PhpZone Documentation

Release 0.2

Jakub Zapletal

April 26, 2015

Contents

CHAPTER 1

Getting Started

1.1 Requirements

PhpZone requires PHP 5.3 or higher.

1.2 Installation

Installation is provided via Composer, if you don't have it, do install:

```
$ curl -s https://getcomposer.org/installer | php
```

then PhpZone can be added into your dependencies by:

```
$ composer require --dev phpzone/phpzone 0.2.*
```

or add it manually into your `composer.json`:

```
{
    "required-dev": {
        "phpzone/phpzone": "0.2.*"
    }
}
```

1.3 Configuration file

The configuration file `phpzone.yml` is the alpha and omega of this tool and its format is YAML.

1.3.1 Creating the configuration file

Default location of the config file is a root of a project where PhpZone should be used.

You can create `phpzone.yml` manually or run:

```
$ vendor/bin/phpzone --init
```

which would automatically create the `phpzone.yml` in the project folder.

Note: If the `phpzone.yml` already exists, it will **not** be overwritten.

1.3.2 Custom path

There is also provided an option for the custom path. You can just basically use:

```
$ vendor/bin/phpzone --config path/to/config.yml
```

1.3.3 Definitions

The file can contain none or each of the following definitions:

imports	Optional	Including another files.
extensions	Optional	Register all required extensions and their configurations included

One example rules them all:

```
imports:
    - { resource: relative/path/to/file_1.yml } # key "resource" is required!
    - { resource: relative/path/to/another/file_2.yml }
extensions:
    Namespace\Foo\ClassFoo: ~ # simple registration of an extension without any value
    Namespace\Bar\ClassBar:
        some_key: some_value
    Namespace\Baz\ClassBaz:
        - value 1 of an array
        - value 2 of an array
```

Important: Every extension has it's own configuration values and their structure depends on the specification of the extension. For more details follow instructions according the extension.

CHAPTER 2

Basic Commands

2.1 List of commands

There is a command to display all available commands:

```
$ vendor/bin/phpzone
# or
$ vendor/bin/phpzone list
```

2.2 Initialize config file

In case of new project or new implementation it can be useful to let PhpZone generate the configuration file by:

```
$ vendor/bin/phpzone --init
```

Note: If the `phpzone.yml` already exists, it will **not** be overwritten.

2.3 Custom config path

There is also provided an option for the custom path for the configuration file. You can just basically use:

```
$ vendor/bin/phpzone --config path/to/config.yml
# or
$ vendor/bin/phpzone -c path/to/config.yml
```

2.4 Shell environment

Shell environment provides an interactive environment with full support of history and auto-complete commands. Very useful when there are more defined commands and the developer often switches between them.

```
$ vendor/bin/phpzone --shell
# or
$ vendor/bin/phpzone -s
```

2.5 Help

Help can be called for general application or for specific command. It will show all available arguments, options or help description of command if defined.

```
$ vendor/bin/phpzone help <COMMAND>
# or
$ vendor/bin/phpzone -h <COMMAND>
```

CHAPTER 3

Official Extensions

3.1 PhpZone Docker

A Docker command builder configured by YAML. Its primary purpose is to provide a simple way to define commands for running Docker containers/instances which could be used in daily workflow of every developer. Since now not all developers need to have a knowledge about Docker but still everyone can simply understand what is running. It is not only about the knowledge but also experienced developers can find an advantage in keeping ready-made commands.

More details ...

3.2 PhpZone Shell

A command/script builder configured by YAML. Its primary purpose is to provide an easy way to define multiple scripts used in daily workflow of every developer.

More details ...

CHAPTER 4

Creating An Extension

4.1 Dependencies

PhpZone is based on Symfony components without rapid custom modifications. All these dependencies are automatically downloaded via Composer. Thanks to that it is easily possible to create own extensions just by following official a documentation of Symfony components.

Dependencies:

Symfony Config	symfony/config	~2.3
Symfony Console	symfony/console	~2.3
Symfony Debug	symfony/debug	~2.3
Symfony DependencyInjection	symfony/dependency-injection	~2.3
Symfony EventDispatcher	symfony/event-dispatcher	~2.3
Symfony Yaml	symfony/yaml	~2.3

More information coming soon...

PhpZone is a generic tool for the easy creation of YAML configured console applications. Its primary purpose is to provide a centralized automation tool for developers to simplify development workflow. **Basically it is a wrapper around commands to provide a unified command line tool.**

Note: As it's built on Symfony components without rapid custom modifications, it can be used as an application skeleton for any individual commands.

Attention: Its power is based on simplicity of centralized configuration via YAML and main value comes from extensions.

CHAPTER 5

Basic Usage

An example speaks a hundred words so let's go through one.

Create a `phpzone.yml` file in the root of a project:

```
extensions:
    PhpZone\Shell\Shell: # register an extension with a configuration
        tests:
            - vendor/bin/behat
            - vendor/bin/phpunit
            - vendor/bin/phpspec
```

and run:

```
$ vendor/bin/phpzone tests
```

As you would expect, the configuration contains the definition for the command `tests` and when you run it, all defined sub-commands will be executed.

Important: The `PhpZone\Shell\Shell` extension is not a part of the `phpzone/phpzone` package, but an aside project based on PhpZone. More info in a chapter dedicated to official extensions.