
Comodojo extender Documentation

Release 1.0.0

Marco Giovinazzi

Dec 04, 2018

Contents

1	Installing extender	3
1.1	Requirements	3
1.2	Finalizing installation	3
2	Basic configuration	5
3	Running extender	7
4	Creating Tasks	9
4.1	Anatomy of a task	9
4.2	Registering a task	9
4.3	Tasks bundles	9
5	Scheduling jobs	11
6	Defining commands	13
6.1	Writing additional commands	13
6.2	Command Bundles	14
7	Plugins	17

Comodojo extender is database driven, multiprocess (pseudo) cron task scheduler in PHP

Table of Contents:

CHAPTER 1

Installing extender

The comodojo extender framework can be installed using `composer` as a product (using the dedicated `extender project package`) or as a library.

To install it as a product:

```
composer create-project comodojo/extender extender
```

Or, to install it as a library in your own project:

```
composer require comodojo/extender.framework
```

1.1 Requirements

To work properly, `comodojo/extender.framework` requires PHP `>=5.6.0` (cli enabled).

Following PHP extensions are also required:

- `ext-posix`: PHP interface to *nix Process Control Extensions
- `ext-pcntl`: process Control support in PHP
- `ext-shmop`: read, write, create and delete Unix shared memory segments
- `ext-sockets`: low-level interface to the socket communication functions

A database is not required but highly recommended; by default, extender creates a new SQLite3 database if no external database is specified.

1.2 Finalizing installation

TBW

CHAPTER 2

Basic configuration

TBW

CHAPTER 3

Running extender

TBW

Tasks are the core components of extender framework.

A task is a self-contained PHP script that does some work and returns a brief string as result. It can be called from one (or more) jobs or directly from econtrol.

4.1 Anatomy of a task

TBW

4.2 Registering a task

TBW

4.3 Tasks bundles

TBW

CHAPTER 5

Scheduling jobs

TBW

Defining commands

Every command that `econtrol` provides is an independent, parametrizable PHP script. The `extender.commandsbundle.default` package contains basic commands used to interact with the framework.

Commands can be defined by user into the `EXTENDER_COMMAND_FOLDER` or packed in bundles and installed directly via composer.

6.1 Writing additional commands

A command is essentially a class that implements the `ComodojoExtenderCommandCommandInterface`. The `ComodojoExtenderCommandAbstractCommand` abstract class can be useful to avoid common methods definition (is, essentially, a trait defined as a class for compatibility reasons).

Note: Take a look at the [api](#) to know all the method that your command should implement.

Supposing to extend the `ComodojoExtenderCommandAbstractCommand` class, a command should only implement the `execute()` method.

Let's take an "hello world" example.:

```
<?php namespace My\Command;

class HelloWorldCommand extends \Comodojo\Extender\Command\AbstractCommand {

    public function execute() {

        // the getOption() method can be used to retrieve options provided to the ↵
        ↵command
        $test = $this->getOption("test");

        // same for the arguments, with getArgument()
        $to = $this->getArgument("to");

        $to = is_null($to) ? "World" : $to;

        // the color object can be used to add colors to command's output
```

(continues on next page)

(continued from previous page)

```
        return $this->color->convert("\n%gHello " . $to . "!\n");
    }
}
```

Once defined, a command should be registered into the framework. The *extender-commands.yaml* file can be used for this purpose.

The format is the following:

```
helloworld:
  package: none
  data:
    class: My\Command\HelloWorldCommand
    description: Greetings from comodojo extender
    aliases:
      - hw
    options:
      test:
        short_name: -t
        long_name: --test
        action: StoreTrue
        description: Void command option
    arguments:
      to:
        choices: [ ]
        multiple: false
        optional: true
        description: hello to...
```

Note: Extender relies from [pear/console_commandline](#) package to handle command line operations. Take a look at package's documentation to know more.

6.2 Command Bundles

Creating a bundle of commands is quite easy.

First, let's take a look at the (proposed) directory structure of a package:

```
mybundle/
- commands/
  - helloworldcommand.php
  - customcommand.php
- composer.json
```

Commands' classes should be autoloaded (using composer); in addition, something should be written in *extender-commands-config.php* file. The project package does all the job automatically using **extra** field of *composer.json*.

To enable this feature, the package's type **should** be declared as *extender-commands-bundle* or *comodojo-bundle* and the *extra* field should contain a *comodojo-commands-register* or (preferably) a *extender-command-register* subfield.

So, the *composer.json* of *mybundle* package will be something like:

```
{
  "name": "my/mybundle",
  "description": "My first commands bundle",
```

(continues on next page)

(continued from previous page)

```
"type": "extender-commands-bundle",
"extra": {
    "extender-command-register": {
        "helloworld": {
            "class": "My\\Command\\Helloworld",
            "description": "Greetings from comodojo extender",
            "aliases": ["hw"],
            "options": {
                "force": {
                    "short_name": "-t",
                    "long_name": "--test",
                    "action": "StoreTrue",
                    "description": "Void command option"
                }
            },
            "arguments": {
                "to": {
                    "choices": {},
                    "multiple": false,
                    "optional": true,
                    "description": "hello to..."
                }
            }
        }
    }
},
"autoload": {
    "psr-4": {
        "My\\Command\\": "commands"
    }
}
```

Once installed, every should be in place to exec those commands using:

```
./econtrol.php helloworld Marvin
```


CHAPTER 7

Plugins

TBW