# comodojo/comodojo-installer Documentation

***Release 1.0***

**Marco Giovinazzi**

**Apr 06, 2019**

# Contents:

The comodojo-installer package provides auto-configuration capabilities to the dispatcher and the extender frameworks. Since 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

As a composer plugin, this library must be included as a direct dependency in a root (project) package.

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

Once installed, the plugin is immediately active and starts to scan the composer.json file looking for a custom configuration (in the *extra* section).

Following an example configuration in the composer.json.

```
"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"
                }
            }
        }
    }
}
```

**See also:**

More details in general-configuration.

## 1.1 Requirements

To work properly, comodojo/comodojo-installer requires:

- PHP >= 5.6.0
- 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. As a result, dispatcher can create the routing table dynamically, extender can do the same for tasks.

As an example, let's consider the definition of routes into dispatcher.

Without an automated installation process, dispatcher services could be defined inside the project package, in a dedicate folder structure accessible using internal composer autoloader. Relative routes have to be hardcoded inside configuration files and the final package can be stored, versioned 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 application is composed by hundreds of services? And, if it uses plugins, how to reuse code across different installation?

That's where the comodojo-installer comes in, enabling the auto-installation of packages and avoiding internal collisions. In other words, this package enables the installation of application's components (e.g. dispatcher services, plugins, extender tasks) as independent bundles, building a complete application from small, manageable and independent packages.

And since composer supports a plugin architecture that can be used to extend its capabilities, the comodojo/comodojo-installer package is designed as a composer plugin to enhance the way it builds a project.

## 2.1 Installer workflow

Since composer supports a plugin architecture that can be used to extend its capabilities, the comodojo/comodojo-installer package is designed as a composer plugin to enhance the way it builds a project. Once installed, it extends the routines that define package lifecycle management and project installation.

First, it hooks to the `post-create-project-cmd` composer event, to create the main configuration and to allow custom scripts to be executed.

Then, it listens for every package that is installed, updated or removed and activates itself when the package-type is recognized as *manageable* (see section *Comodojo packages* for more information). When this happens, installer reads the `extra` field inside new package's *composer.json* and processes its statements using *drivers*, to interpret them, and *persisters*, to save config data somewhere (form more information, see sections *Drivers* and *Persistence*).

**Note:** If you have cloned your project from github, the installer will manage packages but will not create main configuration since the `post-create-project-cmd` event will not be emitted. To launch it manually run:

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

### 2.1.1 Comodojo packages

TBW

CHAPTER 3

# Configuration

The comodojo-installer library reads configuration statements from two locations: project package and manageable bundles.

**Project package**

The composer.json of the project package is the main location that contains both installer and framework configuration.

The installer configuration is the first one that the installer reads, and it's used to configure its features (e.g. main configuration file name, supported bundles).

The project configuration is evaluated at the end of the project installation, when the `post-create-project-cmd` composer event is emitted, and it provides the configuration for the entire project.

**External bundles**

The composer.json of external bundles contains only the information about application components that will be installed including the package in the project.

For example, the metadata needed to auto-configure routes into comodojo/dispatcher is in the bundle's composer.json file, under a specific configuration *stanza* in the *extra* field and following a specific pattern (for more information, see section *Drivers*).

## 3.1 Installer Configuration

This section is used to configure the comodojo/installer and to extend its functionalities.

The configuration *stanza* inside the *extra* field is fixed and cannot be changed:

```
{
    "extra": {
        // this section name CANNOT change
```

(continues on next page)

(continued from previous page)

```
        "comodojo-installer": {...},
        // this section name CAN change
        "comodojo-configuration": {...}
    }
}
```

To understand different statements in this section, let's look at 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"
            }
        }
    }
}
```

## 3.2 Project Configuration

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.

This section depends primarily on the project itself and the framework(s) behind it.

As an example, let's consider the **'comodojo/dispatcher project configuration'_**:

```
{
    "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