

Artisan : The PrestaShop Code Generator



Artisan : The PrestaShop Code Generator is an intelligent code generator for PrestaShop 1.7+ to speed up your development process. This tool will help developers generate resources like modules, controllers, grid, forms, routes, and services. It is extremely flexible and customizable to cover many on the use cases.

The author and contributors have made their best to make this module as functional and stable as possible. Nevertheless, before using it, it is your responsibility to run prior tests to make sure it corresponds to your needs and that you understand its functioning.

To install **Artisan : The PrestaShop Code Generator** module, you must first have purchased the rights to use. Any use of this module without prior purchase of the rights to use is considered as an offence and liable to criminal prosecution.

Prerequisites

Artisan requires PrestaShop from version 1.7.5 The following server environment is required to use the module : PHP 5.6.0 or above

Software quality

Artisan : The PrestaShop Code Generator is validate by the PrestaShop Validator with no error.

The module complies with PrestaShop's coding standards. The module is technically compatible with the software and meets PrestaShop community's way of coding.

Disclaimer

It is highly recommended to backup your server files and database before installing this module. No responsibility can be taken for any adverse effects installation or advice may cause. Recommended you install on a test server initially to carry out your own testing.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Module Overview

If you're a developer, you can use this artisan to develop PrestaShop theme and module.

Features, Highlights and Capabilities

- Create PrestaShop module
- Create legacy Prestashop controllers
- Create modern symfony CRUD controllers (1.7.6+)
- Create model from existing table in database
- Create modern symfony form (1.7.6+)
- Generate multi-language applications
- Create full resources using a single command
- Create very clean, reusable and highly readable code to build on
- Extracts the strings to translate from a module and creates the translation files
- etc.

All commands must be launched from a command interpreter and from the prestashop installation root folder.

List of commands

```
php bin/console artisan:configuration:delete {name}
php bin/console artisan:configuration:set {name} {value}
php bin/console artisan:module:create {moduleName}
php bin/console artisan:module:model:create {moduleName} {modelName}
php bin/console artisan:module:admin:form:create {moduleName} {modelName}
php bin/console artisan:module:admin:legacycontroller:create {moduleName} {modelName}
php bin/console artisan:module:admin:crudcontroller:create {moduleName} {modelName}
php bin/console artisan:module:install {moduleName}
php bin/console artisan:module:uninstall {moduleName}
php bin/console artisan:module:zip {moduleName}
php bin/console artisan:module:list [--active|--not-active|--installed|--not-installed]
php bin/console artisan:module:translation:update {moduleName} --subdomain [Admin|Shop]
php bin/console artisan:module:hook:list {moduleName}
php bin/console artisan:module:hook:register {moduleName} {hooks}
php bin/console artisan:module:hook:unregister {moduleName} {hooks}
php bin/console artisan:override:disable
php bin/console artisan:override:enable
php bin/console artisan:override:class {className} {functionName}
php bin/console artisan:shop:disable
php bin/console artisan:shop:enable
php bin/console artisan:sql:load {filePathAndName}
php bin/console artisan:sql:snapshot {snapshotName}
php bin/console artisan:sql:restore {snapshotName}
php bin/console artisan:theme:create:child {themeName}
php bin/console artisan:debug:disable
php bin/console artisan:debug:enable
```

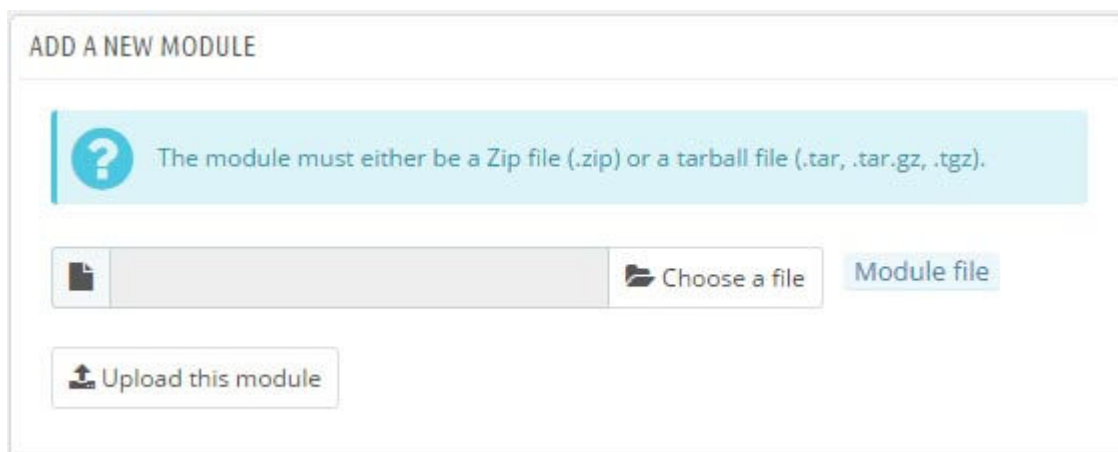
Installation

Once the module is downloaded as a Zip archive (artisan.zip), the installation process goes through 2 steps:

Step 1 - Files installation on the server

Automatic installation with the dedicated form

To install the module "automatically" click the link "Add a new module" at the top of the list of modules. The following form should then display:



- Click "Choose File". You must select the zip archive of the module, not the its file or one of the unzipped files.
- Click on "Upload this module" to confirm.

Once validated, PrestaShop will load the module from your computer, unzip it, place the files in the right place and update the page.

Manually installing

- Decompress (unzip) the archive file of module. You should get a new folder.
- With your file manager software, place the new folder in the PrestaShop modules directory.

Be careful not to transfer this file in the folder of another module (which can occur when using drag and drop). Move the whole folder, not just the files it contains.

Step 2 - Module installation

You can install this module from a shell using the symfony console provided with PrestaShop.

```
php bin/console prestashop:module install artisan
```

Troubleshooting

Installation

If the module does not appear in your list of modules, ensure that you do not have active filters switched on. Your list could prevent the module appearing.

Some hostings don't allow extracted zip files. If so you will receive an error alert "Invalid zip file" or similar. In that case you need to move the extracted archive content throw file manager to your PrestaShop installation folder "modules"

Commands

This tool provides lots of commands through the Symfony console. You must execute it in the terminal from the root of your PrestaShop installation.

```
$php bin/console artisan:module:list --active
```

List of commands

The list of commands may vary depending on the version of the artisan module.

Configuration

- **artisan:configuration:delete** to delete configuration value
- **artisan:configuration:set** to set configuration value

Debug

- **artisan:debug:disable** to disable debug mode
- **artisan:debug:enable** to enable debug mode

Module

- **artisan:module:list** list active, not active, installed and not installed modules
- **artisan:module:install** install the module and its translations
- **artisan:module:uninstall** uninstall the module and its translations
- **artisan:module:zip** creates a zip archive of a module
- **artisan:module:create** create a PrestaShop module
- **artisan:module:admin:form:create** create a Modern back-office form
- **artisan:module:admin:legacycontroller:create** create a legacy CRUD back-office controller
- **artisan:module:admin:crudcontroller:create** create a Symfony CRUD back-office controller
- **artisan:module:model:create** create a Model (ObjectModel)
- **artisan:module:hook:list** list of hooks used by the module
- **artisan:module:hook:register** register a module onto one or more hooks
- **artisan:module:hook:unregister** unregister a module from one or more hooks

Override

- **artisan:override:disable** disable override
- **artisan:override:enable** enable override
- **artisan:override:class** override a function of the prestashop framework

Sql

- **artisan:sql:load** load and execute sql from file
- **artisan:sql:snapshot** take a snapshot of the database
- **artisan:sql:restore** restore the database from a snapshot

Shop

- **artisan:shop:disable** set the shop in maintenance mode

- **artisan:shop:enable** *exit the maintenance mode and enable the shop*

Theme

- **artisan:theme:create:child** *create a PrestaShop child theme*

artisan:debug:disable

This command is used to **disable debug mode**.

Syntax

```
php bin/console artisan:debug:disable
```

How it works

The artisan will set the *PS_MODE_DEV* variable to FALSE in the */config/defines.inc.php* file.

No parameters are saved.

Generated files

none

artisan:debug:enable

This command is used to **enable debug mode**.

Syntax

```
php bin/console artisan:debug:enable
```

How it works

The artisan will set the *PS_MODE_DEV* variable to TRUE in the */config/defines.inc.php* file.

No parameters are saved.

Generated files

none

artisan:module:list

This command list active, not active, installed and not installed modules.

Syntax

```
php bin/console artisan:module:list [--active|--not-active|--installed|--not-installed]
```

- *--active* list active modules
- *--not-active* list not active modules
- *--installed* list installed modules
- *--not-installed* list not installed modules

How it works

The artisan lists the modules according to the parameter.

No parameters are saved.

Generated files

none

artisan:module:install

This command install the module and its translations.

Syntax

```
php bin/console artisan:module:install {moduleName}
```

- *{moduleName}* is the name of the module.

How it works

The artisan copy module translations files in app/Resources/translations and install the module.

No parameters are saved.

Generated files

none

artisan:module:uninstall

This command **uninstall** the module and its translations.

Syntax

```
php bin/console artisan:module:uninstall {moduleName}
```

- *{moduleName}* is the name of the module.

How it works

The artisan uninstall the module and remove module translations files from app/Resources/translations.

No parameters are saved.

Generated files

none

artisan:module:create

This command is used to create a PrestaShop module.

Syntax

```
php bin/console artisan:module:create {moduleName}
```

- If the module already exists, some files can be updated.

How it works

The artisan asks some questions :

- Company name for module namespace? Answer with the name of the company
- Module hook List? Answer with the list of hook name separated by commas
- Implement widget Interface? Answer with Yes or No
- Create templates for content hooks? Answer with Yes or No

Then, the artisan uses your answers to create the PrestaShop module {moduleName} .

Parameters are saved in the file module_(moduleName) and in the section[module].

Generated files

In the module directory:

- index.php
- composer.json
- Readme.md
- .gitignore
- {moduleName}.php

In /config

- index.php
- services.yml
- routes.yml

In /config/services

- index.php

In /config/routes

- index.php

In /src

- index.php

Generated files (Optionnal)

Only if you have answer Yes to the question "Generate templates for content hooks ?"

In /views

- index.php

In /views/templates

- index.php

In /views/templates/hook

- index.php
- {hookName 1}.tpl
- {hookName 2}.tpl
- {hookName ...}.tpl

artisan:module:admin:form:create

This command is used to create a **Modern back-office form**.

Syntax

```
php bin/console artisan:module:admin:form:create {moduleName} {modelName}
```

- The module must exist.
- The modele must exist. (The model class must be accessible).

How it works

The artisan analyzes the model specify in parameters to automatically create the form to edit a record.

Parameters are saved in the file form_(ModelName) and in the section[controller].

Generated files

In src/Form/Admin:

- {modelName}FormType.php
- {modelName}DataProvider.php
- {modelName}DataHandler.php

In config/service

- {modelName}Form.yml

artisan:module:admin:legacycontroller:create

This command is used to create a legacy back-office controller.

Syntax

```
php bin/console artisan:module:admin:legacycontroller:create {moduleName} {modelName}
```

- The module must exist.
- The modele must exist. (The model class must be accessible).

How it works

The artisan analyzes the model specify in parameters to automatically create the CRUD legacy controller.

Parameters are saved in the file `legacy_(ModelName)` and in the section `[controller]`.

Generated files

In controllers/admin:

- `Adminlegacy{modelName}Controller.php`

In src/Manager :

- `TabManager.php`

artisan:module:admin:crudcontroller:create

This command is used to create a **Modern CRUD back-office controller**.

Syntax

```
php bin/console artisan:module:admin:crudcontroller:create {moduleName} {modelName}
```

- The module must exist.
- The modele must exist. (The model class must be accessible).

How it works

The artisan analyzes the model specify in parameters to automatically create a grid and the form to list, create, modify and delete a record.

Parameters are saved in the file `ctrl_(ModelName)` and in the section `[controller]`.

Generated files

In src/Form:

- Admin{modelName}FormType.php
- Admin{modelName}DataProvider.php
- Admin{modelName}DataHandler.php

In src/Filter:

- Admin{modelName}Filter.php

In src/Grid:

- Admin{modelName}GridDefinitionFactory.php
- Admin{modelName}GridQueryBuilder.php

In src/Manager :

- TabManager.php

In src/Controlle :

- Admin{modelName}Controller.php

In views/templates/admin :

- Admin{modelName}Controller.index.html.twig
- Admin{modelName}Controller.form.html.twig

In config/service

- admin{modelName}.yaml

In config/routes

- admin{modelName}.yaml

In views/_dev

- package.json
- webpack.config.js
- postcss.config.js
- js/admin{modelName}controller.js
- scss/admin{modelName}controller.scss
- webpack/css.config.js
- webpack/js.config.js

In views/assets

- css/admin{modelName}controller.css
- js/admin{modelName}controller.js

artisan:module:model:create

This command is used to create a Model (ObjectModel).

Syntax

```
php bin/console artisan:module:model:create {moduleName} {modelName}
```

- The module must exist.
- A table with the same name as the model must exist and be accessible.

How it works

The artisan searches for a table in the database with the same name as the model and automatically generates the corresponding class model (ObjectModel).

Parameters are saved in the file model_(modelName) and in the section[model].

Generated files

In src/Model:

- {modelName}.php

artisan:module:hook:list

This command lists the hooks used by the module

Syntax

```
php bin/console artisan:module:hook:list {moduleName}
```

How it works

The artisan searches on which hooks the module is registered.

No parameters are saved.

Generated files

none

artisan:module:hook:register

This command register a module onto one or more hooks.

Syntax

```
php bin/console artisan:module:hook:register {moduleName} {hooks}
```

- *{moduleName}* is the name of the module.
- *{hooks}* is a name or a list of hook names separated by a space.

How it works

The artisan connect the module to hooks.

No parameters are saved.

Generated files

none

artisan:module:hook:unregister

This command **unregister** a module from one or more hooks.

Syntax

```
php bin/console artisan:module:hook:unregister {moduleName} {hooks}
```

- *{moduleName}* is the name of the module.
- *{hooks}* is a name or a list of hook names separated by a space.

How it works

The artisan disconnect the module from hooks.

No parameters are saved.

Generated files

none

artisan:override:disable

This command is used to **disable override**.

Syntax

```
php bin/console artisan:override:disable
```

How it works

The artisan will set the **PS_DISABLE_OVERRIDES** configuration value to **TRUE**.

No parameters are saved.

Generated files

none

artisan:override:enable

This command is used to **enable override**.

Syntax

```
php bin/console artisan:override:enable
```

How it works

The artisan will set the **PS_DISABLE_OVERRIDES** configuration value to **FALSE**.

No parameters are saved.

Generated files

none

artisan:override:class

This command is used to **override a function of the prestashop framework**.

Syntax

```
php bin/console artisan:override:class {className} {functionName}
```

- *{className}* is the PrestaShop framework class name (without Core).
- *{functionName}* is the name of the function to override.

How it works

The artisan will create the new class and function in the override directory.

No parameters are saved.

Generated files

A new file can be created in the override directory. The name and directory of the generated file are calculated according to the file of the original class.

artisan:sql:load {filePathAndName}

This command load and execute sql from file.

Syntax

```
php bin/console artisan:sql:load {filePathAndName}
```

- *{filePathAndName}* is the file path of the sql file to load and execute.

How it works

The artisan loads the file and executes it.

No parameters are saved.

Generated files

none

artisan:sql:snapshot {snapshotName}

This command **take a snapshot of the database.**

Syntax

```
php bin/console artisan:sql:snapshot {snapshotName}
```

- *{snapshotName}* is the name of the snapshot.

How it works

The artisan creates a copy of all tables in the database. The copy have a name prefixed with the snapshot name.

No parameters are saved.

Generated files

none

artisan:sql:restore {snapshotName}

This command restore the database from a snapshot.

Syntax

```
php bin/console artisan:sql:restore {snapshotName}
```

- *{snapshotName}* is the name of the snapshot.

How it works

The artisan restores the database tables from a snapshot.

No parameters are saved.

Generated files

none

artisan:shop:disable

This command is used to set the shop in maintenance mode.

Syntax

```
php bin/console artisan:shop:disable
```

How it works

The artisan will set the **PS_SHOP_ENABLE** configuration value to **FALSE**.

No parameters are saved.

Generated files

none

artisan:shop:enable

This command is used to **exit the maintenance mode and enable the shop**.

Syntax

```
php bin/console artisan:shop:enable
```

How it works

The artisan will set the **PS_SHOP_ENABLE** configuration value to **TRUE**.

No parameters are saved.

Generated files

none

artisan:theme:create:child

This command is used to create a PrestaShop child theme.

Syntax

```
php bin/console artisan:theme:create:child {themeName}
```

- If the theme already exists, some files can be updated.

How it works

The artisan asks some questions :

- Parent theme? Answer with the name of the parent theme

Generated files

In the theme directory:

- index.php
- Readme.md

In /config

- index.php
- theme.yml

Builder

Builders are the classes used to generate files. A builder uses a twig template to generate code. Templates are stored in the `src/Resources/templates/builder` directory. All builders inherit `ArtisanBuilder`. A builder can authorize or not the modification of an existing file (With the parameter `mustOverwriteIfExists`). If editing an existing file is allowed, the existing file is renamed (the file is saved) before generating the new one.

Builders are grouped by categories in different directories.

- **Assets** - Builders for resource generation (Js, css, etc.)
- **File** - Builders for generating utility files (compose.json, package.json, index.php, etc.)
- **Module** - Builders for generating module files (php code, config file, etc.)
- **Template** - Builders for template generation (Template Smarty)

Assets

Builders in the `Asset` directory are used to generate resource files (JS File, CSS, etc.).

- `CssBuilder` : to generate CSS files
- `JsBuilder` : to generate JS (ES5) files
- `JsES6Builder` : to generate JS (ES6) files
- `JsLibBuilder` : to add a Javascript library in the project
- `SassBuilder` : to generate Sass files

File

Builders in the `File` directory are used to generate utility files (composer.json, package.json, Readme.md, etc.)

- `ComposerBuilder` : to generate composer.json
- `GitIgnoreBuilder` : to generate .gitignore
- `IndexBuilder` : to generate index.php in module directories.
- `ModuleReadmeBuilder` : to generate Readme.md of the module
- `PackageBuilder` : to generate `/views/_dev/package.json`
- `PostCSSBuilder` : to generate `/views/_dev/postcss.config.js`
- `ThemeConfigBuilder` : to generate `/config/theme.yml` du theme
- `ThemeReadmeBuilder` : to generate Readme.md of the theme
- `WebpackBuilder` : to generate `/views/_dev/webpack.config.js`
- `WebpackCSSBuilder` : to generate `/views/_dev/webpack/css.config.js`
- `WebpackJSBuilder` : to generate `/views/_dev/webpack/js.config.js`

Module

Builders in the `Module` directory are used to generate module files (Controller, Grid, Model, Views, etc). Builders are stored in subdirectory.

- **Config** : builders for the configuration files (services.yml, routes.yml, etc.).
- **Controller** : builders for controllers.
- **Filter** : builders for filters.
- **Form** : builders for forms.
- **Grid** builders for grids.
- **Manager** : builders for manager.
- **Model** : builders for model.
- **Views** : builders for views.

Build Parameter

The code generator uses variables and twig templates. The available variables vary according to the type of code generated.

- ControllerBuildParameter is used for Controller
- FilterBuildParameter is used for Filter
- FormBuilderParameter is used for Form
- GridBuildParameter is used for Grid
- ModelBuildParameter is used for Model

Parameter for Controller generation

Section [controller]

- controllerBaseName : The controller base name (ex: AdminData)
- controllerName : {controllerBaseName}Controller
- controllerLegacyName : {controllerBaseName}ControllerLegacyClass
- controllerRoutePrefix : {moduleName}_{controllerBaseName}
- controllerServicePrefix: {moduleName}_{controllerBaseName}
- controllerUrl : /{moduleName}/{controllerBaseName}
- controllerNamespace : {moduleNameNamespace}\Controller

Parameter for Filter generation

Section [filter]

- filterName : {baseControllerName}Filter
- filterLimit : 10
- filterOffset : 0
- filterOrderBy : {primaryKey}
- filterSortOrder : asc
- filterNamespace : {moduleNameNamespace}\Filter

Parameter for Form generation

Section [form]

- formName : {baseControllerName}Form
- formTypeClass : {baseControllerName}FormType
- formDataHandlerClass : {baseControllerName}DataHandler
- formDataProviderClass: {baseControllerName}DataProvider
- formKey : {primaryKey}
- formNamespace : {moduleNameNamespace}\Form

Parameter for Grid generation

Section [grid]

- gridName : {baseControllerName}Grid
- gridNamespace : {moduleNameNamespace}\Grid

Parameter for Model generation

Section [model]

- modelName : {modelName}
- modelDisplayName : {modelName}
- modelNamespace : empty or read from class {modelName}
- tableName : {modelName}
- multilang : true|false
- multilang_shop: true|false
- primary : id_{tableName}
- fieldDefinition : read from class or generate from db
- fields_list : generate from fieldDefinition
- fields_form : generate from fieldDefinition

Extra parameter when creating model from Database only

- createTable : empty or read from db
- createTableLang : empty or read from db

The parameters are extracted from the class model or table structure in the database.

