comodojo/comodojo-installer Documentation

Release 1.0

Marco Giovinazzi

Jan 06, 2019

Contents:

The comodojo-installer package provides auto-configuration capabilities to the dispatcher and the extender frameworks. As every comodojo component relies on composer, this package is designed as a composer plugin, installed as a dependency and activated by internal composer events.

CHAPTER 1

Installation

To add it in personal projects, the first step is to add a specific requirement in the composer.json of the project package:

```
"require": {
    // other libraries
    // ...
    "comodojo/comodojo-installer" : "^1.0"
}
```

The same file should include a specific configuration section in the *extra* field like:

```
"extra": {
    "comodojo-installer": {
        "package-types": [
            "comodojo-bundle"
        ],
        "global-config": {
            "extra-field": "comodojo-configuration",
            "persistence": "\\Comodojo\\Installer\\Persistence\\YamlPersistence",
            "params": {
                "config-file": "config/comodojo-configuration.yml",
                "depth": 6
            }
        },
        "package-extra": {
            "routes": {
                "driver": "\\Comodojo\\Installer\\Drivers\\RouteDriver",
                "persistence": "\\Comodojo\\Installer\\Persistence\\YamlPersistence
↪",
                "params": {
                    "config-file": "config/comodojo-routes.yml"
                }
            }
        }
    }
}
```

More details about configuration statements are availble in general-configuration.

1.1 Requirements

To work properly, comodojo/comodojo-installer requires PHP >=5.6.0 and composer-plugin-api > 1.0

CHAPTER 2

General Concepts

Comodojo project packages, like comodojo/dispatcher or comodojo/extender, are designed to streamline the configuration of frameworks and libraries, automating the build and customization processes. In this way dispatcher can create the routing table dynamically, extender can do the same for tasks.

Considering dispatcher services as an example, these components can be placed inside the project package directly, relative routes can be hardcoded inside default configuration files and the final package can be stored and deployed as a single artifact. In a small installation, perhaps, this could be the right way to organize a project.

But what if your app is composed by hundreds of services? You will need something to help you to build it from small, manageable packages that could be installed independently. And since composer supports a plugin architecture that can be used to extend its capabilities, the comodojo/comodojo-installer package is designed as a plugin to enhance the way composer builds a project, enabling the auto-installation of packages and avoiding internal collisions.

2.1 Installer workflow

The comodojo/comodojo-installer packages is defined as a composer-plugin. Once installed, it extends the behaviour of the routines that define package lifecycle management and project installation.

It hooks to the `post-create-project-cmd` composer event first, to create the main configuration and allow custom scripts to be executed. Then, it listen for every package that is installed, updated or removed and activate itself only when the package-type is recognized as *manageable* (see next section for more informations). When this happens, installer read the `extra` field inseide *composer.json* and process its statements using drivers (to interpret them) and persisters (to save config data somewhere).

Note: If you have cloned your project from github, the installer will manage single packages but will not create main configuration since the `post-create-project-cmd` event will never be raised. You will need launch it manually using the command:

```
composer run-script post-create-project-cmd
```

2.1.1 Comodojo packages

TBW

CHAPTER 3

Configuration

The installer reads configuration statements from:

- project package, to configure itself and to build initial config;
- custom libraries installed by composer, to build application piece by piece.

Note: The default project package is **comodojo-bundle**.

3.1 Project statements

There are two section of the extra field that installer will look for: `comodojo-installer` and `comodojo-configuration`. The first provides configuration for the installer itself, the second for the entire project.

As example, let's dissect the **'comodojo/dispatcher default configuration'_** and, in particular, it's extra field.

It contains the two sections described above:

```
{
    "extra": {
        "comodojo-installer": {...},
        "comodojo-configuration": {...}
    }
}
```

comodojo-installer section

This section will configure the installer and can be used to extend its functionalities.

To better undestand each statement, let's look at the commented version of the **'comodojo/dispatcher default configuration'_**:

```
"comodojo-installer": {
    // what package types installer will look for?
    "package-types": [
        "comodojo-bundle"
    ],
    // this subsection tells installer how to manage the global configuration
    "global-config": {
        // the extra-field where look for configuration statements (see next␣
↪topic)
        "extra-field": "comodojo-configuration",
        // how the configuration will be persisted
        "persistence": "\\Comodojo\\Installer\\Persistence\\YamlPersistence",
        // parameters to pass to the persister
        "params": {
            "config-file": "config/comodojo-configuration.yml",
            "depth": 6
        }
    },
    // this subsection instructs installer to search for specific extra
    // field when a package is recognized as manageable (package-type in␣
↪[package-types])
    "package-extra": {
        // this defines that each valid package could include a routes field␣
↪that will be used to
        // build the routing table of the dispatcher
        "routes": {
            // once found, route statements are processed by a RouteDriver...
            "driver": "\\Comodojo\\Installer\\Drivers\\RouteDriver",
            // ...and persisted using the YamlPersistence class...
            "persistence": "\\Comodojo\\Installer\\Persistence\\YamlPersistence",
            // ...using this parameters.
            "params": {
                "config-file": "config/comodojo-routes.yml"
            }
        },
        "plugins": {
            "driver": "\\Comodojo\\Installer\\Drivers\\PluginDriver",
            "persistence": "\\Comodojo\\Installer\\Persistence\\YamlPersistence",
            "params": {
                "config-file": "config/comodojo-plugins.yml"
            }
        },
        "commands": {
            "driver": "\\Comodojo\\Installer\\Drivers\\CommandDriver",
            "persistence": "\\Comodojo\\Installer\\Persistence\\YamlPersistence",
            "params": {
                "config-file": "config/comodojo-commands.yml"
            }
        }
    }
}
```

comodojo-configuration section

The second section contains the `global-config`, the project's default configuration that will be persisted during project installation and then loaded according to project rules.

```
{
    "comodojo-configuration": {
        "static-config": "config",
        "routing-table-cache": true,
        "routing-table-ttl": 86400,
        "log": {
            "enable": true,
            "name": "dispatcher",
            "providers": {
                "local" : {
                    "type": "StreamHandler",
                    "level": "info",
                    "stream": "logs/dispatcher.log"
                }
            }
        },
        "cache": {
            "enable": true,
            "pick_mode": "PICK_FIRST",
            "providers": {
                "local": {
                    "type": "Filesystem",
                    "cache_folder": "cache"
                }
            }
        }
    }
}
```

CHAPTER 4

Drivers

TBW

CHAPTER 5

Persistence

TBW