's classes

The way to load Plop's classes depends on the installation method selected:

- For the PHAR installation method, add this snippet near the top of your main PHP file:

  ```
  require_once('path/to/Plop-latest.phar');
  ```

  (adjust the path with the name of the PHAR archive you downloaded)

- For the Composer installation method, add this snippet instead near the top of your main PHP file:

  ```
  require_once('path/to/vendor/autoload.php');
  ```

- For an installation from sources,

  ```
  require_once('src/Autoloader.php');
  \Plop\Autoloader::register();
  ```

If you're not interested in fine-tuning Plop, you may skip the rest of this page until you reach the part about Logging some messages.

## 3.2 Configuring Plop

There are 4 different types of objects that you can configure in Plop. Each one is described below.

### 3.2.1 Loggers

A logger intercepts log messages for a given method, class, file or directory. This is decided at construction time based on the arguments passed to **:api:'Plop::Logger::__construct'**.

Internally, Plop builds up a hierarchy of loggers like so:

```
root-level logger
    namespace-level logger
        class-level logger
            method/function-level logger
```

Log messages "bubble up". That is, Plop first looks for a method or function-level logger to handle the message. If none can be found, it looks for a class-level logger (in case the message was emitted from a method). Then it looks for a namespace-level logger, then a logger for the parent's namespace, etc. until it reaches the root-level logger, which always exists.

Whichever logger is found first will be the one to handle the message.

---

**Note:** The root-level logger) comes pre-configured with a handler that logs messages to `STDERR` using some very basic formatting.

---

Several aspects of a logger can be configured, such as:

- The logging level. Whenever a message is received whose level is lower than the logger's logging level, the message is ignored, **but** no other logger will be called to handle the message (effectively preventing the message from bubbling further).
- The record factory. This factory is used to create records of logging messages, intended to keep track of the message's contextual information. This factory must implement the **:api:'Plop::RecordFactoryInterface'** interface and is usually an instance of **:api:'Plop::RecordFactory'**.
- *Filters*.
- *Handlers*.

Once a logger has been created and configured, you can tell Plop about it, using the following code snippet:

```
$logging = \Plop\Plop::getInstance();
$logging[] = $newlyCreatedLogger;
```

This will add the logger to the list of loggers already known to Plop. If a logger had already been registered in Plop with the same "identity" (ie. the same namespace, class and method names), it will automatically be replaced with the new one.

**See also:**

**:api:'Plop::LoggerInterface'** Detailed API documentation on the interface implemented by loggers.

**:api:'Plop::LoggerAbstract'** An abstract class that can be useful when implementing your own logger.

**:api:'Plop::IndirectLoggerAbstract'** An abstract class that can be useful when implementing an indirect logger. An indirect logger is a logger which relies on another logger to work. Plop's main class (**:api:'Plop'**) is an example of such a logger.

**:api:'Plop::Logger'** The most common type of logger.

**:api:'Plop::Psr3Logger'** A logger that supports the PSR-3 `\Psr\Log\LoggerInterface` interface.

### 3.2.2 Filters

Filters are associated with either *loggers* or *handlers* through an object implementing **:api:'Plop::FiltersCollectionInterface'** (usually an instance of **:api:'Plop::FiltersCollection'**) and are used to restrict which messages will be handled. They are applied once the message has been turned into a log record and work by defining various criteria such a record must respect.

If a record respects all of the criteria given in the collection, the *handlers* associated with the logger are called in turn to do their work.

**Note:** The "level" associated with a logger acts like a lightweight filter. In fact, the same effect could be obtained by defining a collection containing an instance of **:api:'Plop::Filter::Level'** with the level desired.

**Warning:** Not all handlers make use of filters. Therefore, depending on the handlers used, it is possible that the filters will be ignored entirely.

To associate a new filter with a logger or handler, use the following code snippet:

```
$filters = $logger_or_handler->getFilters();
$filters[] = $newFilter;
```

Please note that this will **not** replace existing filters. Records will still have to pass the previous filters, but they will also have to pass the new filter before they can be handled.

**See also:**

**:api:'Plop::FiltersCollectionInterface'** Detailed API documentation for the interface representing a collection of filters.

**:api:'Plop::FilterInterface'** Detailed API documentation for the interface implemented by all filters. This page also references all the filters that can be used in a collection.

### 3.2.3 Handlers

Handlers are associated with *loggers* through an object implementing **:api:'Plop::HandlersCollectionInterface'** (usually an instance of **:api:'Plop::HandlersCollection'**) and are used to define the treatment applied to log records.

Various types of handlers exist that can be used to log message to different locations such as the system's event logger (syslog), a (rotated) file, a network socket, ...

Like with loggers, several aspects of a handler can be configured:

- *Its associated formatter*.
- *Filters*.

To associate a new handler with a logger, use the following code snippet:

```
$handlers = $logger->getHandlers();
$handlers[] = $newHandler;
```

Please note that this will **not** replace existing handlers. Also, both the previously defined handlers and the newly added one will be called when a log record must be handled.

**See also:**

**:api:'Plop::HandlersCollectionInterface'** Detailed API documentation for the interface representing a collection of handlers.

**:api:'Plop::HandlerAbstract'** An abstract class that can be useful when implementing a new handler.

**:api:'Plop::HandlerInterface'** Detailed API documentation for the interface implemented by all handlers. This page also references all the handlers that can be used in a collection.

### 3.2.4 Formatters

Each *handler* has an associated formatter, which is used when a record needs formatting. A formatter defines how the final message will look like.

There are a few things about a formatter that you can configure:

- The main format. This string serves as a pattern for the final message.

  When using an instance of **:api:'Plop::Formatter'** with default settings as the formatter, it may contain Python-like string formats using the syntax for dictionaries.

  That is, it may contain something like the following:

  ```
  [%(asctime)s] %(levelname)s - %(message)s
  ```

  The default format in that case is defined in **:api:'Plop::Formatter::DEFAULT_FORMAT'**.

  Several pre-defined formats exist that depend on the particular implementation used to represent records. For example, **:api:'Plop::Record'** closely follows the formats defined by Python's logging module whenever they are applicable.

- The format for dates and times.

  When using an instance of **:api:'Plop::Formatter'** as the formatter, it uses the formatting options from PHP's date() function. Also, the default format for dates and times is then defined in **:api:'Plop::Formatter::DEFAULT_DATE_FORMAT'**.

- The current timezone as a DateTimeZone object. This information is used when formatting dates and times for log records that were created in a timezone that does not match the local timezone.

To associate a new formatter with a handler, use the following code snippet:

```
$handler->setFormatter($newFormatter);
```

Please note that this **will** replace any formatter previously in place.

**See also:**

**:api:'Plop::FormatterInterface'** Detailed API documentation for the interface implemented by all formatters.

**:api:'Plop::Formatter'** The most common implementation of formatters.

**:api:'Plop::Record'** The most common implementation for log records.

**http://php.net/class.datetime.php#datetime.constants.types** PHP's predefined constants to represent several popular types of date/time formatting.

**http://php.net/timezones.php** List of timezone identifiers supported by PHP.

## 3.3 Logging some messages

Logging messages with Plop usually only involves the following sequence:

```
// First, grab an instance of Plop.
// Plop uses a singleton pattern, so the same instance will be returned
// every time you use this method, no matter where you're calling it from.
$logging = \Plop\Plop::getInstance();
```

```
// Now, send a log.
// Various log levels are available by default:
// debug, info, notice, warning, error, critical, alert & emergency.
// There's a method named after each log level's name.
$logging->debug('Hello world');
```

Log messages may contain variables, which will be replaced with their actual value when the logging method is called. This is useful in a lot of situations. For example, you can use it to apply I18N (Internationalization) methods to the messages:

```
$logging = \Plop\Plop::getInstance();
$logging->error(
    _('Sorry %(nick)s, now is not the time for that!'),
    array(
        'nick' => 'Ash',
    )
);
```

CHAPTER 4

# Contributing

So, you took interest in Plop and would like to contribute back? This is the right page!

There are actually several ways by which would may contribute to the project:

- by reporting new issues (or asking for new features)
- by forking the code and sending pull requests

Whichever one it is, you may also join our IRC channel to discuss issues, new ideas / feature requests and follow Plop's development.

## 4.1 Code/patch

If you plan on sending patches (or pull requests), please read our documentation on the coding standard used by Plop's developers first. Your patch will have greater chances to be approved if it abides by that standard when you submit it.

To contribute a patch, you will need a GitHub account. Then you can simply:

- Fork the code to your own account.
- Create a new branch.
- Patch things up as much as you want.
- Create a pull request with your changes.

Once your pull request has been received, it will undergo a review process to decide whether it can be accepted as-is, needs more changes before having a chance to be accepted or is utterly rejected.

CHAPTER 5

# Licenses

## 5.1 Plop

The documentation (both the API documentation and the end-user documentation) for Plop is released under the CC BY-NC-SA license (see http://creativecommons.org/licenses/by-sa/3.0/).

The code itself is released under the GNU General Public License.

```
Copyright © 2010-2014 François Poirotte

Plop is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

Plop is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with Plop.  If not, see <http://www.gnu.org/licenses/>.
```

Badges:

## E

environment variable
    PATH, 9

## P

PATH, 9