The symfony Reference Book

symfony 1.2

This PDF is brought to you by

Version: reference-1.2-en-2009-10-11

Table of Contents

About the Author

Fabien Potencier is a serial entrepreneur. In 1998, right after graduation, Fabien founded his very first company with a fellow student. The company was a web agency focused on simplicity and Open-Source technologies, and was called Sensio. His acute technical knowledge and his endless curiosity won him the confidence of many French big corporate companies.

Fabien is also the creator and the lead developer of the symfony framework.

Today, Fabien spends most of his time as Sensio's CEO and as the symfony project leader.

About Sensio Labs

Sensio Labs is a French web agency well known for its innovative ideas on web development. Founded in 1998 by Fabien Potencier, Gregory Pascal, and Samuel Potencier, Sensio benefited from the Internet growth of the late 1990s and situated itself as a major player for building complex web applications. It survived the Internet bubble burst by applying professional and industrial methods to a business where most players seemed to reinvent the wheel for each project. Most of Sensio's clients are large French corporations, who hire its teams to deal with small- to middle-scale projects with strong time-to-market and innovation constraints.

Sensio Labs develops interactive web applications, both for dot-com and traditional companies. Sensio Labs also provides auditing, consulting, and training on Internet technologies and complex application deployment. It helps define the global Internet strategy of large-scale industrial players. Sensio Labs has projects in France and abroad.

For its own needs, Sensio Labs develops the symfony framework and sponsors its deployment as an Open-Source project. This means that symfony is built from experience and is employed in many web applications, including those of large corporations.

Since its beginnings ten years ago, Sensio has always based its strategy on strong technical expertise. The company focuses on Open-Source technologies, and as for dynamic scripting languages, Sensio offers developments in all LAMP platforms. Sensio acquired strong experience on the best frameworks using these languages, and often develops web applications in Django, Rails, and, of course, symfony.

Sensio Labs is always open to new business opportunities, so if you ever need help developing a web application, learning symfony, or evaluating a symfony development, feel free to contact us at fabien.potencier@sensio.com. The consultants, project managers, web designers, and developers of Sensio can handle projects from A to Z.

Introduction

Using a full-stack framework like symfony is one of the easiest ways to increase your speed and efficiency as a web developer. The framework comes bundled with many useful features that help you concentrate on your application's business logic rather than on the implementation on yet another object pager or yet another database abstraction layer. However, this also comes at a cost; learning all the available features and all the built-in configuration possibilities does not happen overnight.

The *Practical Symfony*[1] book, released at the end of 2008, is a great way for a beginner to learn symfony, understand how it works, and also see best web development practices in action.

When you begin working on your own projects, you need a reference guide. A book where you can easily find answers to your questions at your fingertips. The *Symfony Reference Guide* book aims to provide such a guide. It acts as a complementary book to *Practical symfony*. This is a book you will keep with you whenever you develop with symfony. This book is the fastest way to find every available configuration thanks to a very detailed table of contents, an index of terms, cross-references inside the chapters, tables, and much more.

Despite being the lead developer of symfony, I still use this book from time to time to look for a particular configuration setting, or just browse the book to re-discover some great tips. I hope you will enjoy using it on a day to day basis as much as I do.

1. `http://www.symfony-project.org/jobeet/`

The YAML Format

Most configuration files in symfony are in the YAML format. According to the official YAML[2] website, YAML is “a human friendly data serialization standard for all programming languages”.

YAML is a simple language that describes data. Like PHP, it has a syntax for simple types like strings, booleans, floats, or integers. But unlike PHP, it makes a difference between arrays (sequences) and hashes (mappings).

This section describes the minimum set of features you will need to use YAML as a configuration file format in symfony, although the YAML format is capable of describing much more complex nested data structures.

Scalars

The syntax for scalars is similar to the PHP syntax.

Strings

Listing 2-1
```
A string in YAML
```

Listing 2-2
```
'A singled-quoted string in YAML'
```

In a single quoted string, a single quote `'` must be doubled:

Listing 2-3
```
'A single quote '' in a single-quoted string'
```

Listing 2-4
```
"A double-quoted string in YAML\n"
```

Quoted styles are useful when a string starts or ends with one or more relevant spaces.

The double-quoted style provides a way to express arbitrary strings, by using `\` escape sequences. It is very useful when you need to embed a `\n` or a unicode character in a string.

When a string contains line breaks, you can use the literal style, indicated by the pipe (`|`), to indicate that the string will span several lines. In literals, newlines are preserved:

Listing 2-5

2. `http://yaml.org/`

```
|
  \/ /| |\/| |
  / / | |  | |__
```

Alternatively, strings can be written with the folded style, denoted by `>`, where each line break is replaced by a space:

```
>
  This is a very long sentence
  that spans several lines in the YAML
  but which will be rendered as a string
  without carriage returns.
```

Listing 2-6

Notice the two spaces before each line in the previous examples. They won't appear in the resulting PHP strings.

Numbers

```
# an integer
12
```

Listing 2-7

```
# an octal
014
```

Listing 2-8

```
# a hexadecimal
0xC
```

Listing 2-9

```
# a float
13.4
```

Listing 2-10

```
# an exponent
1.2e+34
```

Listing 2-11

```
# infinity
.inf
```

Listing 2-12

Nulls

Nulls in YAML can be expressed with `null` or `~`.

Booleans

Booleans in YAML are expressed with `true` and `false`.

The symfony YAML parser also recognize `on`, `off`, `yes`, and `no` but it is strongly discouraged to use them as it has been removed from the YAML specifications as of version 1.2.

Dates

YAML uses the ISO-8601 standard to express dates:

```
2001-12-14t21:59:43.10-05:00
```

Listing 2-13

Listing 2-14
```
# simple date
2002-12-14
```

Collections

A YAML file is rarely used to describe a simple scalar. Most of the time, it describes a collection. A collection can be either a sequence or mapping of elements. Sequences and mappings are both converted to PHP arrays.

Sequences use a dash followed by a space (`-`):

Listing 2-15
```
- PHP
- Perl
- Python
```

This is equivalent to the following PHP code:

Listing 2-16
```
array('PHP', 'Perl', 'Python');
```

Mappings use a colon followed by a space (`:`) to mark each key/value pair:

Listing 2-17
```
PHP: 5.2
MySQL: 5.1
Apache: 2.2.20
```

which is equivalent to the following PHP code:

Listing 2-18
```
array('PHP' => 5.2, 'MySQL' => 5.1, 'Apache' => '2.2.20');
```

In a mapping, a key can be any valid YAML scalar.

The number of spaces between the colon and the value does not matter, as long as there is at least one:

Listing 2-19
```
PHP:    5.2
MySQL:  5.1
Apache: 2.2.20
```

YAML uses indentation with one or more spaces to describe nested collections:

Listing 2-20
```
"symfony 1.0":
  PHP:    5.0
  Propel: 1.2
"symfony 1.2":
  PHP:    5.2
  Propel: 1.3
```

This YAML is equivalent to the following PHP code:

Listing 2-21
```
array(
  'symfony 1.0' => array(
    'PHP'    => 5.0,
    'Propel' => 1.2,
  ),
```

```
  'symfony 1.2' => array(
    'PHP'    => 5.2,
    'Propel' => 1.3,
  ),
);
```

There is one important thing you need to remember when using indentation in a YAML file: *Indentation must be done with one or more spaces, but never with tabulations*.

you can nest sequences and mappings as you like or you can nest sequences and mappings like so:

```
'Chapter 1':
  - Introduction
  - Event Types
'Chapter 2':
  - Introduction
  - Helpers
```

Listing 2-22

YAML can also use flow styles for collections, using explicit indicators rather than indentation to denote scope.

A sequence can be written as a comma separated list within square brackets (`[]`):

```
[PHP, Perl, Python]
```

Listing 2-23

A mapping can be written as a comma separated list of key/values within curly braces (`{}`):

```
{ PHP: 5.2, MySQL: 5.1, Apache: 2.2.20 }
```

Listing 2-24

You can also mix and match styles to achieve better readability:

```
'Chapter 1': [Introduction, Event Types]
'Chapter 2': [Introduction, Helpers]
```

Listing 2-25

```
"symfony 1.0": { PHP: 5.0, Propel: 1.2 }
"symfony 1.2": { PHP: 5.2, Propel: 1.3 }
```

Listing 2-26

Comments

Comments can be added in YAML by prefixing them with a hash mark (`#`):

```
# Comment on a line
"symfony 1.0": { PHP: 5.0, Propel: 1.2 } # Comment at the end of a line
"symfony 1.2": { PHP: 5.2, Propel: 1.3 }
```

Listing 2-27

Comments are simply ignored by the YAML parser and do not need to be indented according to the current level of nesting in a collection.

Dynamic YAML files

In symfony, a YAML file can contain PHP code that is evaluated just before the parsing occurs:

Listing 2-28
```
1.0:
  version: <?php echo file_get_contents('1.0/VERSION')."\n" ?>
1.1:
  version: "<?php echo file_get_contents('1.1/VERSION') ?>"
```

Be careful to not mess up with the indentation. Keep in mind the following simple tips when adding PHP code to a YAML file:

- The `<?php ?>` statements must always start the line or be embedded in a value.
- If a `<?php ?>` statement ends a line, you need to explicitly output a new line ("\n").

A Full Length Example

The following example illustrates the YAML syntax explained in this section:

Listing 2-29

```
"symfony 1.0":
  end_of_maintainance: 2010-01-01
  is_stable:           true
  release_manager:     "Gregoire Hubert"
  description: >
    This stable version is the right choice for projects
    that need to be maintained for a long period of time.
  latest_beta:         ~
  latest_minor:        1.0.20
  supported_orms:      [Propel]
  archives:            { source: [zip, tgz], sandbox: [zip, tgz] }

"symfony 1.2":
  end_of_maintainance: 2008-11-01
  is_stable:           true
  release_manager:     'Fabian Lange'
  description: >
    This stable version is the right choice
    if you start a new project today.
  latest_beta:         null
  latest_minor:        1.2.5
  supported_orms:
    - Propel
    - Doctrine
  archives:
    source:
      - zip
      - tgz
    sandbox:
      - zip
      - tgz
```

Configuration File Principles

Symfony configuration files are based on a common set of principles and share some common properties. This section describes them in detail, and acts as a reference for other sections describing YAML configuration files.

Cache

All configuration files in symfony are cached to PHP files by configuration handler classes. When the `is_debug` setting is set to `false` (for instance for the `prod` environment), the YAML file is only accessed for the very first request; the PHP cache is used for subsequent requests. This means that the "heavy" work is done only once, when the YAML file is parsed and interpreted the first time.

In the `dev` environment, where `is_debug` is set to `true` by default, the compilation is done whenever the configuration file changes (symfony checks the file modification time).

The parsing and caching of each configuration file is done by specialized configuration handler classes, configured in `config_handler.yml` *(page 80)*.

In the following sections, when we talk about the "compilation", it means the first time when the YAML file is converted to a PHP file and stored in the cache.

To force the configuration cache to be reloaded, you can use the `cache:clear` task:

Listing 3-1

```
$ php symfony cache:clear --type=config
```

Constants

Configuration files: `core_compile.yml`, `factories.yml`, `generator.yml`, `databases.yml`, `filters.yml`, `view.yml`, `autoload.yml`

Some configuration files allow the usage of pre-defined constants. Constants are declared with placeholders using the %XXX% notation (where XXX is an uppercase key) and are replaced by their actual value at "compilation" time.

Configuration Settings

A constant can be any setting defined in the `settings.yml` configuration file. The placeholder key is then an upper-case setting key name prefixed with `SF_`:

Listing 3-2

```
logging: %SF_LOGGING_ENABLED%
```

When symfony compiles the configuration file, it replaces all occurrences of the %SF_XXX% placeholders by their value from settings.yml. In the above example, it will replace the SF_LOGGING_ENABLED placeholder with the value of the logging_enabled setting defined in settings.yml.

Application Settings

You can also use settings defined in the app.yml configuration file by prefixing the key name with APP_.

Special Constants

By default, symfony defines four constants according to the current front controller:

Constant	Description	Configuration method
SF_APP	The current application name	getApplication()
SF_ENVIRONMENT	The current environment name	getEnvironment()
SF_DEBUG	Whether debug is enabled or not	isDebug()
SF_SYMFONY_LIB_DIR	The symfony libraries directory	getSymfonyLibDir()

Directories

Constants are also very useful when you need to reference a directory or a file path without hardcoding it. Symfony defines a number of constants for common project and application directories.

At the root of the hierarchy is the project root directory, SF_ROOT_DIR. All other constants are derived from this root directory.

The project directory structure is defined as follows:

Constants	Default Value
SF_APPS_DIR	SF_ROOT_DIR/apps
SF_CONFIG_DIR	SF_ROOT_DIR/config
SF_CACHE_DIR	SF_ROOT_DIR/cache
SF_DATA_DIR	SF_ROOT_DIR/data
SF_DOC_DIR	SF_ROOT_DIR/doc
SF_LIB_DIR	SF_ROOT_DIR/lib
SF_LOG_DIR	SF_ROOT_DIR/log
SF_PLUGINS_DIR	SF_ROOT_DIR/plugins
SF_TEST_DIR	SF_ROOT_DIR/test
SF_WEB_DIR	SF_ROOT_DIR/web
SF_UPLOAD_DIR	SF_WEB_DIR/uploads

The application directory structure is defined under the SF_APPS_DIR/APP_NAME directory:

Constants	Default Value
SF_APP_CONFIG_DIR	SF_APP_DIR/config
SF_APP_LIB_DIR	SF_APP_DIR/lib

Constants	Default Value
SF_APP_MODULE_DIR	SF_APP_DIR/modules
SF_APP_TEMPLATE_DIR	SF_APP_DIR/templates
SF_APP_I18N_DIR	SF_APP_DIR/i18n

Eventually, the application cache directory structure is defined as follows:

Constants	Default Value
SF_APP_BASE_CACHE_DIR	SF_CACHE_DIR/APP_NAME
SF_APP_CACHE_DIR	SF_CACHE_DIR/APP_NAME/ENV_NAME
SF_TEMPLATE_CACHE_DIR	SF_APP_CACHE_DIR/template
SF_I18N_CACHE_DIR	SF_APP_CACHE_DIR/i18n
SF_CONFIG_CACHE_DIR	SF_APP_CACHE_DIR/config
SF_TEST_CACHE_DIR	SF_APP_CACHE_DIR/test
SF_MODULE_CACHE_DIR	SF_APP_CACHE_DIR/modules

environment-awareness

Configuration files: `settings.yml`, `factories.yml`, `databases.yml`, `app.yml`

Some symfony configuration files are environment-aware — their interpretation depends on the current symfony environment. These files have different sections that define the configuration should vary for each environment. When creating a new application, symfony creates sensible configuration for the three default symfony environments: `prod`, `test`, and `dev`:

Listing 3-3

```
prod:
  # Configuration for the `prod` environment

test:
  # Configuration for the `test` environment

dev:
  # Configuration for the `dev` environment

all:
  # Default configuration for all environments
```

When symfony needs a value from a configuration file, it merges the configuration found in the current environment section with the `all` configuration. The special `all` section describes the default configuration for all environments. If the environment section is not defined, symfony falls back to the `all` configuration.

Configuration Cascade

Configuration files: `core_compile.yml`, `autoload.yml`, `settings.yml`, `factories.yml`, `databases.yml`, `security.yml`, `cache.yml`, `app.yml`, `filters.yml`, `view.yml`

Some configuration files can be defined in several `config/` sub-directories contained in the project directory structure.

When the configuration is compiled, the values from all the different files are merged according to a precedence order:

- The module configuration (`PROJECT_ROOT_DIR/apps/APP_NAME/modules/MODULE_NAME/config/XXX.yml`)
- The application configuration (`PROJECT_ROOT_DIR/apps/APP_NAME/config/XXX.yml`)
- The project configuration (`PROJECT_ROOT_DIR/config/XXX.yml`)
- The configuration defined in the plugins (`PROJECT_ROOT_DIR/plugins/*/config/XXX.yml`)
- The default configuration defined in the symfony libraries (`SF_LIB_DIR/config/XXX.yml`)

For instance, the `settings.yml` defined in an application directory inherits from the configuration set in the main `config/` directory of the project, and eventually from the default configuration contained in the framework itself (`lib/config/config/settings.yml`).

When a configuration file is environment-aware and can be defined in several directories, the following priority list applies:

1. Module
2. Application
3. Project
4. Specific environment
5. All environments
6. Default

The settings.yml Configuration File

Most aspects of symfony can be configured either via a configuration file written in YAML, or with plain PHP. In this section, the main configuration file for an application, `settings.yml`, will be described.

The main `settings.yml` configuration file for an application can be found in the `apps/APP_NAME/config/` directory.

As discussed in the introduction, the `settings.yml` file is **environment-aware** *(page 20)*, and benefits from the **configuration cascade mechanism** *(page 20)*.

Each environment section has two sub-sections: `.actions` and `.settings`. All configuration directives go under the `.settings` sub-section, except for the default actions to be rendered for some common pages.

The `settings.yml` configuration file is cached as a PHP file; the process is automatically managed by the `sfDefineEnvironmentConfigHandler` class *(page 80)*.

Settings

- `.actions`
  - `error_404` *(page 24)*
  - `login` *(page 24)*
  - `secure` *(page 24)*
  - `module_disabled` *(page 24)*
- `.settings`
  - `cache` *(page 26)*
  - `charset` *(page 25)*
  - `check_lock` *(page 28)*
  - `check_symfony_version` *(page 28)*
  - `compressed` *(page 28)*
  - `csrf_secret` *(page 25)*
  - `default_culture` *(page 27)*
  - `default_timezone` *(page 26)*
  - `enabled_modules` *(page 25)*
  - `error_reporting` *(page 28)*
  - `escaping_strategy` *(page 24)*
  - `escaping_method` *(page 25)*
  - `etag` *(page 26)*
  - `i18n` *(page 26)*
  - `lazy_cache_key` *(page 27)*
  - `logging_enabled` *(page 27)*
  - `no_script_name` *(page 27)*
  - `max_forwards` *(page 29)*
  - `standard_helpers` *(page 27)*
  - `strip_comments` *(page 29)*
  - `use_database` *(page 28)*
  - `web_debug` *(page 27)*
  - `web_debug_web_dir` *(page 29)*

The `.actions` Sub-Section

Default configuration:

Listing 4-1
```
default:
  .actions:
    error_404_module:       default
    error_404_action:       error404

    login_module:           default
    login_action:           login

    secure_module:          default
    secure_action:          secure

    module_disabled_module: default
    module_disabled_action: disabled
```

The `.actions` sub-section defines the action to execute when common pages must be rendered. Each definition has two components: one for the module (suffixed by `_module`), and one for the action (suffixed by `_action`).

`error_404`

The `error_404` action is executed when a 404 page must be rendered.

`login`

The `login` action is executed when a non-authenticated user tries to access a secure page.

`secure`

The `secure` action is executed when a user doesn't have the required credentials.

`module_disabled`

The `module_disabled` action is executed when a user requests a disabled module.

The `.settings` Sub-Section

The `.settings` sub-section is where the framework configuration occurs. The paragraphs below describe all possible settings and are roughly ordered by importance.

All settings defined in the `.settings` section are available anywhere in the code by using the `sfConfig` object and prefixing the setting with `sf_`. For instance, to get the value of the `charset` setting, use:

Listing 4-2
```
sfConfig::get('sf_charset');
```

`escaping_strategy`

Default: `off`

The `escaping_strategy` setting is a Boolean setting that determines if the output escaper sub-framework is enabled. When enabled, all variables made available in the templates are automatically escaped by calling the helper function defined by the `escaping_method` setting (see below).

Be careful that the `escaping_method` is the default helper used by symfony, but this can be overridden on a case by case basis, when outputting a variable in a JavaScript script tag for example.

The output escaper sub-framework uses the `charset` setting for the escaping.

It is highly recommended to change the default value to `on`.

This settings can be set when you create an application with the `generate:app` task by using the `--escaping-strategy` option.

`escaping_method`

Default: `ESC_SPECIALCHARS`

The `escaping_method` defines the default function to use for escaping variables in templates (see the `escaping_strategy` setting above).

You can choose one of the built-in values: `ESC_SPECIALCHARS`, `ESC_RAW`, `ESC_ENTITIES`, `ESC_JS`, `ESC_JS_NO_ENTITIES`, and `ESC_SPECIALCHARS`, or create your own function.

Most of the time, the default value is fine. The `ESC_ENTITIES` helper can also be used, especially if you are only working with English or European languages.

`csrf_secret`

Default: `false`

The `csrf_secret` is a unique secret for your application. If not set to `false`, it enables CSRF protection for all forms defined with the form framework. This settings is also used by the `link_to()` helper when it needs to convert a link to a form (to simulate a `DELETE` HTTP method for example).

It is highly recommended to change the default value to a unique secret.

This settings can be set when you create an application with the `generate:app` task by using the `--csrf-secret` option.

`charset`

Default: `utf-8`

The `charset` setting is the charset that will be used everywhere in the framework: from the response `Content-Type` header, to the output escaping feature.

Most of the time, the default is fine.

> *This setting is used in many different places in the framework, and so its value is cached in several places. After changing it, the configuration cache must be cleared, even in the development environment.*

`enabled_modules`

Default: `[default]`

The `enabled_modules` is an array of module names to enable for this application. Modules defined in plugins or in the symfony core are not enabled by default, and must be listed in this setting to be accessible.

Adding a module is as simple as appending it to the list (the order of the modules do not matter):

Listing 4-3
```
enabled_modules: [default, sfGuardAuth]
```

The `default` module defined in the framework contains all the default actions set in the `.actions` sub-section of `settings.yml`. It is recommended that you customize all of them, and then remove the `default` module from this setting.

`default_timezone`

Default: none

The `default_timezone` setting defines the default timezone used by PHP. It can be any timezone[3] recognized by PHP.

If you don't define a timezone, you are advised to define one in the `php.ini` file. If not, symfony will try to guess the best timezone by calling the `date_default_timezone_get()`[4] PHP function.

`cache`

Default: `off`

The `cache` setting enables or disables template caching.

The general configuration of the cache system is done in the `view_cache_manager` *(page 37)* and `view_cache` *(page 38)* sections of the `factories.yml` configuration file. The fined-grained configuration is done in the `cache.yml` *(page 61)* configuration file.

`etag`

Default: `on` by default except for the `dev` and `test` environments

The `etag` setting enables or disables the automatic generation of `ETag` HTTP headers. The ETag generated by symfony is a simple md5 of the response content.

`i18n`

Default: `off`

The `i18n` setting is a Boolean that enables or disables the i18n sub-framework. If your application is internationalized, set it to `on`.

The general configuration of the i18n system is to be done in the `i18n` *(page 38)* section of the `factories.yml` configuration file.

3. `http://www.php.net/manual/en/class.datetimezone.php`
4. `http://www.php.net/date_default_timezone_get`

default_culture

Default: en

The default_culture setting defines the default culture used by the i18n sub-framework. It can be any valid culture.

standard_helpers

Default: [Partial, Cache, Form]

The standard_helpers setting is an array of helper groups to load for all templates (name of the group helper without the Helper suffix).

no_script_name

Default: on for the prod environment of the first application created, off for all others

The no_script_name setting determines whether the front controller script name is prepended to generated URLs or not. By default, it is set to on by the generate:app task for the prod environment of the first application created.

Obviously, only one application and environment can have this setting set to on if all front controllers are in the same directory (web/). If you want more than one application with no_script_name set to on, move the corresponding front controller(s) under a sub-directory of the web root directory.

lazy_cache_key

Default: on for new projects, off for upgraded projects

When enabled, the lazy_cache_key setting delays the creation of a cache key until after checking whether an action or partial is cacheable. This can result in a big performance improvement, depending on your usage of template partials.

This setting was introduced in symfony 1.2.7 to improve performance without breaking backward compatibility with previous 1.2 releases. It will be removed in symfony 1.3 as the optimization will always be enabled.

This setting is only available for symfony 1.2.7 and up.

---

logging_enabled

Default: on for all environments except prod

The logging_enabled setting enables the logging sub-framework. Setting it to false bypasses the logging mechanism completely and provides a small performance gain.

The fined-grained configuration of the logging is to be done in the factories.yml configuration file.

---

web_debug

Default: off for all environments except dev

The web_debug setting enables the web debug toolbar. The web debug toolbar is injected into a page when the response content type is HTML.

error_reporting

Default:

- `prod`: E_PARSE | E_COMPILE_ERROR | E_ERROR | E_CORE_ERROR | E_USER_ERROR
- `dev`: E_ALL | E_STRICT
- `test`: (E_ALL | E_STRICT) ^ E_NOTICE
- default: E_PARSE | E_COMPILE_ERROR | E_ERROR | E_CORE_ERROR | E_USER_ERROR

The `error_reporting` setting controls the level of PHP error reporting (to be displayed in the browser and written to the logs).

The PHP website has some information about how to use bitwise operators[5].

The default configuration is the most sensible one, and should not be altered.

The display of errors in the browser is automatically disabled for front controllers that have `debug` disabled, which is the case by default for the `prod` environment.

compressed

Default: `off`

The `compressed` setting enables native PHP response compression. If set to `on`, symfony will use `ob_gzhandler`[6] as a callback function for `ob_start()`.

It is recommended to keep it to `off`, and use the native compression mechanism of your web server instead.

use_database

Default: `on`

The `use_database` determines if the application uses a database or not.

check_lock

Default: `off`

The `check_lock` setting enables or disables the application lock system triggered by some tasks like `cache:clear` and `project:disable`.

If set to `on`, all requests to disabled applications are automatically redirected to the symfony core `lib/exception/data/unavailable.php` page.

You can override the default unavailable template by adding a `config/unavailable.php` file to your project or application.

check_symfony_version

Default: `off`

5. `http://www.php.net/language.operators.bitwise`
6. `http://www.php.net/ob_gzhandler`

The `check_symfony_version` enables or disables checking of the current symfony version upon every request. If enabled, symfony will clear the cache automatically when the framework is upgraded.

It is highly recommended to not set this to `on` as it adds a small overhead, and because it is simple to just clear the cache when you deploy a new version of your project. This setting is only useful if several projects share the same symfony code, which is not recommended.

`web_debug_web_dir`

Default: `/sf/sf_web_debug`

The `web_debug_web_dir` sets the web path to the web debug toolbar assets (images, stylesheets, and JavaScript files).

`strip_comments`

Default: `on`

The `strip_comments` determines if symfony should strip the comments when compiling core classes. This setting is only used if the application `debug` setting is set to `off`.

If you have blank pages in the production environment only, try to set this setting to `off`.

`max_forwards`

Default: `5`

The `max_forwards` setting sets the maximum number of internal forwards allowed before symfony throws an exception. This is to avoid infinite loops.

The factories.yml Configuration File

Factories are core objects needed by the framework during the life of any request. They are configured in the `factories.yml` configuration file and always accessible via the `sfContext` object:

Listing 5-1
```
// get the user factory
sfContext::getInstance()->getUser();
```

The main `factories.yml` configuration file for an application can be found in the `apps/APP_NAME/config/` directory.

As discussed in the introduction, the `factories.yml` file is **environment-aware** *(page 20)*, benefits from the **configuration cascade mechanism** *(page 20)*, and can include **constants** *(page 18)*.

The `factories.yml` configuration file contains a list of named factories:

Listing 5-2
```
FACTORY_1:
  # definition of factory 1

FACTORY_2:
  # definition of factory 2

# ...
```

The supported factory names are: `controller`, `logger`, `i18n`, `request`, `response`, `routing`, `storage`, `user`, `view_cache`, and `view_cache_manager`.

When the `sfContext` initializes the factories, it reads the `factories.yml` file for the class name of the factory (`class`) and the parameters (`param`) used to configure the factory object:

Listing 5-3
```
FACTORY_NAME:
  class: CLASS_NAME
  param: { ARRAY OF PARAMETERS }
```

Being able to customize the factories means that you can use a custom class for symfony core objects instead of the default one. You can also change the default behavior of these classes by customizing the parameters sent to them.

If the factory class cannot be autoloaded, a `file` path can be defined and will be automatically included before the factory is created:

Listing 5-4
```
FACTORY_NAME:
  class: CLASS_NAME
  file:  ABSOLUTE_PATH_TO_FILE
```

The `factories.yml` configuration file is cached as a PHP file; the process is automatically managed by the `sfFactoryConfigHandler` class *(page 80)*.

Factories

- `request` *(page 34)*
  - `formats` *(page 34)*
  - `path_info_array` *(page 34)*
  - `path_info_key` *(page 34)*
  - `relative_url_root` *(page 34)*
- `response` *(page 34)*
  - `charset` *(page 35)*
  - `http_protocol` *(page 35)*
  - `send_http_headers` *(page 35)*
- `user` *(page 35)*
  - `default_culture` *(page 36)*
  - `timeout` *(page 35)*
  - `use_flash` *(page 36)*
- `storage` *(page 36)*
  - `auto_start` *(page 36)*
  - `database` *(page 37)*
  - `db_table` *(page 37)*
  - `db_id_col` *(page 37)*
  - `db_data_col` *(page 37)*
  - `db_time_col` *(page 37)*
  - `session_cache_limiter` *(page 37)*
  - `session_cookie_domain` *(page 37)*
  - `session_cookie_httponly` *(page 37)*
  - `session_cookie_lifetime` *(page 37)*
  - `session_cookie_path` *(page 37)*
  - `session_cookie_secure` *(page 37)*
  - `session_name` *(page 36)*
- `view_cache_manager` *(page 37)*
- `view_cache` *(page 38)*
- `i18n` *(page 38)*
  - `cache` *(page 39)*
  - `debug` *(page 38)*
  - `source` *(page 38)*
  - `untranslated_prefix` *(page 39)*
  - `untranslated_suffix` *(page 39)*
- `routing` *(page 39)*
  - `cache` *(page 39)*
  - `extra_parameters_as_query_string` *(page 40)*
  - `generate_shortest_url` *(page 39)*
  - `lazy_routes_deserialize` *(page 40)*
  - `lookup_cache_dedicated_keys` *(page 40)*
  - `load_configuration` *(page 40)*
  - `segment_separators` *(page 39)*

    - `suffix` *(page 40)*
    - `variable_prefixes` *(page 39)*
- `logger` *(page 41)*
    - `level` *(page 41)*
    - `loggers` *(page 41)*
- `controller` *(page 42)*

request

sfContext Accessor: `$context->getRequest()`

Default configuration:

Listing 5-5
```
request:
  class: sfWebRequest
  param:
    logging:           %SF_LOGGING_ENABLED%
    path_info_array:   SERVER
    path_info_key:     PATH_INFO
    relative_url_root: ~
    formats:
      txt:  text/plain
      js:   [application/javascript, application/x-javascript, text/
javascript]
      css:  text/css
      json: [application/json, application/x-json]
      xml:  [text/xml, application/xml, application/x-xml]
      rdf:  application/rdf+xml
      atom: application/atom+xml
```

path_info_array

The `path_info_array` option defines the global PHP array that will be used to retrieve information. On some configurations you may want to change the default `SERVER` value to `ENV`.

path_info_key

The `path_info_key` option defines the key under which the `PATH_INFO` information can be found.

If you use IIS with a rewriting module like `IIFR` or `ISAPI`, you may need to change this value to `HTTP_X_REWRITE_URL`.

formats

The `formats` option defines an array of file extensions and their corresponding `Content-Type`s. It is used by the framework to automatically manage the `Content-Type` of the response, based on the request URI extension.

relative_url_root

The `relative_url_root` option defines the part of the URL before the front controller. Most of the time, this is automatically detected by the framework and does not need to be changed.

response

sfContext Accessor: `$context->getResponse()`

Default configuration:

Listing 5-6
```
response:
  class: sfWebResponse
  param:
    logging:           %SF_LOGGING_ENABLED%
    charset:           %SF_CHARSET%
    send_http_headers: true
```

Default configuration for the `test` *environment*:

Listing 5-7
```
response:
  class: sfWebResponse
  param:
    send_http_headers: false
```

`send_http_headers`

The `send_http_headers` option specifies whether the response should send HTTP response headers along with the response content. This setting is mostly useful for testing, as headers are sent with the `header()` PHP function which sends warnings if you try to send headers after some output.

`charset`

The `charset` option defines the charset to use for the response. By default, it uses the `charset` setting from `settings.yml`, which is what you want most of the time.

`http_protocol`

The `http_protocol` option defines the HTTP protocol version to use for the response. By default, it checks the `$_SERVER['SERVER_PROTOCOL']` value if available or defaults to `HTTP/1.0`.

user

sfContext Accessor: `$context->getUser()`

Default configuration:

Listing 5-8
```
user:
  class: myUser
  param:
    timeout:         1800
    logging:         %SF_LOGGING_ENABLED%
    use_flash:       true
    default_culture: %SF_DEFAULT_CULTURE%
```

By default, the `myUser` class inherits from `sfBasicSecurityUser`, which can be configured in the `security.yml` *(page 59)* configuration file.

`timeout`

The `timeout` option defines the timeout for user authentication. It is not related to the session timeout. The default setting automatically unauthenticates a user after 30 minutes of inactivity.

This setting is only used by user classes that inherit from the `sfBasicSecurityUser` base class, which is the case of the generated `myUser` class.

To avoid unexpected behavior, the user class automatically forces the maximum lifetime for the session garbage collector (`session.gc_maxlifetime`) to be greater than the timeout.

`use_flash`

The `use_flash` option enables or disables the flash component.

`default_culture`

The `default_culture` option defines the default culture to use for a user who comes to the site for the first time. By default, it uses the `default_culture` setting from `settings.yml`, which is what you want most of the time.

If you change the `default_culture` setting in `factories.yml` or `settings.yml`, you need to clear your cookies in your browser to check the result.

`storage`

The storage factory is used by the user factory to persist user data between HTTP requests.

sfContext Accessor: `$context->getStorage()`

Default configuration:

Listing 5-9
```
storage:
  class: sfSessionStorage
  param:
    session_name: symfony
```

Default configuration for the `test` *environment*:

Listing 5-10
```
storage:
  class: sfSessionTestStorage
  param:
    session_path: %SF_TEST_CACHE_DIR%/sessions
```

`auto_start`

The `auto_start` option enables or disables the session auto-starting feature of PHP (via the `session_start()` function).

`session_name`

The `session_name` option defines the name of the cookie used by symfony to store the user session. By default, the name is `symfony`, which means that all your applications share the same cookie (and as such the corresponding authentication and authorizations).

`session_set_cookie_params()` parameters

The `storage` factory calls the `session_set_cookie_params()`[7] function with the value of the following options:

- `session_cookie_lifetime`: Lifetime of the session cookie, defined in seconds.
- `session_cookie_path`: Path on the domain where the cookie will work. Use a single slash (`/`) for all paths on the domain.
- `session_cookie_domain`: Cookie domain, for example `www.php.net`. To make cookies visible on all subdomains then the domain must be prefixed with a dot like `.php.net`.
- `session_cookie_secure`: If `true` cookie will only be sent over secure connections.
- `session_cookie_httponly`: If set to `true` then PHP will attempt to send the `httponly` flag when setting the session cookie.

The description of each option comes from the `session_set_cookie_params()` function description on the PHP website

`session_cache_limiter`

If the `session_cache_limiter` option is set, PHP's `session_cache_limiter()`[8] function is called and the option value is passed as an argument.

Database Storage-specific Options

When using a storage that inherits from the `sfDatabaseSessionStorage` class, several additional options are available:

- `database`: The database name (required)
- `db_table`: The table name (required)
- `db_id_col`: The primary key column name (`sess_id` by default)
- `db_data_col`: The data column name (`sess_data` by default)
- `db_time_col`: The time column name (`sess_time` by default)

`view_cache_manager`

sfContext Accessor: `$context->getViewCacheManager()`

Default configuration:

```
view_cache_manager:
  class: sfViewCacheManager
```

Listing 5-11

This factory is only created if the `cache` *(page 26)* setting is set to `on`.

The view cache manager configuration does not include a `param` key. This configuration is done via the `view_cache` factory, which defines the underlying cache object used by the view cache manager.

7. `http://www.php.net/session_set_cookie_params`
8. `http://www.php.net/session_cache_limiter`

view_cache

sfContext Accessor: none (used directly by the `view_cache_manager` factory)

Default configuration:

Listing 5-12
```
view_cache:
  class: sfFileCache
  param:
    automatic_cleaning_factor: 0
    cache_dir:                 %SF_TEMPLATE_CACHE_DIR%
    lifetime:                  86400
    prefix:                    %SF_APP_DIR%/template
```

This factory is only defined if the `cache` *(page 26)* setting is set to `on`.

The `view_cache` factory defines a cache class that must inherit from `sfCache` (see the Cache section for more information).

i18n

sfContext Accessor: `$context->getI18N()`

Default configuration:

Listing 5-13
```
i18n:
  class: sfI18N
  param:
    source:               XLIFF
    debug:                off
    untranslated_prefix:  "[T]"
    untranslated_suffix:  "[/T]"
    cache:
      class: sfFileCache
      param:
        automatic_cleaning_factor: 0
        cache_dir:                 %SF_I18N_CACHE_DIR%
        lifetime:                  31556926
        prefix:                    %SF_APP_DIR%/i18n
```

This factory is only defined if the `i18n` *(page 26)* setting is set to `on`.

source

The `source` option defines the container type for translations.

Built-in containers: `XLIFF`, `SQLite`, `MySQL`, and `gettext`.

debug

The `debug` option sets the debugging mode. If set to `on`, un-translated messages are decorated with a prefix and a suffix (see below).

untranslated_prefix

The `untranslated_prefix` defines a prefix to used for un-translated messages.

untranslated_suffix

The `untranslated_suffix` defines a suffix to used for un-translated messages.

cache

The `cache` option defines a anonymous cache factory to be used for caching i18n data (see the Cache section for more information).

routing

sfContext Accessor: `$context->getRouting()`

Default configuration:

Listing 5-14

```
routing:
  class: sfPatternRouting
  param:
    load_configuration:              true
    suffix:                          ''
    default_module:                  default
    default_action:                  index
    debug:                           %SF_DEBUG%
    logging:                         %SF_LOGGING_ENABLED%
    generate_shortest_url:           true
    extra_parameters_as_query_string: true
    cache:
      class: sfFileCache
      param:
        automatic_cleaning_factor: 0
        cache_dir:                 %SF_CONFIG_CACHE_DIR%/routing
        lifetime:                  31556926
        prefix:                    %SF_APP_DIR%/routing
```

variable_prefixes

Default: `:`

The `variable_prefixes` option defines the list of characters that starts a variable name in a route pattern.

segment_separators

Default: `/` and `.`

The `segment_separators` option defines the list of route segment separators. Most of the time, you don't want to override this option for the whole routing, but for specific routes.

generate_shortest_url

Default: `true` for new projects, `false` for upgraded projects

If set to `true`, the `generate_shortest_url` option will tell the routing system to generate the shortest route possible. Set it to `false` if you want your routes to be backward compatible with symfony 1.0 and 1.1.

`extra_parameters_as_query_string`

Default: `true` for new projects, `false` for upgraded projects

When some parameters are not used in the generation of a route, the `extra_parameters_as_query_string` allows those extra parameters to be converted to a query string. Set it to `false` to fallback to the behavior of symfony 1.0 or 1.1. In those versions, the extra parameters were just ignored by the routing system.

`cache`

The `cache` option defines an anonymous cache factory to be used for caching routing configuration and data (see the Cache section for more information).

`suffix`

Default: none

The default suffix to use for all routes. This option is deprecated and is not useful anymore.

`load_configuration`

Default: `true`

The `load_configuration` option defines whether the `routing.yml` files must be automatically loaded and parsed. Set it to `false` if you want to use the routing system of symfony outside of a symfony project.

`lazy_routes_deserialize`

Default: `false`

If set to `true`, the `lazy_routes_deserialize` setting enables lazy unserialization of the routing cache. It can improve the performance of your applications if you have a large number of routes and if most matching routes are among the first ones. It is strongly advised to test the setting before deploying to production, as it can harm your performance in certain circumstances.

This setting is only available for symfony 1.2.7 and up.

`lookup_cache_dedicated_keys`

Default: `false`

The `lookup_cache_dedicated_keys` setting determines how the routing cache is constructed. When set to `false`, the cache is stored as one big value; when set to `true`, each route has its own cache store. This setting is a performance optimization setting.

As a rule of thumb, setting this to `false` is better when using a file-based cache class (`sfFileCache` for instance), and setting it to `true` is better when using a memory-based cache class (`sfAPCCache` for instance).

This setting is only available for symfony 1.2.7 and up.

logger

sfContext Accessor: `$context->getLogger()`

Default configuration:

Listing 5-15

```
logger:
  class: sfAggregateLogger
  param:
    level: debug
    loggers:
      sf_web_debug:
        class: sfWebDebugLogger
        param:
          level: debug
          condition:       %SF_WEB_DEBUG%
          xdebug_logging:  true
          web_debug_class: sfWebDebug
      sf_file_debug:
        class: sfFileLogger
        param:
          level: debug
          file: %SF_LOG_DIR%/%SF_APP%_%SF_ENVIRONMENT%.log
```

Default configuration for the `prod` *environment*:

Listing 5-16

```
logger:
  class:   sfNoLogger
  param:
    level:   err
    loggers: ~
```

This factory is always defined, but the logging only occurs if the `logging_enabled` setting is set to `on`.

level

The `level` option defines the level of the logger.

Possible values: `EMERG`, `ALERT`, `CRIT`, `ERR`, `WARNING`, `NOTICE`, `INFO`, or `DEBUG`.

loggers

The `loggers` option defines a list of loggers to use. The list is an array of anonymous logger factories.

Built-in logger classes: `sfConsoleLogger`, `sfFileLogger`, `sfNoLogger`, `sfStreamLogger`, and `sfVarLogger`.

controller

sfContext Accessor: `$context->getController()`

Default configuration:

Listing 5-17
```
controller:
  class: sfFrontWebController
```

Anonymous Cache Factories

Several factories (`view_cache`, `i18n`, and `routing`) can take advantage of a cache object if defined in their configuration. The configuration of the cache object is similar for all factories. The `cache` key defines an anonymous cache factory. Like any other factory, it takes a `class` and a `param` entries. The `param` entry can take any option available for the given cache class.

The `prefix` option is the most important one as it allows to share or separate a cache between different environments/applications/projects.

Built-in cache classes: `sfAPCCache`, `sfEAcceleratorCache`, `sfFileCache`, `sfMemcacheCache`, `sfNoCache`, `sfSQLiteCache`, and `sfXCacheCache`.

The generator.yml Configuration File

The admin generator of symfony allows the creation of a backend interface for your model classes. It works whether you use Propel or Doctrine as your ORM.

Creation

Admin generator modules are created by the `propel:generate-admin` or `doctrine:generate-admin` tasks:

Listing 6-1

```
$ php symfony propel:generate-admin backend Article

$ php symfony doctrine:generate-admin backend Article
```

The above command creates an `article` admin generator module for the `Article` model class.

The `generator.yml` configuration file is cached as a PHP file; the process is automatically managed by the `sfGeneratorConfigHandler` class *(page 80)*.

Configuration File

The configuration of such a module can be done in the `apps/backend/modules/job/article/generator.yml` file:

Listing 6-2

```
generator:
  class: sfPropelGenerator
  param:
    # An array of parameters
```

The file contains two main entries: `class` and `param`. The class is `sfPropelGenerator` for Propel and `sfDoctrineGenerator` for Doctrine.

The `param` entry contains the configuration options for the generated module. The `model_class` defines the model class bound to this module, and the `theme` option defines the default theme to use.

But the main configuration is done under the `config` entry. It is organized into seven sections:

- `actions`: Default configuration for the actions found on the list and on the forms
- `fields`: Default configuration for the fields
- `list`: Configuration for the list
- `filter`: Configuration for the filters
- `form`: Configuration for the new/edit form
- `edit`: Specific configuration for the edit page

- `new`: Specific configuration for the new page

When first generated, all sections are defined as empty, as the admin generator defines sensible defaults for all possible options:

Listing 6-3
```
generator:
  param:
    config:
      actions: ~
      fields:  ~
      list:    ~
      filter:  ~
      form:    ~
      edit:    ~
      new:     ~
```

This document describes all possible options you can use to customize the admin generator through the `config` entry.

All options are available for both Propel and Doctrine and works the same if not stated otherwise.

Fields

A lot of options take a list of fields as an argument. A field can be a real column name, or a virtual one. In both cases, a getter must be defined in the model class (`get` suffixed by the camel-cased field name).

Based on the context, the admin generator is smart enough to know how to render fields. To customize the rendering, you can create a partial or a component. By convention, partials are prefixed with an underscore (_), and components by a tilde (~):

Listing 6-4
```
display: [_title, ~content]
```

In the above example, the `title` field will be rendered by the `title` partial, and the `content` field by the `content` component.

The admin generator passes some parameters to partials and components:

- For the `new` and `edit` page:
  - `form`: The form associated with the current model object
  - `attributes`: An array of HTML attributes to be applied to the widget
- For the `list` page:
  - `type`: `list`
  - `MODEL_NAME`: The current object instance, where `MODEL_NAME` is the model class name lowercased.

In an `edit` or `new` page, if you want to keep the two column layout (field label and widget), the partial or component template should follow this template:

Listing 6-5
```
<div class="sf_admin_form_row">
  <label>
    <!-- Field label or content to be displayed in the first column -->
  </label>
```

```
  <!-- Field widget or content to be displayed in the second column -->
</div>
```

Object Placeholders

Some options can take model object placeholders. A placeholder is a string which follows the pattern: `%%NAME%%`. The `NAME` string can be anything that can be converted to a valid object getter method (`get` suffixed by the camel-cased version of the `NAME` string). For instance, `%%title%%` will be replaced by the value of `$article->getTitle()`. Placeholder values are dynamically replaced at runtime according to the object associated with the current context.

When a model has a foreign key to another model, Propel and Doctrine define a getter for the related object. As for any other getter, it can be used as a placeholder if you define a meaningful `__toString()` method that converts the object to a string.

Configuration Inheritance

The admin generator configuration is based on a configuration cascade principle. The inheritance rules are the following:

- `new` and `edit` inherit from `form` which inherits from `fields`
- `list` inherits from `fields`
- `filter` inherits from `fields`

Credentials

Actions in the admin generator (on the list and on the forms) can be hidden, based on the user credentials using the `credential` option (see below). However, even if the link or button does not appear, the actions must still be properly secured from illicit access. The credential management in the admin generator only takes care of the display.

The `credential` option can also be used to hide columns on the list page.

Actions Customization

When configuration is not sufficient, you can override the generated methods:

Method	Description
`executeIndex()`	`list` view action
`executeFilter()`	Updates the filters
`executeNew()`	`new` view action
`executeCreate()`	Creates a new Job
`executeEdit()`	`edit` view action
`executeUpdate()`	Updates a Job
`executeDelete()`	Deletes a Job
`executeBatch()`	Executes a batch action
`executeBatchDelete()`	Executes the `_delete` batch action
`processForm()`	Processes the Job form
`getFilters()`	Returns the current filters

Method	Description
`setFilters()`	Sets the filters
`getPager()`	Returns the list pager
`getPage()`	Gets the pager page
`setPage()`	Sets the pager page
`buildCriteria()`	Builds the `Criteria` for the list
`addSortCriteria()`	Adds the sort `Criteria` for the list
`getSort()`	Returns the current sort column
`setSort()`	Sets the current sort column

Templates Customization

Each generated template can be overridden:

Template	Description
`_assets.php`	Renders the CSS and JS to use for templates
`_filters.php`	Renders the filters box
`_filters_field.php`	Renders a single filter field
`_flashes.php`	Renders the flash messages
`_form.php`	Displays the form
`_form_actions.php`	Displays the form actions
`_form_field.php`	Displays a singe form field
`_form_fieldset.php`	Displays a form fieldset
`_form_footer.php`	Displays the form footer
`_form_header.php`	Displays the form header
`_list.php`	Displays the list
`_list_actions.php`	Displays the list actions
`_list_batch_actions.php`	Displays the list batch actions
`_list_field_boolean.php`	Displays a single boolean field in the list
`_list_footer.php`	Displays the list footer
`_list_header.php`	Displays the list header
`_list_td_actions.php`	Displays the object actions for a row
`_list_td_batch_actions.php`	Displays the checkbox for a row
`_list_td_stacked.php`	Displays the stacked layout for a row
`_list_td_tabular.php`	Displays a single field for the list
`_list_th_stacked.php`	Displays a single column name for the header
`_list_th_tabular.php`	Displays a single column name for the header
`_pagination.php`	Displays the list pagination
`editSuccess.php`	Displays the `edit` view
`indexSuccess.php`	Displays the `list` view
`newSuccess.php`	Displays the `new` view

Look and Feel Customization

The look of the admin generator can be tweaked very easily as the generated templates define a lot of `class` and `id` HTML attributes.

In the `edit` or `new` page, each field HTML container has the following classes:

- `sf_admin_form_row`
- a class depending on the field type: `sf_admin_text`, `sf_admin_boolean`, `sf_admin_date`, `sf_admin_time`, or `sf_admin_foreignkey`.
- `sf_admin_form_field_COLUMN` where `COLUMN` is the column name

In the `list` page, each field HTML container has the following classes:

- a class depending on the field type: `sf_admin_text`, `sf_admin_boolean`, `sf_admin_date`, `sf_admin_time`, or `sf_admin_foreignkey`.
- `sf_admin_form_field_COLUMN` where `COLUMN` is the column name

Available Configuration Options

- `actions` *(page 50)*
  - `name` *(page 50)*
  - `action` *(page 50)*
  - `credentials` *(page 45)*
- `fields` *(page 49)*
  - `label` *(page 49)*
  - `help` *(page 49)*
  - `attributes` *(page 49)*
  - `credentials` *(page 45)*
  - `renderer` *(page 49)*
  - `renderer_arguments` *(page 50)*
- `list` *(page 50)*
  - `title` *(page 50)*
  - `display` *(page 50)*
  - `hide` *(page 51)*
  - `layout` *(page 51)*
  - `params` *(page 51)*
  - `sort` *(page 51)*
  - `max_per_page` *(page 52)*
  - `pager_class` *(page 52)*
  - `batch_actions` *(page 52)*
  - `object_actions` *(page 52)*
  - `actions` *(page 52)*
  - `peer_method` *(page 53)*
  - `peer_count_method` *(page 53)*
  - `table_method` *(page 53)*
  - `table_count_method` *(page 53)*
- `filter` *(page 53)*
  - `display` *(page 50)*
  - `class` *(page 54)*
- `form` *(page 54)*
  - `display` *(page 50)*
  - `class` *(page 54)*
- `edit` *(page 55)*
  - `title` *(page 50)*
  - `actions` *(page 52)*
- `new` *(page 55)*
  - `title` *(page 50)*
  - `actions` *(page 52)*

fields

The `fields` section defines the default configuration for each field. This configuration is defined for all pages and can be overridden on a page per page basis (`list`, `filter`, `form`, `edit`, and `new`).

label

Default: The humanized column name

The `label` option defines the label to use for the field:

Listing 6-6

```
config:
  fields:
    slug: { label: "URL shortcut" }
```

help

Default: none

The `help` option defines the help text to display for the field.

attributes

Default: `array()`

The `attributes` option defines the HTML attributes to pass to the widget:

Listing 6-7

```
config:
  fields:
    slug: { attributes: { class: foo } }
```

credentials

Default: none

The `credentials` option defines credentials the user must have for the field to be displayed. The credentials are only enforced for the object list.

Listing 6-8

```
config:
  fields:
    slug:      { credentials: [admin] }
    is_online: { credentials: [[admin, moderator]] }
```

The credential are to be defined with the same rules as in the `security.yml` configuration file.

renderer

Default: none

The `renderer` option defines a PHP callback to call to render the field. If defined, it overrides any other flag like the partial or component ones.

The callback is called with the value of the field and the arguments defined by the `renderer_arguments` option.

`renderer_arguments`

Default: `array()`

The `renderer_arguments` option defines the arguments to pass to the `renderer` PHP callback when rendering the field. It is only used if the `renderer` option is defined.

`actions`

The framework defines several built-in actions. They are all prefixed by an underscore (`_`). Each action can be customized with the options described in this section. The same options can be used when defining an action in the `list`, `edit`, or `new` entries.

`name`

Default: The action key

The `name` option defines the label to use for the action.

`action`

Default: Defined based on the action name

The `action` option defines the action name to execute without the `execute` prefix.

`credentials`

Default: none

The `credentials` option defines credentials the user must have for the action to be displayed.

The credentials are to be defined with the same rules as in the `security.yml` configuration file.

`list`

`title`

Default: The humanized model class name suffixed with “List”

The `title` option defines the title of the list page.

`display`

Default: All model columns, in the order of their definition in the schema file

The `display` option defines an array of ordered columns to display in the list.

An equal sign (`=`) before a column is a convention to convert the string to a link that goes to the `edit` page of the current object.

```
config:
  list:
    display: [=name, slug]
```

Listing 6-9

Also see the `hide` option to hide some columns.

`hide`

Default: none

The `hide` option defines the columns to hide from the list. Instead of specifying the columns to be displayed with the `display` option, it is sometimes faster to hide some columns:

```
config:
  list:
    hide: [created_at, updated_at]
```

Listing 6-10

If both the `display` and the `hide` options are provided, the `hide` option is ignored.

`layout`

Default: `tabular`

Possible values: `tabular` or `stacked`

The `layout` option defines what layout to use to display the list.

With the `tabular` layout, each column value is in its own table column.

With the `stacked` layout, each object is represented by a single string, which is defined by the `params` option (see below).

The `display` option is still needed when using the `stacked` layout as it defines the columns that will be sortable by the user.

`params`

Default value: none

The `params` option is used to define the HTML string pattern to use when using a `stacked` layout. This string can contain model object placeholders:

```
config:
  list:
    params:  |
      %%title%% written by %%author%% and published on %%published_at%%.
```

Listing 6-11

An equal sign (=) before a column is a convention to convert the string to a link that goes to the `edit` page of the current object.

`sort`

Default value: none

The `sort` option defines the default sort column. It is an array composed of two components: the column name and the sort order: `asc` or `desc`:

Listing 6-12
```
config:
  list:
    sort: [published_at, desc]
```

max_per_page

Default value: `20`

The `max_per_page` option defines the maximum number of objects to display on one page.

pager_class

Default value: `sfPropelPager` for Propel and `sfDoctrinePager` for Doctrine

The `pager_class` option defines the pager class to use when displaying the list.

batch_actions

Default value: `{ _delete: ~ }`

The `batch_actions` option defines the list of actions that can be executed for a selection of objects in the list.

If you don't define an `action`, the admin generator will look for a method named after the camel-cased name prefixed by `executeBatch`.

The executed method received the primary keys of the selected objects via the `ids` request parameter.

The batch actions feature can be disabled by setting the option to an empty array: `{}`

object_actions

Default value: `{ _edit: ~, _delete: ~ }`

The `object_actions` option defines the list of actions that can be executed on each object of the list.

If you don't define an `action`, the admin generator will look for a method named after the camel-cased name prefixed by `executeList`.

The object actions feature can be disabled by setting the option to an empty array: `{}`

actions

Default value: `{ _new: ~ }`

The `actions` option defines actions that take no object, like the creation of a new object.

If you don't define an `action`, the admin generator will look for a method named after the camel-cased name prefixed by `executeList`.

The object actions feature can be disabled by setting the option to an empty array: `{}`

`peer_method`

Default value: `doSelect`

The `peer_method` option defines the method to call to retrieve the objects to display in the list.

This option only exists for Propel. For Doctrine, use the `table_method` option.

`table_method`

Default value: `doSelect`

The `table_method` option defines the method to call to retrieve the objects to display in the list.

This option only exists for Doctrine. For Propel, use the `peer_method` option.

`peer_count_method`

Default value: `doCount`

The `peer_count_method` option defines the method to call to compute the number of objects for the current filter.

This option only exists for Propel. For Doctrine, use the `table_count_method` option.

`table_count_method`

Default value: `doCount`

The `table_count_method` option defines the method to call to compute the number of objects for the current filter.

This option only exists for Doctrine. For Propel, use the `peer_count_method` option.

`filter`

The `filter` section defines the configuration for the filtering form displayed on the list page.

`display`

Default value: All fields defined in the filter form class, in the order of their definition

The `display` option defines the ordered list of fields to display.

As filter fields are always optional, there is no need to override the filter form class to configure the fields to be displayed.

`class`

Default value: The model class name suffixed by `FormFilter`

The `class` option defines the form class to use for the `filter` form.

To completely remove the filtering feature, set the `class` to `false`.

`form`

The `form` section only exists as a fallback for the `edit` and `new` sections (see the inheritance rules in the introduction).

For form sections (`form`, `edit`, and `new`), the `label` and `help` options override the ones defined in the form classes.

`display`

Default value: All fields defined in the form class, in the order of their definition

The `display` option defines the ordered list of fields to display.

This option can also be used to arrange fields into groups:

Listing 6-13

```
# apps/backend/modules/job/config/generator.yml
config:
  form:
    display:
      Content: [title, body, author]
      Admin:   [is_published, expires_at]
```

The above configuration defines two groups (`Content` and `Admin`), each containing a subset of the form fields.

All the fields defined in the model form must be present in the `display` option. If not, it could lead to unexpected validation errors.

`class`

Default value: The model class name suffixed by `Form`

The `class` option defines the form class to use for the `edit` and `new` pages.

Even though you can define a `class` option in both the `new` and `edit` sections, it is better to use one class and take care of the differences using conditional logic.

edit

The `edit` section takes the same options as the `form` section.

title

Default: "Edit " suffixed by the humanized model class name

The `title` option defines the title heading of the edit page. It can contain model object placeholders.

actions

Default value: `{ _delete: ~, _list: ~, _save: ~ }`

The `actions` option defines actions available when submitting the form.

new

The `new` section takes the same options as the `form` section.

title

Default: "New " suffixed by the humanized model class name

The `title` option defines the title of the new page. It can contain model object placeholders.

Even if the object is new, it can have default values you want to output as part of the title.

actions

Default value: `{ _delete: ~, _list: ~, _save: ~, _save_and_add: ~ }`

The `actions` option defines actions available when submitting the form.

The databases.yml Configuration File

The `databases.yml` configuration allows for the configuration of the database connection. It is used by both ORMs bundled with symfony: Propel and Doctrine.

The main `databases.yml` configuration file for a project can be found in the `config/` directory.

Most of the time, all applications of a project share the same database. That's why the main database configuration file is in the project `config/` directory. You can of course override the default configuration by defining a `databases.yml` configuration file in your application configuration directories.

As discussed in the introduction, the `databases.yml` file is **environment-aware** *(page 20)*, benefits from the **configuration cascade mechanism** *(page 20)*, and can include **constants** *(page 18)*.

Each connection described in `databases.yml` must include a name, a database handler class name, and a set of parameters (`param`) used to configure the database object:

Listing 7-1
```
CONNECTION_NAME:
  class: CLASS_NAME
  param: { ARRAY OF PARAMETERS }
```

The `class` name should extend the `sfDatabase` base class.

If the database handler class cannot be autoloaded, a `file` path can be defined and will be automatically included before the factory is created:

Listing 7-2
```
CONNECTION_NAME:
  class: CLASS_NAME
  file:  ABSOLUTE_PATH_TO_FILE
```

The `databases.yml` configuration file is cached as a PHP file; the process is automatically managed by the `sfDatabaseConfigHandler` class *(page 80)*.

The database configuration can also be configured by using the `database:configure` task. This task updates the `databases.yml` according to the arguments you pass to it.

Propel

Default Configuration:

Listing 7-3

```
dev:
  propel:
    param:
      classname:  DebugPDO

test:
  propel:
    param:
      classname:  DebugPDO

all:
  propel:
    class:        sfPropelDatabase
    param:
      classname:  PropelPDO
      dsn:        mysql:dbname=##PROJECT_NAME##;host=localhost
      username:   root
      password:
      encoding:   utf8
      persistent: true
      pooling:    true
```

The following parameters can be customized under the `param` section:

Key	Description	Default Value
`classname`	The Propel adapter class	`PropelPDO`
`dsn`	The PDO DSN (required)	-
`username`	The database username	-
`password`	The database password	-
`pooling`	Whether to enable pooling	`true`
`encoding`	The default charset	`UTF-8`
`persistent`	Whether to create persistent connections	`false`
`options`	A set of Propel options	-

Doctrine

Default Configuration:

Listing 7-4

```
all:
  doctrine:
    class:        sfDoctrineDatabase
    param:
      dsn:        mysql:dbname=##PROJECT_NAME##;host=localhost
      username:   root
      password:
      attributes:
        quote_identifier: false
        use_native_enum: false
```

```
        validate: all
        idxname_format: %s_idx
        seqname_format: %s_seq
        tblname_format: %s
```

The following parameters can be customized under the `param` section:

Key	Description	Default Value
`dsn`	The PDO DSN (required)	-
`username`	The database username	-
`password`	The database password	-
`encoding`	The default charset	`UTF-8`
`attributes`	A set of Doctrine attributes	-

The following attributes can be customized under the `attributes` section:

Key	Description	Default Value
`quote_identifier`	Whether to wrap identifiers with quotes	`false`
`use_native_enum`	Whether to use native enums	`false`
`validate`	Whether to enable data validation	`true`
`idxname_format`	Format for index names	`%s_idx`
`seqname_format`	Format for sequence names	`%s_seq`
`tblname_format`	Format for table names	`%s`

The security.yml Configuration File

The `security.yml` configuration file describes the authentication and authorization rules for a symfony application.

The configuration information from the `security.yml` file is used by the `user` *(page 35)* factory class (`sfBasicSecurityUser` by default). The enforcement of the authentication and authorization is done by the `security` filter *(page 74)*.

When an application is created, symfony generates a default `security.yml` file in the application `config/` directory which describes the security for the whole application (under the `default` key):

Listing 8-1

```
default:
  is_secure: off
```

As discussed in the introduction, the `security.yml` file benefits from the **configuration cascade mechanism** *(page 20)*, and can include **constants** *(page 18)*.

The default application configuration can be overridden for a module by creating a `security.yml` file in the `config/` directory of the module. The main keys are action names without the `execute` prefix (`index` for the `executeIndex` method for instance).

To determine if an action is secure or not, symfony looks for the information in the following order:

- a configuration for the specific action in the module configuration file if it exists;
- a configuration for the whole module in the module configuration file if it exists (under the `all` key);
- the default application configuration (under the `default` key).

The same precedence rules are used to determine the credentials needed to access an action.

The `security.yml` configuration file is cached as a PHP file; the process is automatically managed by the `sfSecurityConfigHandler` class *(page 80)*.

Authentication

The default configuration of `security.yml`, installed by default for each application, authorizes access to anybody:

Listing 8-2

```
default:
  is_secure: off
```

By setting the `is_secure` key to `on` in the application `security.yml` file, the entire application will require authentication for all users.

When an un-authenticated user tries to access a secured action, symfony forwards the request to the `login` action configured in `settings.yml`.

To modify authentication requirements for a module, create a `security.yml` file in the `config/` directory of the module and define an `all` key:

Listing 8-3
```
all:
  is_secure: on
```

To modify authentication requirements for a single action of a module, create a `security.yml` file in the `config/` directory of the module and define a key after the name of the action:

Listing 8-4
```
index:
  is_secure: off
```

It is not possible to secure the login action. This is to avoid infinite recursion.

Authorization

When a user is authenticated, the access to some actions can be even more restricted by defining credentials. When credentials are defined, a user must have the required credentials to access the action:

Listing 8-5
```
all:
  is_secure:   on
  credentials: admin
```

The credential system of symfony is simple and powerful. A credential is a string that can represent anything you need to describe the application security model (like groups or permissions).

The `credentials` key supports Boolean operations to describe complex credential requirements by using the notation array.

If a user must have the credential A **and** the credential B, wrap the credentials with square brackets:

Listing 8-6
```
index:
  credentials: [A, B]
```

If a user must have credential the A **or** the credential B, wrap them with two pairs of square brackets:

Listing 8-7
```
index:
  credentials: [[A, B]]
```

You can also mix and match brackets to describe any kind of Boolean expression with any number of credentials.

The cache.yml Configuration File

The `cache.yml` configuration file describes the cache configuration for the view layer. This configuration file is only active if the `cache` *(page 26)* setting is enabled in `settings.yml`.

The configuration of the class used for caching and its associated configuration is to be done in the `view_cache_manager` *(page 37)* and `view_cache` *(page 38)* sections of the `factories.yml` configuration file.

When an application is created, symfony generates a default `cache.yml` file in the application `config/` directory which describes the cache for the whole application (under the `default` key). By default, the cache is globally set to `off`:

Listing 9-1

```
default:
  enabled:     off
  with_layout: false
  lifetime:    86400
```

As the `enabled` setting is set to `false` by default, you need to enable the cache selectively. You can also work the other way around: enable the cache globally and then, disable it on specific pages that cannot be cached. Your approach should depend on what represents less work for your application.

As discussed in the introduction, the `cache.yml` file benefits from the **configuration cascade mechanism** *(page 20)*, and can include **constants** *(page 18)*.

The `cache.yml` configuration file is cached as a PHP file; the process is automatically managed by the `sfCacheConfigHandler` class *(page 80)*.

The default application configuration can be overridden for a module by creating a `cache.yml` file in the `config/` directory of the module. The main keys are action names without the `execute` prefix (`index` for the `executeIndex` method for instance). A partial or component can also be cached by using its name prefixed with an underscore (_).

To determine if an action is cached or not, symfony looks for the information in the following order:

- a configuration for the specific action, partial, or component in the module configuration file, if it exists;
- a configuration for the whole module in the module configuration file, if it exists (under the `all` key);
- the default application configuration (under the `default` key).

An incoming request with `GET` parameters in the query string or submitted with the `POST`, `PUT`, or `DELETE` method will never be cached by symfony, regardless of the configuration.

enabled

Default: `off`

The `enabled` setting enables or disables the cache for the current scope.

with_layout

Default: `false`

The `with_layout` setting determines whether the cache must be for the entire page (`true`), or for the action only (`false`).

The `with_layout` option is not taken into account for partial and component caching as they cannot be decorated by a layout.

lifetime

Default: `86400`

The `lifetime` setting defines the server-side lifetime of the cache in seconds (`86400` seconds equals one day).

client_lifetime

Default: Same value as the `lifetime` one

The `client_lifetime` setting defines the client-side lifetime of the cache in seconds.

This setting is used to automatically set the `Expires` header and the `max-cache` cache control variable, unless a `Last-Modified` or `Expires` header has already been set.

You can disable client-side caching by setting the value to `0`.

contextual

Default: `false`

The `contextual` setting determines if the cache depends on the current page context or not. The setting is therefore only meaningful when used for partials and components.

When a partial output is different depending on the template in which it is included, the partial is said to be contextual, and the `contextual` setting must be set to `true`. By default, the setting is set to `false`, which means that the output for partials and components are always the same, wherever it is included.

The cache is still obviously different for a different set of parameters.

The routing.yml Configuration File

The `routing.yml` configuration file allows the definition of routes.

The main `routing.yml` configuration file for an application can be found in the `apps/APP_NAME/config/` directory.

The `routing.yml` configuration file contains a list of named route definitions:

Listing 10-1

```
ROUTE_1:
  # definition of route 1

ROUTE_2:
  # definition of route 2

# ...
```

When a request comes in, the routing system tries to match a route to the incoming URL. The first route that matches wins, so the order in which routes are defined in the `routing.yml` configuration file is important.

When the `routing.yml` configuration file is read, each route is converted to an object of class `class`:

Listing 10-2

```
ROUTE_NAME:
  class: CLASS_NAME
  # configuration if the route
```

The `class` name should extend the `sfRoute` base class. If not provided, the `sfRoute` base class is used as a fallback.

The `routing.yml` configuration file is cached as a PHP file; the process is automatically managed by the `sfRoutingConfigHandler` class *(page 80)*.

Route Classes

Route Configuration

The `routing.yml` configuration file supports several settings to further configure the routes. These settings are used by the `sfRoutingConfigHandler` class to convert each route to an object.

`class`

Default: `sfRoute` (or `sfRouteCollection` if `type` is `collection`, see below)

The `class` setting allows to change the route class to use for the route.

`url`

Default: `/`

The `url` setting is the pattern that must match an incoming URL for the route to be used for the current request.

The pattern is made of segments:

- variables (a word prefixed with a colon `:` *(page 39)*)
- constants
- a wildcard (*) to match a sequence of key/value pairs

Each segment must be separated by one of the pre-defined separator (`/` or `.` by default *(page 39)*).

`params`

Default: An empty array

The `params` setting defines an array of parameters associated with the route. They can be default values for variables contained in the `url`, or any other variable relevant for this route.

`param`

Default: An empty array

This setting is equivalent to the `params` settings.

`options`

Default: An empty array

The `options` setting is an array of options to be passed to the route object to further customize its behavior. The following sections describe the available options for each route class.

`requirements`

Default: An empty array

The `requirements` settings is an array of requirements that must be satisfied by the `url` variables. The keys are the url variables and the values are regular expressions that the variable values must match.

The regular expression will be included in a another regular expression, and as such, you don't need to wrap them between separators, nor do you need to bound them with ^ or $ to match the whole value.

type

Default: `null`

If set to `collection`, the route will be read as a route collection.

This setting is automatically set to `collection` by the config handler class if the `class` name contains the word `Collection`. It means that most of the time, you do not need to use this setting.

sfRoute

All route classes extends the `sfRoute` base class, which provides the required settings to configure a route.

sfRequestRoute

sf_method

Default: `get`

The `sf_method` option is to be used in the `requirements` array. It enforces the HTTP request in the route matching process.

sfObjectRoute

All the following options of `sfObjectRoute` must be used inside the `options` setting of the `routing.yml` configuration file.

model

The `model` option is mandatory and is the name of the model class to be associated with the current route.

type

The `type` option is mandatory and is the type of route you want for your model; it can be either `object` or `list`. A route of type `object` represents a single model object, and a route of type `list` represents a collection of model objects.

method

The `method` option is mandatory. It is the method to call on the model class to retrieve the object(s) associated with this route. This must be a static method. The method is called with the parameters of the parsed route as an argument.

allow_empty

Default: true

If the allow_empty option is set to false, the route will throw a 404 exception if no object is returned by the call to the model method.

convert

Default: toParams

The convert option is a method to call to convert a model object to an array of parameters suitable for generating a route based on this model object. It must returns an array with at least the required parameters of the route pattern (as defined by the url setting).

sfPropelRoute

method_for_criteria

Default: doSelect for collections, doSelectOne for single objects

The method_for_criteria option defines the method called on the model Peer class to retrieve the object(s) associated with the current request. The method is called with the parameters of the parsed route as an argument.

sfDoctrineRoute

method_for_query

Default: none

The method_for_query option defines the method to call on the model to retrieve the object(s) associated with the current request. The current query object is passed as an argument.

If the option is not set, the query is just "executed" with the execute() method.

sfRouteCollection

The sfRouteCollection base class represents a collection of routes.

sfObjectRouteCollection

model

The model option is mandatory and is the name of the model class to be associated with the current route.

actions

Default: false

The `actions` option defines an array of authorized actions for the route. The actions must be a sub-set of all available actions: `list`, `new`, `create`, `edit`, `update`, `delete`, and `show`.

If the option is set to `false`, the default, all actions will be available except for the `show` one if the `with_show` option is set to `false` (see below).

`module`

Default: The route name

The `module` option defines the module name.

`prefix_path`

Default: `/` followed by the route name

The `prefix_path` option defines a prefix to prepend to all `url` patterns. It can be any valid pattern and can contain variables and several segments.

`column`

Default: `id`

The `column` option defines the column of the model to use as the unique identifier for the model object.

`with_show`

Default: `true`

The `with_show` option is used when the `actions` option is set to `false` to determine if the `show` action must be included in the list of authorized actions for the route.

`segment_names`

Default: array('edit' => 'edit', 'new' => 'new'),

The `segment_names` defines the words to use in the `url` patterns for the `edit` and `new` actions.

`model_methods`

Default: An empty array

The `model_methods` options defines the methods to call to retrieve the object(s) from the model (see the `method` option of `sfObjectRoute`). This is actually an array defining the `list` and the `object` methods:

Listing 10-3

```
model_methods:
  list:   getObjects
  object: getObject
```

`requirements`

Default: `\d+` for the `column`

The `requirements` option defines an array of requirements to apply to the route variables.

`with_wildcard_routes`

Default: `false`

The `with_wildcard_routes` option allows for any action to be accessed via two wildcard routes: one for a single object, and another for object collections.

`route_class`

Default: `sfObjectRoute`

The `route_class` option can override the default route object used for the collection.

`collection_actions`

Default: An empty array

The `collection_actions` options defines an array of additional actions available for the collection routes.

`object_actions`

Default: An empty array

The `object_actions` options defines an array of additional actions available for the object routes.

`sfPropelRouteCollection`

The `sfPropelRouteCollection` route class extends the `sfRouteCollection`, and changes the default route class to `sfPropelRoute` (see the `route_class` option above).

`sfDoctrineRouteCollection`

The `sfDoctrineRouteCollection` route class extends the `sfRouteCollection`, and changes the default route class to `sfDoctrineRoute` (see the `route_class` option above).

The app.yml Configuration File

The symfony framework provides a built-in configuration file for application specific settings, the `app.yml` configuration file.

This YAML file can contain any setting you want that makes sense for your specific application. In the code, these settings are available through the global `sfConfig` class, and keys are prefixed with the `app_` string:

```
sfConfig::get('app_active_days');
```

Listing 11-1

All settings are prefixed by `app_` because the `sfConfig` class also provides access to symfony settings *(page 18)* and project directories *(page 19)*.

As discussed in the introduction, the `app.yml` file is **environment-aware** *(page 20)*, and benefits from the **configuration cascade mechanism** *(page 20)*.

The `app.yml` configuration file is a great place to define settings that change based on the environment (an API key for instance), or settings that can evolve over time (an email address for instance). It is also the best place to define settings that need to be changed by someone who does not necessarily understand symfony or PHP (a system administrator for instance).

Refrain from using `app.yml` to bundle application logic.

The `app.yml` configuration file is cached as a PHP file; the process is automatically managed by the `sfDefineEnvironmentConfigHandler` class *(page 80)*.

The filters.yml Configuration File

The `filters.yml` configuration file describes the filter chain to be executed for every request.

The main `filters.yml` configuration file for an application can be found in the `apps/APP_NAME/config/` directory.

As discussed in the introduction, the `filters.yml` file benefits from the **configuration cascade mechanism** *(page 20)*, and can include **constants** *(page 18)*.

The `filters.yml` configuration file contains a list of named filter definitions:

Listing 12-1
```
FILTER_1:
  # definition of filter 1

FILTER_2:
  # definition of filter 2

# ...
```

When the controller initializes the filter chain for a request, it reads the `filters.yml` file and registers the filters by looking for the class name of the filter (`class`) and the parameters (`param`) used to configure the filter object:

Listing 12-2
```
FILTER_NAME:
  class: CLASS_NAME
  param: { ARRAY OF PARAMETERS }
```

The filters are executed in the same order as they appear in the configuration file. As symfony executes the filters as a chain, the first registered filter is executed first and last.

The `class` name should extend the `sfFilter` base class.

If the filter class cannot be autoloaded, a `file` path can be defined and will be automatically included before the filter object is created:

Listing 12-3
```
FACTORY_NAME:
  class: CLASS_NAME
  file:  ABSOLUTE_PATH_TO_FILE
```

When you override the `filters.yml` file, you must keep all filters from the inherited configuration file:

Listing 12-4
```
rendering: ~
security:  ~
cache:     ~
common:    ~
execution: ~
```

To remove a filter, you need to disable it by setting the `enabled` key to `false`:

Listing 12-5

```
FACTORY_NAME:
  enabled: false
```

There are two special name filters: `rendering` and `execution`. They are both mandatory and are identified with the `type` parameter. The `rendering` filter should always be the first registered filter and the `execution` filter should be the last one:

Listing 12-6

```
rendering:
  class: sfRenderingFilter
  param:
    type: rendering

# ...

execution:
  class:  sfExecutionFilter
  param:
    type: execution
```

The `filters.yml` configuration file is cached as a PHP file; the process is automatically managed by the `sfFilterConfigHandler` class *(page 80)*.

Filters

`rendering`

Default configuration:

Listing 12-7
```
rendering:
  class: sfRenderingFilter
  param:
    type: rendering
```

The rendering filter is responsible for the output of the response to the browser. As it should be the first filter registered, it is also the last one to have a chance to manage the request.

`security`

Default configuration:

Listing 12-8
```
security:
  class: sfBasicSecurityFilter
  param:
    type: security
```

The security filter checks the security by calling the `getCredential()` method of the action. Once the credential has been acquired, it verifies that the user has the same credential by calling the `hasCredential()` method of the user object.

The security filter must have a type of `security`.

The fine-grained configuration of the security filter is done via the `security.yml` configuration file *(page 59)*.

If the requested action is not configured as secure in `security.yml`, the security filter will not be executed.

`cache`

Default configuration:

Listing 12-9
```
cache:
  class: sfCacheFilter
  param:
    condition: %SF_CACHE%
```

The cache filter manages the caching of actions and pages. It is also responsible for adding the needed HTTP cache headers to the response (`Last-Modified`, `ETag`, `Cache-Control`, `Expires`, ...).

common

Default configuration:

Listing 12-10

```
common:
  class: sfCommonFilter
```

The common filter adds JavaScripts and stylesheets tags to the main response if they have not been already included.

If you use the `include_stylesheets()` and `include_javascripts()` helpers in your layout, you can safely disable this filter, and benefit from a small performance boost.

execution

Default configuration:

Listing 12-11

```
execution:
  class:  sfExecutionFilter
  param:
    type: execution
```

The execution filter is at the center of the filter chain and does all action and view execution.

The execution filter should be the last registered filter.

The view.yml Configuration File

The View layer can be configured by editing the `view.yml` configuration file.

As discussed in the introduction, the `view.yml` file benefits from the **configuration cascade mechanism** *(page 20)*, and can include **constants** *(page 18)*.

This configuration file is mostly deprecated in favors of helpers used directly in the templates or methods called from actions.

The `view.yml` configuration file contains a list of view configurations:

Listing 13-1
```
VIEW_NAME_1:
  # configuration

VIEW_NAME_2:
  # configuration

# ...
```

The `view.yml` configuration file is cached as a PHP file; the process is automatically managed by the `sfViewConfigHandler` class *(page 80)*.

Layout

Default configuration:

Listing 13-2
```
default:
  has_layout: on
  layout:     layout
```

The `view.yml` configuration file defines the default layout used by the application. By default, the name is `layout`, and so symfony decorates every page with the `layout.php` file, found in the application `templates/` directory. You can also disable the decoration process altogether by setting the `has_layout` entry to `false`.

The layout is automatically disabled for XML HTTP requests and non-HTML content types, unless explicitly set for the view.

Stylesheets

Default Configuration:

```
default:
  stylesheets: [main.css]
```

Listing 13-3

The `stylesheets` entry defines an array of stylesheets to use for the current view.

The inclusion of the stylesheets defined in `view.yml` can be done either manually with the `include_stylesheets()` helper, or automatically with the common filter *(page 75)*.

If many files are defined, symfony will include them in the same order as the definition:

```
stylesheets: [main.css, foo.css, bar.css]
```

Listing 13-4

You can also change the `media` attribute or omit the `.css` suffix:

```
stylesheets: [main, foo.css, bar.css, print.css: { media: print }]
```

Listing 13-5

This setting is *deprecated* in favor of the `use_stylesheet()` helper:

```
<?php use_stylesheet('main.css') ?>
```

Listing 13-6

In the default `view.yml` configuration file, the referenced file is `main.css`, and not `/css/main.css`. As a matter of fact, both definitions are equivalent as symfony prefixes relative paths with `/css/`.

JavaScripts

Default Configuration:

```
default:
  javascripts: []
```

Listing 13-7

The `javascripts` entry defines an array of JavaScript files to use for the current view.

The inclusion of the JavaScript files defined in `view.yml` can be done either manually with the `include_javascripts()` helper, or automatically with the common filter *(page 75)*.

If many files are defined, symfony will include them in the same order as the definition:

```
javascripts: [foo.js, bar.js]
```

Listing 13-8

You can also omit the `.js` suffix:

```
javascripts: [foo, bar]
```

Listing 13-9

This setting is *deprecated* in favor of the `use_javascript()` helper:

```
<?php use_javascript('foo.js') ?>
```

Listing 13-10

When using relative paths, like `foo.js`, symfony prefixes them with `/js/`.

Metas and HTTP Metas

Default Configuration:

Listing 13-11
```
default:
  http_metas:
    content-type: text/html

  metas:
    #title:        symfony project
    #description:  symfony project
    #keywords:     symfony, project
    #language:     en
    #robots:       index, follow
```

The `http_metas` and `metas` settings allows the definition of meta tags to be included in the layout.

The inclusion of the meta tags defined in `view.yml` can be done manually with the `include_metas()` and `include_http_metas()` helpers.

These settings are *deprecated* in favor of pure HTML in the layout for static metas (like the content type), or in favor of a slot for dynamic metas (like the title or the description).

When it makes sense, the `content-type` HTTP meta is automatically modified to include the charset defined in the `settings.yml` configuration file *(page 25)* if not already present.

Other Configuration Files

This chapter describes other symfony configuration files, that rarely need to be changed.

`autoload.yml`

The `autoload.yml` configuration determines which directories need to be autoloaded by symfony. Each directory is scanned for PHP classes and interfaces.

As discussed in the introduction, the `autoload.yml` file benefits from the **configuration cascade mechanism** *(page 20)*, and can include **constants** *(page 18)*.

The `autoload.yml` configuration file is cached as a PHP file; the process is automatically managed by the `sfAutoloadConfigHandler` class *(page 80)*.

The default configuration is fine for most projects:

Listing 14-1

```
autoload:
  # project
  project:
    name:           project
    path:           %SF_LIB_DIR%
    recursive:      on
    exclude:        [model, symfony]

  project_model:
    name:           project model
    path:           %SF_LIB_DIR%/model
    recursive:      on

  # application
  application:
    name:           application
    path:           %SF_APP_LIB_DIR%
    recursive:      on

  modules:
    name:           module
    path:           %SF_APP_DIR%/modules/*/lib
    prefix:         1
    recursive:      on
```

Each configuration has a name and must be set under a key with that name. It allows for the default configuration to be overridden.

As you can see, the `lib/vendor/symfony/` directory is excluded by default, as symfony uses a different autoloading mechanism for core classes.

Several keys can be used to customize the autoloading behavior:

- `name`: A description
- `path`: The path to autoload
- `recursive`: Whether to look for PHP classes in sub-directories
- `exclude`: An array of directory names to exclude from the search
- `prefix`: Set to `true` if the classes found in the path should only be autoloaded for a given module (`false` by default)
- `files`: An array of files to explicitly parse for PHP classes
- `ext`: The extension of PHP classes (`.php` by default)

For instance, if you embed a large library within your project under the `lib/` directory, and if it already supports autoloading, you can exclude it from the symfony default autoloading system to benefit from a performance boost by modifying the `project` autoload configuration:

Listing 14-2

```
autoload:
  project:
    name:           project
    path:           %SF_LIB_DIR%
    recursive:      on
    exclude:        [model, symfony, vendor/large_lib]
```

config_handlers.yml

The `config_handlers.yml` configuration file describes the configuration handler classes used to parse and interpret all other YAML configuration files. Here is the default configuration used to load the `settings.yml` configuration file:

Listing 14-3

```
config/settings.yml:
  class:    sfDefineEnvironmentConfigHandler
  param:
    prefix: sf_
```

Each configuration file is defined by a class (`class` entry) and can be further customized by defining some parameters under the `param` section.

The default `config_handlers.yml` file defines the parser classes as follows:

Configuration File	Config Handler Class
`autoload.yml`	`sfAutoloadConfigHandler`
`databases.yml`	`sfDatabaseConfigHandler`
`settings.yml`	`sfDefineEnvironmentConfigHandler`
`app.yml`	`sfDefineEnvironmentConfigHandler`
`factories.yml`	`sfFactoryConfigHandler`
`core_compile.yml`	`sfCompileConfigHandler`
`filters.yml`	`sfFilterConfigHandler`
`routing.yml`	`sfRoutingConfigHandler`

Configuration File	Config Handler Class
generator.yml	sfGeneratorConfigHandler
view.yml	sfViewConfigHandler
security.yml	sfSecurityConfigHandler
cache.yml	sfCacheConfigHandler
module.yml	sfDefineEnvironmentConfigHandler

core_compile.yml

The core_compile.yml configuration file describes the PHP files that are merged into one big file in the prod environment, to speed up the time it takes for symfony to load. By default, the main symfony core classes are defined in this configuration file. If your application relies on some classes that need to be loaded for each request, you can create a core_compile.yml configuration file in your project or application and add them to it. Here is an extract of the default configuration:

Listing 14-4

```
- %SF_SYMFONY_LIB_DIR%/autoload/sfAutoload.class.php
- %SF_SYMFONY_LIB_DIR%/action/sfComponent.class.php
- %SF_SYMFONY_LIB_DIR%/action/sfAction.class.php
- %SF_SYMFONY_LIB_DIR%/action/sfActions.class.php
```

As discussed in the introduction, the core_compile.yml file benefits from the **configuration cascade mechanism** *(page 20)*, and can include **constants** *(page 18)*.

The core_compile.yml configuration file is cached as a PHP file; the process is automatically managed by the sfCompileConfigHandler class *(page 80)*.

Events

The symfony core components are decoupled thanks to an `sfEventDispatcher` object. The event dispatcher manages the communication between core components.

Any object can notify an event to the dispatcher, and any other object can connect to the dispatcher to listen to a specific event.

An event is just a name composed of a namespace and a name separated by a dot (`.`).

Usage

You can notify an event by first creating an event object:

Listing 15-1
```
$event = new sfEvent($this, 'user.change_culture', array('culture' =>
$culture));
```

And notify it:

Listing 15-2
```
$dispatcher->notify($event);
```

The `sfEvent` constructor takes three arguments:

- The "subject" of the event (most of the time, this is the object notifying the event, but it can also be `null`)
- The event name
- An array of parameters to pass to the listeners

To listen for an event, connect to that event name:

Listing 15-3
```
$dispatcher->connect('user.change_culture', array($this,
'listenToChangeCultureEvent'));
```

The `connect` method takes two arguments:

- The event name
- A PHP callable to call when the event is notified

Here is an implementation example of a listener:

Listing 15-4
```
public function listenToChangeCultureEvent(sfEvent $event)
{
  // change the message format object with the new culture
  $this->setCulture($event['culture']);
}
```

The listener gets the event as the first argument. The event object has several methods to get event information:

- `getSubject()`: Gets the subject object attached to the event
- `getParameters()`: Returns the event parameters

The event object can also be accessed as an array to get its parameters.

Event Types

Events can be triggered by three different methods:

- `notify()`
- `notifyUntil()`
- `filter()`

`notify`

The `notify()` method notifies all listeners. The listeners cannot return a value and all listeners are guaranteed to be executed.

`notifyUntil`

The `notifyUntil()` method notifies all listeners until one stops the chain by returning a `true` value.

The listener that stops the chain may also call the `setReturnValue()` method.

The notifier can check if a listener has processed the event by calling the `isProcessed()` method:

Listing 15-5

```
if ($event->isProcessed())
{
  // ...
}
```

`filter`

The `filter()` method notifies all listeners that they can filter the given value, passed as a second argument by the notifier, and retrieved by the listener callable as the second argument. All listeners are passed the value and they must return the filtered value. All listeners are guaranteed to be executed.

The notifier can get the filtered value by calling the `getReturnValue()` method:

Listing 15-6

```
$ret = $event->getReturnValue();
```

Events

    - `user.change_culture` *(page 91)*
    - `user.method_not_found` *(page 92)*
    - `user.change_authentication` *(page 92)*
- `view` *(page 92)*
    - `view.configure_format` *(page 92)*
    - `view.method_not_found` *(page 93)*
- `view.cache` *(page 93)*
    - `view.cache.filter_content` *(page 93)*

application

application.log

Notify method: `notify`

Default notifiers: lot of classes

Parameter	Description
`priority`	The priority level (`sfLogger::EMERG`, `sfLogger::ALERT`, `sfLogger::CRIT`, `sfLogger::ERR`, `sfLogger::WARNING`, `sfLogger::NOTICE`, `sfLogger::INFO`, or `sfLogger::DEBUG`)

The `application.log` event is the mechanism used by symfony to do the logging for web request (see the logger factory). The event is notified by most symfony core components.

application.throw_exception

Notify method: `notifyUntil`

Default notifiers: `sfException`

The `application.throw_exception` event is notified when an uncaught exception is thrown during the handling of a request.

You can listen to this event to do something special whenever an uncaught exception is thrown(like sending an email, or logging the error). You can also override the default exception management mechanism of symfony by processing the event.

command

command.log

Notify method: `notify`

Default notifiers: `sfCommand*` classes

Parameter	Description
`priority`	The priority level (`sfLogger::EMERG`, `sfLogger::ALERT`, `sfLogger::CRIT`, `sfLogger::ERR`, `sfLogger::WARNING`, `sfLogger::NOTICE`, `sfLogger::INFO`, or `sfLogger::DEBUG`)

The `command.log` event is the mechanism used by symfony to do the logging for the symfony CLI utility (see the logger factory).

command.pre_command

Notify method: `notifyUntil`

Default notifiers: `sfTask`

Parameter	Description
`arguments`	An array of arguments passed on the CLI

Parameter	Description
`options`	An array of options passed on the CLI

The `command.pre_command` event is notified just before a task is executed.

`command.post_command`

Notify method: `notify`

Default notifiers: `sfTask`

The `command.post_command` event is notified just after a task is executed.

`command.filter_options`

Notify method: `filter`

Default notifiers: `sfTask`

Parameter	Description
`command_manager`	The `sfCommandManager` instance

The `command.filter_options` event is notified before the task CLI options are parsed. This event can be used to filter the options passed by the user.

configuration

`configuration.method_not_found`

Notify method: `notifyUntil`

Default notifiers: `sfProjectConfiguration`

Parameter	Description
`method`	The name of the called missing method
`arguments`	The arguments passed to the method

The `configuration.method_not_found` event is notified when a method is not defined in the `sfProjectConfiguration` class. By listening to this event, a method can be added to the class, without using inheritance.

component

`component.method_not_found`

Notify method: `notifyUntil`

Default notifiers: `sfComponent`

Parameter	Description
`method`	The name of the called missing method
`arguments`	The arguments passed to the method

The `component.method_not_found` event is notified when a method is not defined in the `sfComponent` class. By listening to this event, a method can be added to the class, without using inheritance.

context

`context.load_factories`

Notify method: `notify`

Default notifiers: `sfContext`

The `context.load_factories` event is notified once per request by the `sfContext` object just after all factories have been initialized. This is the first event to be notified with all core classes initialized.

controller

`controller.change_action`

Notify method: `notify`

Default notifiers: `sfController`

Parameter	Description
`module`	The module name to be executed
`action`	The action name to be executed

The `controller.change_action` is notified just before an action is executed.

`controller.method_not_found`

Notify method: `notifyUntil`

Default notifiers: `sfController`

Parameter	Description
`method`	The name of the called missing method
`arguments`	The arguments passed to the method

The `controller.method_not_found` event is notified when a method is not defined in the `sfController` class. By listening to this event, a method can be added to the class, without using inheritance.

`controller.page_not_found`

Notify method: `notify`

Default notifiers: `sfController`

Parameter	Description
`module`	The module name that generated the 404 error

Parameter	Description
`action`	The action name that generated the 404 error

The `controller.page_not_found` is notified whenever a 404 error is generated during the handling of a request.

You can listen to this event to do something special whenever a 404 page occurs, like sending an email, or logging the error. the event.

plugin

plugin.pre_install

Notify method: `notify`

Default notifiers: `sfPluginManager`

Parameter	Description
`channel`	The plugin channel
`plugin`	The plugin name
`is_package`	Whether the plugin to install is a local package (`true`), or a web package (`false`)

The `plugin.pre_install` event is notified just before a plugin will be installed.

plugin.post_install

Notify method: `notify`

Default notifiers: `sfPluginManager`

Parameter	Description
`channel`	The plugin channel
`plugin`	The plugin name

The `plugin.post_install` event is notified just after a plugin has been installed.

plugin.pre_uninstall

Notify method: `notify`

Default notifiers: `sfPluginManager`

Parameter	Description
`channel`	The plugin channel
`plugin`	The plugin name

The `plugin.pre_uninstall` event is notified just before a plugin will be uninstalled.

plugin.post_uninstall

Notify method: `notify`

Default notifiers: `sfPluginManager`

Parameter	Description
`channel`	The plugin channel
`plugin`	The plugin name

The `plugin.post_uninstall` event is notified just after a plugin has been uninstalled.

request

`request.filter_parameters`

Notify method: `filter`

Default notifiers: `sfWebRequest`

Parameter	Description
`path_info`	The request path

The `request.filter_parameters` event is notified when the request parameters are initialized.

`request.method_not_found`

Notify method: `notifyUntil`

Default notifiers: `sfRequest`

Parameter	Description
`method`	The name of the called missing method
`arguments`	The arguments passed to the method

The `request.method_not_found` event is notified when a method is not defined in the `sfRequest` class. By listening to this event, a method can be added to the class, without using inheritance.

response

`response.method_not_found`

Notify method: `notifyUntil`

Default notifiers: `sfResponse`

Parameter	Description
`method`	The name of the called missing method
`arguments`	The arguments passed to the method

The `response.method_not_found` event is notified when a method is not defined in the `sfResponse` class. By listening to this event, a method can be added to the class, without using inheritance.

response.filter_content

Notify method: `filter`

Default notifiers: `sfResponse`

The `response.filter_content` event is notified before a response is sent. By listening to this event, you can manipulate the content of the response before it is sent.

routing

routing.load_configuration

Notify method: `notify`

Default notifiers: `sfRouting`

The `routing.load_configuration` event is notified when the routing factory loads the routing configuration.

task

task.cache.clear

Notify method: `notifyUntil`

Default notifiers: `sfCacheClearTask`

Parameter	Description
`app`	The application name
`type`	The type of cache (`all`, `config`, `i18n`, `routing`, `module`, and `template`)
`env`	The environment

The `task.cache.clear` event is notified whenever the user clears the cache from the CLI with the `cache:clear` task.

template

template.filter_parameters

Notify method: `filter`

Default notifiers: `sfViewParameterHolder`

The `template.filter_parameters` event is notified before a view file is rendered. By listening to this event you can access and manipulate variables passed to a template.

user

user.change_culture

Notify method: `notify`

Default notifiers: `sfUser`

Parameter	Description
`culture`	The user culture

The `user.change_culture` event is notified when the user culture is changed during a request.

`user.method_not_found`

Notify method: `notifyUntil`

Default notifiers: `sfUser`

Parameter	Description
`method`	The name of the called missing method
`arguments`	The arguments passed to the method

The `user.method_not_found` event is notified when a method is not defined in the `sfUser` class. By listening to this event, a method can be added to the class, without using inheritance.

`user.change_authentication`

Notify method: `notify`

Default notifiers: `sfBasicSecurityUser`

Parameter	Description
`authenticated`	Whether the user is authenticated or not

The `user.change_authentication` event is notified whenever the user authentication status changes.

`view`

`view.configure_format`

Notify method: `notify`

Default notifiers: `sfView`

Parameter	Description
`format`	The requested format
`response`	The response object
`request`	The request object

The `view.configure_format` event is notified by the view when the request has the `sf_format` parameter set. The event is notified after symfony has done simple things like changing setting or unsetting the layout. This event allows the view and the response object to be changed according to the requested format.

view.method_not_found

Notify method: `notifyUntil`

Default notifiers: `sfView`

Parameter	Description
`method`	The name of the called missing method
`arguments`	The arguments passed to the method

The `view.method_not_found` event is notified when a method is not defined in the `sfView` class. By listening to this event, a method can be added to the class, without using inheritance.

view.cache

view.cache.filter_content

Notify method: `filter`

Default notifiers: `sfViewCacheManager`

Parameter	Description
`response`	The response object
`uri`	The URI of the cached content
`new`	Whether the content is new in cache or not

The `view.cache.filter_content` event is notified whenever a content is retrieved from the cache.

Tasks

The symfony framework comes bundled with a command line interface tool. Built-in tasks allow the developer to perform a lot of fastidious and recurrent tasks in the life of a project.

If you execute the `symfony` CLI without any arguments, a list of available tasks is displayed:

Listing 16-1
```
$ php symfony
```

By passing the `-V` option, you get some information about the version of symfony and the path of the symfony libraries used by the CLI:

Listing 16-2
```
$ php symfony -V
```

The CLI tool takes a task name as its first argument:

Listing 16-3
```
$ php symfony list
```

A task name can be composed of an optional namespace and a name, separated by a colon (`:`):

Listing 16-4
```
$ php symfony cache:clear
```

After the task name, arguments and options can be passed:

Listing 16-5
```
$ php symfony cache:clear --type=template
```

The CLI tool supports both long options and short ones, with or without values.

The `-t` option is a global option to ask any task to output more debugging information.

Available Tasks

help

The `help` task displays help for a task:

```
$ php symfony help  [task_name]
```

Listing 16-6

Alias(es): `h`

Argument	Default	Description
`task_name`	`help`	The task name

list

The `list` task lists tasks:

```
$ php symfony list  [namespace]
```

Listing 16-7

Argument	Default	Description
`namespace`	`-`	The namespace name

The `list` task lists all tasks:

```
./symfony list
```

Listing 16-8

You can also display the tasks for a specific namespace:

```
./symfony list test
```

Listing 16-9

app

app::routes

The `app::routes` task displays current routes for an application:

```
$ php symfony app:routes  application [name]
```

Listing 16-10

Argument	Default	Description
`application`	`-`	The application name
`name`	`-`	A route name

The `app:routes` displays the current routes for a given application:

```
./symfony app:routes frontend
```

Listing 16-11

cache

cache::clear

The `cache::clear` task clears the cache:

Listing 16-12
```
$ php symfony cache:clear [--app["..."]] [--env["..."]] [--type["..."]]
```

Alias(es): `cc, clear-cache`

Option (Shortcut)	Default	Description
`--app`	-	The application name
`--env`	-	The environment
`--type`	`all`	The type

The `cache:clear` task clears the symfony cache.

By default, it removes the cache for all available types, all applications, and all environments.

You can restrict by type, application, or environment:

For example, to clear the `frontend` application cache:

Listing 16-13
```
./symfony cache:clear --app=frontend
```

To clear the cache for the `prod` environment for the `frontend` application:

Listing 16-14
```
./symfony cache:clear --app=frontend --env=prod
```

To clear the cache for all `prod` environments:

Listing 16-15
```
./symfony cache:clear --env=prod
```

To clear the `config` cache for all `prod` environments:

Listing 16-16
```
./symfony cache:clear --type=config --env=prod
```

The built-in types are: `config`, `i18n`, `routing`, `module` and `template`.

configure

configure::author

The `configure::author` task configure project author:

Listing 16-17
```
$ php symfony configure:author  author
```

Argument	Default	Description
`author`	-	The project author

The `configure:author` task configures the author for a project:

Listing 16-18
```
./symfony configure:author "Fabien Potencier
<fabien.potencier@symfony-project.com>"
```

The author is used by the generates to pre-configure the PHPDoc header for each generated file.

The value is stored in [config/properties.ini].

configure::database

The configure::database task configure database DSN:

```
$ php symfony configure:database [--env["..."]] [--name["..."]]
[--class["..."]] [--app["..."]] dsn [username] [password]
```

Listing 16-19

Argument	Default	Description
dsn	-	The database dsn
username	root	The database username
password	-	The database password

Option (Shortcut)	Default	Description
--env	all	The environment
--name	propel	The connection name
--class	sfPropelDatabase	The database class name
--app	-	The application name

The configure:database task configures the database DSN for a project:

```
./symfony configure:database mysql:host=localhost;dbname=example root
mYsEcret
```

Listing 16-20

By default, the task change the configuration for all environment. If you want to change the dsn for a specific environment, use the env option:

```
./symfony configure:database --env=dev
mysql:host=localhost;dbname=example_dev root mYsEcret
```

Listing 16-21

To change the configuration for a specific application, use the app option:

```
./symfony configure:database --app=frontend
mysql:host=localhost;dbname=example root mYsEcret
```

Listing 16-22

You can also specify the connection name and the database class name:

```
./symfony configure:database --name=main --class=sfDoctrineDatabase
mysql:host=localhost;dbname=example root mYsEcret
```

Listing 16-23

WARNING: The propel.ini file is also updated when you use a Propel database and configure for all environments with no app.

doctrine

doctrine::build-all

The doctrine::build-all task generates Doctrine model, SQL and initializes the database:

```
$ php symfony doctrine:build-all [--application["..."]] [--env="..."]
[--no-confirmation] [--skip-forms|-F]
```

Listing 16-24

Alias(es): `doctrine-build-all`

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`dev`	The environment
`--no-confirmation`	`-`	Do not ask for confirmation
`--skip-forms (-F)`	`-`	Skip generating forms

The `doctrine:build-all` task is a shortcut for three other tasks:

Listing 16-25
```
./symfony doctrine:build-all
```

The task is equivalent to:

Listing 16-26
```
./symfony doctrine:build-model
./symfony doctrine:build-sql
./symfony doctrine:build-forms
./symfony doctrine:insert-sql
```

See those three tasks help page for more information.

To bypass the confirmation, you can pass the `no-confirmation` option:

Listing 16-27
```
./symfony doctrine:buil-all-load --no-confirmation
```

`doctrine::build-all-load`

The `doctrine::build-all-load` task generates Doctrine model, SQL, initializes database, and loads fixtures data:

Listing 16-28
```
$ php symfony doctrine:build-all-load [--application["..."]]
[--env="..."] [--connection="..."] [--no-confirmation] [--skip-forms|-F]
[--dir="..."]
```

Alias(es): `doctrine-build-all-load`

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`dev`	The environment
`--connection`	`doctrine`	The connection name
`--no-confirmation`	`-`	Do not ask for confirmation
`--skip-forms (-F)`	`-`	Skip generating forms
`--dir`	`-`	The directories to look for fixtures (multiple values allowed)

The `doctrine:build-all-load` task is a shortcut for two other tasks:

Listing 16-29
```
./symfony doctrine:build-all-load
```

The task is equivalent to:

```
./symfony doctrine:build-all
./symfony doctrine:data-load
```

Listing 16-30

The task takes an application argument because of the `doctrine:data-load` task. See `doctrine:data-load` help page for more information.

To bypass the confirmation, you can pass the `no-confirmation` option:

```
./symfony doctrine:build-all-load --no-confirmation
```

Listing 16-31

`doctrine::build-all-reload`

The `doctrine::build-all-reload` task generates Doctrine model, SQL, initializes database, and load data:

```
$ php symfony doctrine:build-all-reload [--application[="..."]]
[--env="..."] [--connection="..."] [--no-confirmation] [--skip-forms|-F]
[--dir="..."]
```

Listing 16-32

Alias(es): `doctrine-build-all-reload`

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`dev`	The environment
`--connection`	`doctrine`	The connection name
`--no-confirmation`	-	Do not ask for confirmation
`--skip-forms (-F)`	-	Skip generating forms
`--dir`	-	The directories to look for fixtures (multiple values allowed)

The `doctrine:build-all-reload` task is a shortcut for five other tasks:

```
./symfony doctrine:build-all-reload
```

Listing 16-33

The task is equivalent to:

```
./symfony doctrine:drop-db
./symfony doctrine:build-db
./symfony doctrine:build-model
./symfony doctrine:insert-sql
./symfony doctrine:data-load
```

Listing 16-34

`doctrine::build-all-reload-test-all`

The `doctrine::build-all-reload-test-all` task generates Doctrine model, SQL, initializes database, load data and run all tests:

```
$ php symfony doctrine:build-all-reload-test-all [--application[="..."]]
[--env="..."] [--append] [--dir="..."] [--force]
```

Listing 16-35

Alias(es): `doctrine-build-all-reload-test-all`

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`dev`	The environment
`--append`	-	Don't delete current data in the database
`--dir`	-	The directories to look for fixtures (multiple values allowed)
`--force`	-	Whether to force dropping of the database

The `doctrine:build-all-reload` task is a shortcut for four other tasks:

Listing 16-36
```
./symfony doctrine:build-all-reload-test-all frontend
```

The task is equivalent to:

Listing 16-37
```
./symfony doctrine:drop-db
./symfony doctrine:build-db
./symfony doctrine:build-model
./symfony doctrine:insert-sql
./symfony doctrine:data-load
./symfony test-all
```

The task takes an application argument because of the `doctrine:data-load` task. See `doctrine:data-load` help page for more information.

`doctrine::build-db`

The `doctrine::build-db` task creates database for current model:

Listing 16-38
```
$ php symfony doctrine:build-db [--application[="..."]] [--env="..."]
```

Alias(es): `doctrine-build-db`

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`dev`	The environment

The `doctrine:build-db` task creates the database:

Listing 16-39
```
./symfony doctrine:build-db
```

The task read connection information in `config/doctrine/databases.yml`:

`doctrine::build-filters`

The `doctrine::build-filters` task creates filter form classes for the current model:

Listing 16-40
```
$ php symfony doctrine:build-filters [--connection="..."]
[--model-dir-name="..."] [--filter-dir-name="..."] [--application[="..."]]
[--env="..."]
```

Option (Shortcut)	Default	Description
`--connection`	`doctrine`	The connection name
`--model-dir-name`	`model`	The model dir name

Option (Shortcut)	Default	Description
`--filter-dir-name`	`filter`	The filter form dir name
`--application`	`1`	The application name
`--env`	`dev`	The environment

The `doctrine:build-filters` task creates filter form classes from the schema:

```
./symfony doctrine:build-filters
```

Listing 16-41

The task read the schema information in `config/*schema.xml` and/or `config/*schema.yml` from the project and all installed plugins.

The task use the `doctrine` connection as defined in `config/databases.yml`. You can use another connection by using the `--connection` option:

```
./symfony doctrine:build-filters --connection="name"
```

Listing 16-42

The model filter form classes files are created in `lib/filter`.

This task never overrides custom classes in `lib/filter`. It only replaces base classes generated in `lib/filter/base`.

`doctrine::build-forms`

The `doctrine::build-forms` task creates form classes for the current model:

```
$ php symfony doctrine:build-forms [--connection="..."]
[--model-dir-name="..."] [--form-dir-name="..."] [--application[="..."]]
[--env="..."]
```

Listing 16-43

Option (Shortcut)	Default	Description
`--connection`	`doctrine`	The connection name
`--model-dir-name`	`model`	The model dir name
`--form-dir-name`	`form`	The form dir name
`--application`	`1`	The application name
`--env`	`dev`	The environment

The `doctrine:build-forms` task creates form classes from the schema:

```
./symfony doctrine:build-forms
```

Listing 16-44

The task read the schema information in `config/*schema.xml` and/or `config/*schema.yml` from the project and all installed plugins.

The task use the `doctrine` connection as defined in `config/databases.yml`. You can use another connection by using the `--connection` option:

```
./symfony doctrine:build-forms --connection="name"
```

Listing 16-45

The model form classes files are created in `lib/form`.

This task never overrides custom classes in `lib/form`. It only replaces base classes generated in `lib/form/base`.

doctrine::build-model

The `doctrine::build-model` task creates classes for the current model:

Listing 16-46
```
$ php symfony doctrine:build-model [--application[="..."]] [--env="..."]
```

Alias(es): `doctrine-build-model`

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`dev`	The environment

The `doctrine:build-model` task creates model classes from the schema:

Listing 16-47
```
./symfony doctrine:build-model
```

The task read the schema information in `config/doctrine/*.yml` from the project and all installed plugins.

The model classes files are created in `lib/model/doctrine`.

This task never overrides custom classes in `lib/model/doctrine`. It only replaces files in `lib/model/doctrine/base`.

doctrine::build-schema

The `doctrine::build-schema` task creates a schema from an existing database:

Listing 16-48
```
$ php symfony doctrine:build-schema [--application[="..."]] [--env="..."]
```

Alias(es): `doctrine-build-schema`

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`dev`	The environment

The `doctrine:build-schema` task introspects a database to create a schema:

Listing 16-49
```
./symfony doctrine:build-schema
```

The task creates a yml file in `config/doctrine`

doctrine::build-sql

The `doctrine::build-sql` task creates SQL for the current model:

Listing 16-50
```
$ php symfony doctrine:build-sql [--application[="..."]] [--env="..."]
```

Alias(es): `doctrine-build-sql`

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`dev`	The environment

The `doctrine:build-sql` task creates SQL statements for table creation:

```
./symfony doctrine:build-sql
```

Listing 16-51

The generated SQL is optimized for the database configured in `config/databases.yml`:

```
doctrine.database = mysql
```

Listing 16-52

doctrine::data-dump

The `doctrine::data-dump` task dumps data to the fixtures directory:

```
$ php symfony doctrine:data-dump [--application[="..."]] [--env="..."]
[target]
```

Listing 16-53

Alias(es): `doctrine-dump-data`

Argument	Default	Description
`target`	`-`	The target filename

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`dev`	The environment

The `doctrine:data-dump` task dumps database data:

```
./symfony doctrine:data-dump
```

Listing 16-54

The task dumps the database data in `data/fixtures/%target%`.

The dump file is in the YML format and can be reimported by using the `doctrine:data-load` task.

```
./symfony doctrine:data-load frontend
```

Listing 16-55

doctrine::data-load

The `doctrine::data-load` task loads data from fixtures directory:

```
$ php symfony doctrine:data-load [--application[="..."]] [--env="..."]
[--append] [--connection="..."] [--dir="..."]
```

Listing 16-56

Alias(es): `doctrine-load-data`

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`dev`	The environment
`--append`	`-`	Don't delete current data in the database
`--connection`	`doctrine`	The connection name
`--dir`	`-`	The directories to look for fixtures (multiple values allowed)

The `doctrine:data-load` task loads data fixtures into the database:

```
./symfony doctrine:data-load frontend
```

Listing 16-57

The task loads data from all the files found in `data/fixtures/`.

If you want to load data from other directories, you can use the `--dir` option:

Listing 16-58
```
./symfony doctrine:data-load --dir="data/fixtures" --dir="data/data"
frontend
```

If you don't want the task to remove existing data in the database, use the `--append` option:

Listing 16-59
```
./symfony doctrine:data-load --append frontend
```

doctrine::dql

The `doctrine::dql` task execute a DQL query and view the results:

Listing 16-60
```
$ php symfony doctrine:dql [--application[="..."]] [--env="..."]
[--show-sql] dql_query
```

Alias(es): `doctrine-dql`

Argument	Default	Description
`dql_query`	-	The DQL query to execute

Option (Shortcut)	Default	Description
`--application`	1	The application name
`--env`	dev	The environment
`--show-sql`	-	Show the sql that would be executed

The `doctrine:data-dql` task executes a DQL query and display the formatted results:

Listing 16-61
```
./symfony doctrine:dql "FROM User u"
```

You can show the SQL that would be executed by using the `--dir` option:

Listing 16-62
```
./symfony doctrine:dql --show-sql "FROM User u"
```

doctrine::drop-db

The `doctrine::drop-db` task drops database for current model:

Listing 16-63
```
$ php symfony doctrine:drop-db [--application[="..."]] [--env="..."]
[--no-confirmation]
```

Alias(es): `doctrine-drop-db`

Option (Shortcut)	Default	Description
`--application`	1	The application name
`--env`	dev	The environment
`--no-confirmation`	-	Whether to force dropping of the database

The `doctrine:drop-db` task drops the database:

Listing 16-64
```
./symfony doctrine:drop-db
```

The task read connection information in `config/doctrine/databases.yml`:

`doctrine::generate-admin`

The `doctrine::generate-admin` task generates a Doctrine admin module:

```
$ php symfony doctrine:generate-admin [--module="..."] [--theme="..."]
[--singular="..."] [--plural="..."] [--env="..."] application
route_or_model
```

Listing 16-65

Argument	Default	Description
`application`	-	The application name
`route_or_model`	-	The route name or the model class

Option (Shortcut)	Default	Description
`--module`	-	The module name
`--theme`	`admin`	The theme name
`--singular`	-	The singular name
`--plural`	-	The plural name
`--env`	`dev`	The environment

The `doctrine:generate-admin` task generates a Doctrine admin module:

```
./symfony doctrine:generate-admin frontend Article
```

Listing 16-66

The task creates a module in the `%frontend%` application for the `%Article%` model.

The task creates a route for you in the application `routing.yml`.

You can also generate a Doctrine admin module by passing a route name:

```
./symfony doctrine:generate-admin frontend article
```

Listing 16-67

The task creates a module in the `%frontend%` application for the `%article%` route definition found in `routing.yml`.

For the filters and batch actions to work properly, you need to add the `wildcard` option to the route:

```
article:
  class: sfDoctrineRouteCollection
  options:
    model:              Article
    with_wildcard_routes:   true
```

Listing 16-68

`doctrine::generate-migration`

The `doctrine::generate-migration` task generate migration class:

```
$ php symfony doctrine:generate-migration [--application[="..."]]
[--env="..."] name
```

Listing 16-69

Alias(es): `doctrine-generate-migration`

Argument	Default	Description
name	-	The name of the migration

Option (Shortcut)	Default	Description
--application	1	The application name
--env	dev	The environment

The doctrine:generate-migration task generates migration template

Listing 16-70
```
./symfony doctrine:generate-migration
```

doctrine::generate-migrations-db

The doctrine::generate-migrations-db task generate migration classes from existing database connections:

Listing 16-71
```
$ php symfony doctrine:generate-migrations-db [--application[="..."]]
[--env="..."]
```

Alias(es): doctrine-generate-migrations-db, doctrine-gen-migrations-from-db

Option (Shortcut)	Default	Description
--application	1	The application name
--env	dev	The environment

The doctrine:generate-migration task generates migration classes from existing database connections

Listing 16-72
```
./symfony doctrine:generate-migration
```

doctrine::generate-migrations-models

The doctrine::generate-migrations-models task generate migration classes from an existing set of models:

Listing 16-73
```
$ php symfony doctrine:generate-migrations-models [--application[="..."]]
[--env="..."]
```

Alias(es): doctrine-generate-migrations-models, doctrine-gen-migrations-from-models

Option (Shortcut)	Default	Description
--application	1	The application name
--env	dev	The environment

The doctrine:generate-migration task generates migration classes from an existing set of models

Listing 16-74
```
./symfony doctrine:generate-migration
```

doctrine::generate-module

The doctrine::generate-module task generates a Doctrine module:

```
$ php symfony doctrine:generate-module [--theme="..."]
[--generate-in-cache] [--non-verbose-templates] [--with-show]
[--singular="..."] [--plural="..."] [--route-prefix="..."]
[--with-doctrine-route] [--env="..."] application module model
```

Listing 16-75

Alias(es): doctrine-generate-crud, doctrine:generate-crud

Argument	Default	Description
application	-	The application name
module	-	The module name
model	-	The model class name

Option (Shortcut)	Default	Description
--theme	default	The theme name
--generate-in-cache	-	Generate the module in cache
--non-verbose-templates	-	Generate non verbose templates
--with-show	-	Generate a show method
--singular	-	The singular name
--plural	-	The plural name
--route-prefix	-	The route prefix
--with-doctrine-route	-	Whether you will use a Doctrine route
--env	dev	The environment

The doctrine:generate-module task generates a Doctrine module:

```
./symfony doctrine:generate-module frontend article Article
```

Listing 16-76

The task creates a %module% module in the %application% application for the model class %model%.

You can also create an empty module that inherits its actions and templates from a runtime generated module in %sf_app_cache_dir%/modules/auto%module% by using the --generate-in-cache option:

```
./symfony doctrine:generate-module --generate-in-cache frontend article
Article
```

Listing 16-77

The generator can use a customized theme by using the --theme option:

```
./symfony doctrine:generate-module --theme="custom" frontend article
Article
```

Listing 16-78

This way, you can create your very own module generator with your own conventions.

doctrine::generate-module-for-route

The doctrine::generate-module-for-route task generates a Doctrine module for a route definition:

Listing 16-79
```
$ php symfony doctrine:generate-module-for-route [--theme="..."]
[--non-verbose-templates] [--singular="..."] [--plural="..."]
[--env="..."] application route
```

Argument	Default	Description
application	-	The application name
route	-	The route name

Option (Shortcut)	Default	Description
--theme	default	The theme name
--non-verbose-templates	-	Generate non verbose templates
--singular	-	The singular name
--plural	-	The plural name
--env	dev	The environment

The `doctrine:generate-module-for-route` task generates a Doctrine module for a route definition:

Listing 16-80
```
./symfony doctrine:generate-module-for-route frontend article
```

The task creates a module in the `%frontend%` application for the `%article%` route definition found in `routing.yml`.

doctrine::insert-sql

The `doctrine::insert-sql` task inserts SQL for current model:

Listing 16-81
```
$ php symfony doctrine:insert-sql [--application[="..."]] [--env="..."]
```

Alias(es): `doctrine-insert-sql`

Option (Shortcut)	Default	Description
--application	1	The application name
--env	dev	The environment

The `doctrine:insert-sql` task creates database tables:

Listing 16-82
```
./symfony doctrine:insert-sql
```

The task connects to the database and creates tables for all the `lib/model/doctrine/*.php` files.

doctrine::migrate

The `doctrine::migrate` task migrates database to current/specified version:

Listing 16-83
```
$ php symfony doctrine:migrate [--application[="..."]] [--env="..."]
[version]
```

Alias(es): `doctrine-migrate`

Argument	Default	Description
version	-	The version to migrate to

Option (Shortcut)	Default	Description
--application	1	The application name
--env	dev	The environment

The doctrine:migrate task migrates database to current/specified version

```
./symfony doctrine:migrate
```

Listing 16-84

doctrine::rebuild-db

The doctrine::rebuild-db task creates database for current model:

```
$ php symfony doctrine:rebuild-db [--application[="..."]] [--env="..."]
[--no-confirmation]
```

Listing 16-85

Alias(es): doctrine-rebuild-db

Option (Shortcut)	Default	Description
--application	1	The application name
--env	dev	The environment
--no-confirmation	-	Whether to no-confirmation dropping of the database

The doctrine:rebuild-db task creates the database:

```
./symfony doctrine:rebuild-db
```

Listing 16-86

The task read connection information in config/doctrine/databases.yml:

generate

generate::app

The generate::app task generates a new application:

```
$ php symfony generate:app [--escaping-strategy="..."]
[--csrf-secret="..."] application
```

Listing 16-87

Alias(es): init-app

Argument	Default	Description
application	-	The application name

Option (Shortcut)	Default	Description
--escaping-strategy	“	Output escaping strategy
--csrf-secret	“	Secret to use for CSRF protection

The `generate:app` task creates the basic directory structure for a new application in the current project:

Listing 16-88
```
./symfony generate:app frontend
```

This task also creates two front controller scripts in the `web/` directory:

Listing 16-89
```
web/%application%.php`      for the production environment
web/%application%_dev.php` for the development environment
```

For the first application, the production environment script is named `index.php`.

If an application with the same name already exists, it throws a `sfCommandException`.

You can enable output escaping (to prevent XSS) by using the `escaping-strategy` option:

Listing 16-90
```
./symfony generate:app frontend --escaping-strategy=on
```

You can enable session token in forms (to prevent CSRF) by defining a secret with the `csrf-secret` option:

Listing 16-91
```
./symfony generate:app frontend --csrf-secret=UniqueSecret
```

`generate::module`

The `generate::module` task generates a new module:

Listing 16-92
```
$ php symfony generate:module  application module
```

Alias(es): `init-module`

Argument	Default	Description
`application`	-	The application name
`module`	-	The module name

The `generate:module` task creates the basic directory structure for a new module in an existing application:

Listing 16-93
```
./symfony generate:module frontend article
```

The task can also change the author name found in the `actions.class.php` if you have configure it in `config/properties.ini`:

Listing 16-94
```
  name=blog
  author=Fabien Potencier <fabien.potencier@sensio.com>
```

You can customize the default skeleton used by the task by creating a `%sf_data_dir%/skeleton/module` directory.

The task also creates a functional test stub named `%sf_test_dir%/functional/%application%/%module%ActionsTest.class.php` that does not pass by default.

If a module with the same name already exists in the application, it throws a `sfCommandException`.

`generate::project`

The `generate::project` task generates a new project:

```
$ php symfony generate:project  name
```

Listing 16-95

Alias(es): `init-project`

Argument	Default	Description
`name`	-	The project name

The `generate:project` task creates the basic directory structure for a new project in the current directory:

```
./symfony generate:project blog
```

Listing 16-96

If the current directory already contains a symfony project, it throws a `sfCommandException`.

`generate::task`

The `generate::task` task creates a skeleton class for a new task:

```
$ php symfony generate:task [--dir="..."] [--use-database="..."]
[--brief-description="..."] task_name
```

Listing 16-97

Argument	Default	Description
`task_name`	-	The task name (can contain namespace)

Option (Shortcut)	Default	Description
`--dir`	`lib/task`	The directory to create the task in
`--use-database`	`propel`	Whether the task needs model initialization to access database
`--brief-description`	-	A brief task description (appears in task list)

The `generate:task` creates a new sfTask class based on the name passed as argument:

```
./symfony generate:task namespace:name
```

Listing 16-98

The `namespaceNameTask.class.php` skeleton task is created under the `lib/task/` directory. Note that the namespace is optional.

If you want to create the file in another directory (relative to the project root folder), pass it in the `--dir` option. This directory will be created if it does not already exist.

```
./symfony generate:task namespace:name --dir=plugins/myPlugin/lib/task
```

Listing 16-99

If you want the task to default to a connection other than `propel`, provide the name of this connection with the `--use-database` option:

```
./symfony generate:task namespace:name --use-database=main
```

Listing 16-100

The `--use-database` option can also be used to disable database initialization in the generated task:

```
./symfony generate:task namespace:name --use-database=false
```

Listing 16-101

You can also specify a description:

Listing 16-102
```
./symfony generate:task namespace:name --brief-description="Does
interesting things"
```

i18n

i18n::extract

The `i18n::extract` task extracts i18n strings from php files:

Listing 16-103
```
$ php symfony i18n:extract [--display-new] [--display-old] [--auto-save]
[--auto-delete] application culture
```

Argument	Default	Description
`application`	-	The application name
`culture`	-	The target culture

Option (Shortcut)	Default	Description
`--display-new`	-	Output all new found strings
`--display-old`	-	Output all old strings
`--auto-save`	-	Save the new strings
`--auto-delete`	-	Delete old strings

The `i18n:extract` task extracts i18n strings from your project files for the given application and target culture:

Listing 16-104
```
./symfony i18n:extract frontend fr
```

By default, the task only displays the number of new and old strings it found in the current project.

If you want to display the new strings, use the `--display-new` option:

Listing 16-105
```
./symfony i18n:extract --display-new frontend fr
```

To save them in the i18n message catalogue, use the `--auto-save` option:

Listing 16-106
```
./symfony i18n:extract --auto-save frontend fr
```

If you want to display strings that are present in the i18n messages catalogue but are not found in the application, use the `--display-old` option:

Listing 16-107
```
./symfony i18n:extract --display-old frontend fr
```

To automatically delete old strings, use the `--auto-delete` but be careful, especially if you have translations for plugins as they will appear as old strings but they are not:

Listing 16-108
```
./symfony i18n:extract --auto-delete frontend fr
```

i18n::find

The `i18n::find` task finds non "i18n ready" strings in an application:

```
$ php symfony i18n:find [--env="..."] application
```

Listing 16-109

Argument	Default	Description
application	-	The application name

Option (Shortcut)	Default	Description
--env	dev	The environment

The `i18n:find` task finds non internationalized strings embedded in templates:

```
./symfony i18n:find frontend
```

Listing 16-110

This task is able to find non internationalized strings in pure HTML and in PHP code:

```
<p>Non i18n text</p>
<p><?php echo 'Test' ?></p>
```

Listing 16-111

As the task returns all strings embedded in PHP, you can have some false positive (especially if you use the string syntax for helper arguments).

log

log::clear

The `log::clear` task clears log files:

```
$ php symfony log:clear
```

Listing 16-112

Alias(es): `log-purge`

The `log:clear` task clears all symfony log files:

```
./symfony log:clear
```

Listing 16-113

log::rotate

The `log::rotate` task rotates an application log files:

```
$ php symfony log:rotate [--history="..."] [--period="..."] application env
```

Listing 16-114

Alias(es): `log-rotate`

Argument	Default	Description
application	-	The application name
env	-	The environment name

Option (Shortcut)	Default	Description
--history	10	The maximum number of old log files to keep

Option (Shortcut)	Default	Description
`--period`	7	The period in days

The `log:rotate` task rotates application log files for a given environment:

Listing 16-115
```
./symfony log:rotate frontend dev
```

You can specify a `period` or a `history` option:

Listing 16-116
```
./symfony --history=10 --period=7 log:rotate frontend dev
```

plugin

plugin::add-channel

The `plugin::add-channel` task add a new PEAR channel:

Listing 16-117
```
$ php symfony plugin:add-channel  name
```

Argument	Default	Description
`name`	-	The channel name

The `plugin:add-channel` task adds a new PEAR channel:

Listing 16-118
```
./symfony plugin:add-channel symfony.plugins.pear.example.com
```

plugin::install

The `plugin::install` task installs a plugin:

Listing 16-119
```
$ php symfony plugin:install [--stability|-s="..."] [--release|-r="..."]
[--channel|-c="..."] [--install_deps|-d] [--force-license] name
```

Alias(es): `plugin-install`

Argument	Default	Description
`name`	-	The plugin name

Option (Shortcut)	Default	Description
`--stability (-s)`	-	The preferred stability (stable, beta, alpha)
`--release (-r)`	-	The preferred version
`--channel (-c)`	-	The PEAR channel name
`--install_deps (-d)`	-	Whether to force installation of required dependencies
`--force-license`	-	Whether to force installation even if the license is not MIT like

The `plugin:install` task installs a plugin:

```
./symfony plugin:install sfGuardPlugin
```

Listing 16-120

By default, it installs the latest `stable` release.

If you want to install a plugin that is not stable yet, use the `stability` option:

```
./symfony plugin:install --stability=beta sfGuardPlugin
./symfony plugin:install -s beta sfGuardPlugin
```

Listing 16-121

You can also force the installation of a specific version:

```
./symfony plugin:install --release=1.0.0 sfGuardPlugin
./symfony plugin:install -r 1.0.0 sfGuardPlugin
```

Listing 16-122

To force installation of all required dependencies, use the `install_deps` flag:

```
./symfony plugin:install --install-deps sfGuardPlugin
./symfony plugin:install -d sfGuardPlugin
```

Listing 16-123

By default, the PEAR channel used is `symfony-plugins` (plugins.symfony-project.org).

You can specify another channel with the `channel` option:

```
./symfony plugin:install --channel=mypearchannel sfGuardPlugin
./symfony plugin:install -c mypearchannel sfGuardPlugin
```

Listing 16-124

You can also install PEAR packages hosted on a website:

```
./symfony plugin:install http://somewhere.example.com/
sfGuardPlugin-1.0.0.tgz
```

Listing 16-125

Or local PEAR packages:

```
./symfony plugin:install /home/fabien/plugins/sfGuardPlugin-1.0.0.tgz
```

Listing 16-126

If the plugin contains some web content (images, stylesheets or javascripts), the task creates a `%name%` symbolic link for those assets under `web/`. On Windows, the task copy all the files to the `web/%name%` directory.

`plugin::list`

The `plugin::list` task lists installed plugins:

```
$ php symfony plugin:list
```

Listing 16-127

Alias(es): `plugin-list`

The `plugin:list` task lists all installed plugins:

```
./symfony plugin:list
```

Listing 16-128

It also gives the channel and version for each plugin.

`plugin::publish-assets`

The `plugin::publish-assets` task publishes web assets for all plugins:

Listing 16-129

```
$ php symfony plugin:publish-assets [--core-only]
[--symfony-lib-dir="..."]
```

Option (Shortcut)	Default	Description
`--core-only`	-	If set only core plugins will publish their assets
`--symfony-lib-dir`	-	The symfony lib dir

The `plugin:publish-assets` task will publish web assets from all plugins.

Listing 16-130
```
./symfony plugin:publish-assets
```

In fact this will send the `plugin.post_install` event to each plugin.

`plugin::uninstall`

The `plugin::uninstall` task uninstalls a plugin:

Listing 16-131
```
$ php symfony plugin:uninstall [--channel|-c="..."] [--install_deps|-d]
name
```

Alias(es): `plugin-uninstall`

Argument	Default	Description
`name`	-	The plugin name

Option (Shortcut)	Default	Description
`--channel (-c)`	-	The PEAR channel name
`--install_deps (-d)`	-	Whether to force installation of dependencies

The `plugin:uninstall` task uninstalls a plugin:

Listing 16-132
```
./symfony plugin:uninstall sfGuardPlugin
```

The default channel is `symfony`.

You can also uninstall a plugin which has a different channel:

Listing 16-133
```
./symfony plugin:uninstall --channel=mypearchannel sfGuardPlugin

./symfony plugin:uninstall -c mypearchannel sfGuardPlugin
```

Or you can use the `channel/package` notation:

Listing 16-134
```
./symfony plugin:uninstall mypearchannel/sfGuardPlugin
```

You can get the PEAR channel name of a plugin by launching the `plugin:list] task.

If the plugin contains some web content (images, stylesheets or javascripts), the task also removes the [web/%name%` symbolic link (on *nix) or directory (on Windows).

`plugin::upgrade`

The `plugin::upgrade` task upgrades a plugin:

```
$ php symfony plugin:upgrade [--stability|-s="..."] [--release|-r="..."]
[--channel|-c="..."] name
```

Listing 16-135

Alias(es): `plugin-upgrade`

Argument	Default	Description
`name`	-	The plugin name

Option (Shortcut)	Default	Description
`--stability (-s)`	-	The preferred stability (stable, beta, alpha)
`--release (-r)`	-	The preferred version
`--channel (-c)`	-	The PEAR channel name

The `plugin:upgrade` task tries to upgrade a plugin:

```
./symfony plugin:upgrade sfGuardPlugin
```

Listing 16-136

The default channel is `symfony`.

If the plugin contains some web content (images, stylesheets or javascripts), the task also updates the `web/%name%` directory content on Windows.

See `plugin:install` for more information about the format of the plugin name and options.

project

project::clear-controllers

The `project::clear-controllers` task clears all non production environment controllers:

```
$ php symfony project:clear-controllers
```

Listing 16-137

Alias(es): `clear-controllers`

The `project:clear-controllers` task clears all non production environment controllers:

```
./symfony project:clear-controllers
```

Listing 16-138

You can use this task on a production server to remove all front controller scripts except the production ones.

If you have two applications named `frontend` and `backend`, you have four default controller scripts in `web/`:

```
index.php
frontend_dev.php
backend.php
backend_dev.php
```

Listing 16-139

After executing the `project:clear-controllers` task, two front controller scripts are left in `web/`:

Listing 16-140
```
index.php
backend.php
```

Those two controllers are safe because debug mode and the web debug toolbar are disabled.

`project::deploy`

The `project::deploy` task deploys a project to another server:

Listing 16-141
```
$ php symfony project:deploy [--go] [--rsync-dir="..."]
[--rsync-options["..."]] server
```

Alias(es): `sync`

Argument	Default	Description
`server`	`-`	The server name

Option (Shortcut)	Default	Description
`--go`	`-`	Do the deployment
`--rsync-dir`	`config`	The directory where to look for rsync*.txt files
`--rsync-options`	`-azC --force --delete`	To options to pass to the rsync executable

The `project:deploy` task deploys a project on a server:

Listing 16-142
```
./symfony project:deploy production
```

The server must be configured in `config/properties.ini`:

Listing 16-143
```
  host=www.example.com
  port=22
  user=fabien
  dir=/var/www/sfblog/
  type=rsync
```

To automate the deployment, the task uses rsync over SSH. You must configure SSH access with a key or configure the password in `config/properties.ini`.

By default, the task is in dry-mode. To do a real deployment, you must pass the `--go` option:

Listing 16-144
```
./symfony project:deploy --go production
```

Files and directories configured in `config/rsync_exclude.txt` are not deployed:

Listing 16-145
```
.svn
/web/uploads/*
/cache/*
/log/*
```

You can also create a `rsync.txt` and `rsync_include.txt` files.

If you need to customize the `rsync*.txt` files based on the server, you can pass a `rsync-dir` option:

```
./symfony project:deploy --go --rsync-dir=config/production production
```
Listing 16-146

Last, you can specify the options passed to the rsync executable, using the `rsync-options` option (defaults are `-azC`):

```
./symfony project:deploy --go --rsync-options=avz
```
Listing 16-147

`project::disable`

The `project::disable` task disables an application in a given environment:

```
$ php symfony project:disable  application env
```
Listing 16-148

Alias(es): `disable`

Argument	Default	Description
`application`	-	The application name
`env`	-	The environment name

The `project:disable` task disables an application for a specific environment:

```
./symfony project:disable frontend prod
```
Listing 16-149

`project::enable`

The `project::enable` task enables an application in a given environment:

```
$ php symfony project:enable  application env
```
Listing 16-150

Alias(es): `enable`

Argument	Default	Description
`application`	-	The application name
`env`	-	The environment name

The `project:enable` task enables an application for a specific environment:

```
./symfony project:enable frontend prod
```
Listing 16-151

`project::freeze`

The `project::freeze` task freezes symfony libraries:

```
$ php symfony project:freeze  symfony_data_dir
```
Listing 16-152

Alias(es): `freeze`

Argument	Default	Description
`symfony_data_dir`	-	The symfony data directory

The `project:freeze` task copies all the symfony core files to the current project:

```
./symfony project:freeze /path/to/symfony/data/directory
```
Listing 16-153

The task takes a mandatory argument of the path to the symfony data directory.

The task also changes `config/config.php` to switch to the embedded symfony files.

`project::permissions`

The `project::permissions` task fixes symfony directory permissions:

Listing 16-154
```
$ php symfony project:permissions
```

Alias(es): `permissions, fix-perms`

The `project:permissions` task fixes directory permissions:

Listing 16-155
```
./symfony project:permissions
```

`project::unfreeze`

The `project::unfreeze` task unfreezes symfony libraries:

Listing 16-156
```
$ php symfony project:unfreeze
```

Alias(es): `unfreeze`

The `project:unfreeze` task removes all the symfony core files from the current project:

Listing 16-157
```
./symfony project:unfreeze
```

The task also changes `config/config.php` to switch to the old symfony files used before the `project:freeze` command was used.

`project::upgrade1.1`

The `project::upgrade1.1` task upgrade a symfony project to the 1.1 symfony release:

Listing 16-158
```
$ php symfony project:upgrade1.1
```

The `project:upgrade1.1` task upgrades a symfony project based the 1.0 release to the 1.1 symfony release.

Listing 16-159
```
./symfony project:upgrade1.1
```

Please read the UPGRADE_TO_1_1 file to have information on what does this task.

`project::upgrade1.2`

The `project::upgrade1.2` task upgrade a symfony project to the 1.2 symfony release (from 1.1):

Listing 16-160
```
$ php symfony project:upgrade1.2
```

The `project:upgrade1.2` task upgrades a symfony project based on the 1.1 release to the 1.2 symfony release.

Listing 16-161
```
./symfony project:upgrade1.2
```

Please read the UPGRADE_TO_1_2 file to have information on what does this task.

propel

propel::build-all

The `propel::build-all` task generates Propel model and form classes, SQL and initializes the database:

```
$ php symfony propel:build-all [--application["..."]] [--env="..."]
[--connection="..."] [--no-confirmation] [--skip-forms|-F]
[--classes-only|-C] [--phing-arg="..."]
```

Listing 16-162

Alias(es): `propel-build-all`

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`dev`	The environment
`--connection`	`propel`	The connection name
`--no-confirmation`	`-`	Do not ask for confirmation
`--skip-forms (-F)`	`-`	Skip generating forms
`--classes-only (-C)`	`-`	Do not initialize the database
`--phing-arg`	`-`	Arbitrary phing argument (multiple values allowed)

The `propel:build-all` task is a shortcut for five other tasks:

```
./symfony propel:build-all
```

Listing 16-163

The task is equivalent to:

```
./symfony propel:build-model
./symfony propel:build-forms
./symfony propel:build-filters
./symfony propel:build-sql
./symfony propel:insert-sql
```

Listing 16-164

See those tasks' help pages for more information.

To bypass confirmation prompts, you can pass the `no-confirmation` option:

```
./symfony propel:buil-all --no-confirmation
```

Listing 16-165

To build all classes but skip initializing the database, use the `classes-only` option:

```
./symfony propel:build-all --classes-only
```

Listing 16-166

propel::build-all-load

The `propel::build-all-load` task generates Propel model and form classes, SQL, initializes the database, and loads data:

Listing 16-167

```
$ php symfony propel:build-all-load [--application[="..."]] [--env="..."]
[--connection="..."] [--no-confirmation] [--skip-forms|-F]
[--classes-only|-C] [--phing-arg="..."] [--append] [--dir="..."]
```

Alias(es): `propel-build-all-load`

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`dev`	The environment
`--connection`	`propel`	The connection name
`--no-confirmation`	-	Do not ask for confirmation
`--skip-forms (-F)`	-	Skip generating forms
`--classes-only (-C)`	-	Do not initialize the database
`--phing-arg`	-	Arbitrary phing argument (multiple values allowed)
`--append`	-	Don't delete current data in the database
`--dir`	-	The directories to look for fixtures (multiple values allowed)

The `propel:build-all-load` task is a shortcut for two other tasks:

Listing 16-168
```
./symfony propel:build-all-load
```

The task is equivalent to:

Listing 16-169
```
./symfony propel:build-all
./symfony propel:data-load
```

See those tasks' help pages for more information.

To bypass the confirmation, you can pass the `no-confirmation` option:

Listing 16-170
```
./symfony propel:buil-all-load --no-confirmation
```

`propel::build-filters`

The `propel::build-filters` task creates filter form classes for the current model:

Listing 16-171
```
$ php symfony propel:build-filters [--connection="..."]
[--model-dir-name="..."] [--filter-dir-name="..."] [--application[="..."]]
```

Option (Shortcut)	Default	Description
`--connection`	`propel`	The connection name
`--model-dir-name`	`model`	The model dir name
`--filter-dir-name`	`filter`	The filter form dir name
`--application`	`1`	The application name

The `propel:build-filters` task creates filter form classes from the schema:

```
./symfony propel:build-filters
```

Listing 16-172

The task read the schema information in `config/*schema.xml` and/or `config/*schema.yml` from the project and all installed plugins.

The task use the `propel` connection as defined in `config/databases.yml`. You can use another connection by using the `--connection` option:

```
./symfony propel:build-filters --connection="name"
```

Listing 16-173

The model filter form classes files are created in `lib/filter`.

This task never overrides custom classes in `lib/filter`. It only replaces base classes generated in `lib/filter/base`.

`propel::build-forms`

The `propel::build-forms` task creates form classes for the current model:

```
$ php symfony propel:build-forms [--connection="..."]
[--model-dir-name="..."] [--form-dir-name="..."] [--application[="..."]]
```

Listing 16-174

Option (Shortcut)	Default	Description
`--connection`	`propel`	The connection name
`--model-dir-name`	`model`	The model dir name
`--form-dir-name`	`form`	The form dir name
`--application`	`1`	The application name

The `propel:build-forms` task creates form classes from the schema:

```
./symfony propel:build-forms
```

Listing 16-175

The task read the schema information in `config/*schema.xml` and/or `config/*schema.yml` from the project and all installed plugins.

The task use the `propel` connection as defined in `config/databases.yml`. You can use another connection by using the `--connection` option:

```
./symfony propel:build-forms --connection="name"
```

Listing 16-176

The model form classes files are created in `lib/form`.

This task never overrides custom classes in `lib/form`. It only replaces base classes generated in `lib/form/base`.

`propel::build-model`

The `propel::build-model` task creates classes for the current model:

```
$ php symfony propel:build-model [--phing-arg="..."]
```

Listing 16-177

Alias(es): `propel-build-model`

Option (Shortcut)	Default	Description
`--phing-arg`	-	Arbitrary phing argument (multiple values allowed)

The `propel:build-model` task creates model classes from the schema:

Listing 16-178
```
./symfony propel:build-model
```

The task read the schema information in `config/*schema.xml` and/or `config/*schema.yml` from the project and all installed plugins.

You mix and match YML and XML schema files. The task will convert YML ones to XML before calling the Propel task.

The model classes files are created in `lib/model`.

This task never overrides custom classes in `lib/model`. It only replaces files in `lib/model/om` and `lib/model/map`.

`propel::build-schema`

The `propel::build-schema` task creates a schema from an existing database:

Listing 16-179
```
$ php symfony propel:build-schema [--application[="..."]] [--env="..."]
[--connection="..."] [--xml] [--phing-arg="..."]
```

Alias(es): `propel-build-schema`

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`cli`	The environment
`--connection`	-	The connection name
`--xml`	-	Creates an XML schema instead of a YML one
`--phing-arg`	-	Arbitrary phing argument (multiple values allowed)

The `propel:build-schema` task introspects a database to create a schema:

Listing 16-180
```
./symfony propel:build-schema
```

By default, the task creates a YML file, but you can also create a XML file:

Listing 16-181
```
./symfony --xml propel:build-schema
```

The XML format contains more information than the YML one.

`propel::build-sql`

The `propel::build-sql` task creates SQL for the current model:

Listing 16-182
```
$ php symfony propel:build-sql [--phing-arg="..."]
```

Alias(es): `propel-build-sql`

Option (Shortcut)	Default	Description
`--phing-arg`	-	Arbitrary phing argument (multiple values allowed)

The `propel:build-sql` task creates SQL statements for table creation:

Listing 16-183
```
./symfony propel:build-sql
```

The generated SQL is optimized for the database configured in `config/propel.ini`:

```
propel.database = mysql
```

Listing 16-184

`propel::data-dump`

The `propel::data-dump` task dumps data to the fixtures directory:

```
$ php symfony propel:data-dump [--application[="..."]] [--env="..."]
[--connection="..."] [--classes="..."] [target]
```

Listing 16-185

Alias(es): `propel-dump-data`

Argument	Default	Description
`target`	-	The target filename

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`cli`	The environement
`--connection`	`propel`	The connection name
`--classes`	-	The class names to dump (separated by a colon)

The `propel:data-dump` task dumps database data:

```
./symfony propel:data-dump > data/fixtures/dump.yml
```

Listing 16-186

By default, the task outputs the data to the standard output, but you can also pass a filename as a second argument:

```
./symfony propel:data-dump dump.yml
```

Listing 16-187

The task will dump data in `data/fixtures/%target%` (data/fixtures/dump.yml in the example).

The dump file is in the YML format and can be re-imported by using the `propel:data-load` task.

By default, the task use the `propel` connection as defined in `config/databases.yml`. You can use another connection by using the `connection` option:

```
./symfony propel:data-dump --connection="name"
```

Listing 16-188

If you only want to dump some classes, use the `classes` option:

```
./symfony propel:data-dump --classes="Article,Category"
```

Listing 16-189

If you want to use a specific database configuration from an application, you can use the `application` option:

```
./symfony propel:data-dump --application=frontend
```

Listing 16-190

`propel::data-load`

The `propel::data-load` task loads data from fixtures directory:

Listing 16-191
```
$ php symfony propel:data-load [--application["..."]] [--env="..."]
[--append] [--connection="..."] [--dir="..."]
```

Alias(es): `propel-load-data`

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`cli`	The environment
`--append`	`-`	Don't delete current data in the database
`--connection`	`propel`	The connection name
`--dir`	`-`	The directories to look for fixtures (multiple values allowed)

The `propel:data-load` task loads data fixtures into the database:

Listing 16-192
```
./symfony propel:data-load
```

The task loads data from all the files found in `data/fixtures/`.

If you want to load data from other directories, you can use the `--dir` option:

Listing 16-193
```
./symfony propel:data-load --dir="data/fixtures" --dir="data/data"
```

The task use the `propel` connection as defined in `config/databases.yml`. You can use another connection by using the `--connection` option:

Listing 16-194
```
./symfony propel:data-load --connection="name"
```

If you don't want the task to remove existing data in the database, use the `--append` option:

Listing 16-195
```
./symfony propel:data-load --append
```

If you want to use a specific database configuration from an application, you can use the `application` option:

Listing 16-196
```
./symfony propel:data-load --application=frontend
```

`propel::generate-admin`

The `propel::generate-admin` task generates a Propel admin module:

Listing 16-197
```
$ php symfony propel:generate-admin [--module="..."] [--theme="..."]
[--singular="..."] [--plural="..."] [--env="..."] application
route_or_model
```

Argument	Default	Description
`application`	`-`	The application name
`route_or_model`	`-`	The route name or the model class

Option (Shortcut)	Default	Description
`--module`	`-`	The module name
`--theme`	`admin`	The theme name
`--singular`	`-`	The singular name

Option (Shortcut)	Default	Description
`--plural`	-	The plural name
`--env`	dev	The environment

The `propel:generate-admin` task generates a Propel admin module:

```
./symfony propel:generate-admin frontend Article
```

Listing 16-198

The task creates a module in the `%frontend%` application for the `%Article%` model.

The task creates a route for you in the application `routing.yml`.

You can also generate a Propel admin module by passing a route name:

```
./symfony propel:generate-admin frontend article
```

Listing 16-199

The task creates a module in the `%frontend%` application for the `%article%` route definition found in `routing.yml`.

For the filters and batch actions to work properly, you need to add the `wildcard` option to the route:

```
article:
  class: sfPropelRouteCollection
  options:
    model:                Article
    with_wildcard_routes: true
```

Listing 16-200

`propel::generate-module`

The `propel::generate-module` task generates a Propel module:

```
$ php symfony propel:generate-module [--theme="..."] [--generate-in-cache]
[--non-verbose-templates] [--with-show] [--singular="..."]
[--plural="..."] [--route-prefix="..."] [--with-propel-route]
[--env="..."] application module model
```

Listing 16-201

Alias(es): `propel-generate-crud, propel:generate-crud`

Argument	Default	Description
`application`	-	The application name
`module`	-	The module name
`model`	-	The model class name

Option (Shortcut)	Default	Description
`--theme`	`default`	The theme name
`--generate-in-cache`	-	Generate the module in cache
`--non-verbose-templates`	-	Generate non verbose templates
`--with-show`	-	Generate a show method
`--singular`	-	The singular name
`--plural`	-	The plural name
`--route-prefix`	-	The route prefix
`--with-propel-route`	-	Whether you will use a Propel route

Option (Shortcut)	Default	Description
`--env`	`dev`	The environment

The `propel:generate-module` task generates a Propel module:

Listing 16-202
```
./symfony propel:generate-module frontend article Article
```

The task creates a `%module%` module in the `%application%` application for the model class `%model%`.

You can also create an empty module that inherits its actions and templates from a runtime generated module in `%sf_app_cache_dir%/modules/auto%module%` by using the `--generate-in-cache` option:

Listing 16-203
```
./symfony propel:generate-module --generate-in-cache frontend article
Article
```

The generator can use a customized theme by using the `--theme` option:

Listing 16-204
```
./symfony propel:generate-module --theme="custom" frontend article Article
```

This way, you can create your very own module generator with your own conventions.

`propel::generate-module-for-route`

The `propel::generate-module-for-route` task generates a Propel module for a route definition:

Listing 16-205
```
$ php symfony propel:generate-module-for-route [--theme="..."]
[--non-verbose-templates] [--singular="..."] [--plural="..."]
[--env="..."] application route
```

Argument	Default	Description
`application`	-	The application name
`route`	-	The route name

Option (Shortcut)	Default	Description
`--theme`	`default`	The theme name
`--non-verbose-templates`	-	Generate non verbose templates
`--singular`	-	The singular name
`--plural`	-	The plural name
`--env`	`dev`	The environment

The `propel:generate-module-for-route` task generates a Propel module for a route definition:

Listing 16-206
```
./symfony propel:generate-module-for-route frontend article
```

The task creates a module in the `%frontend%` application for the `%article%` route definition found in `routing.yml`.

propel::graphviz

The `propel::graphviz` task generates a graphviz chart of current object model:

```
$ php symfony propel:graphviz [--phing-arg="..."]
```

Listing 16-207

Option (Shortcut)	Default	Description
`--phing-arg`	-	Arbitrary phing argument (multiple values allowed)

The `propel:graphviz` task creates a graphviz DOT visualization for automatic graph drawing of object model:

```
./symfony propel:graphviz
```

Listing 16-208

propel::init-admin

The `propel::init-admin` task initializes a Propel admin module:

```
$ php symfony propel:init-admin [--theme="..."] application module model
```

Listing 16-209

Alias(es): `propel-init-admin`

Argument	Default	Description
`application`	-	The application name
`module`	-	The module name
`model`	-	The model class name

Option (Shortcut)	Default	Description
`--theme`	`default`	The theme name

The `propel:init-admin` task generates a Propel admin module:

```
./symfony propel:init-admin frontend article Article
```

Listing 16-210

The task creates a `%module%` module in the `%application%` application for the model class `%model%`.

The created module is an empty one that inherit its actions and templates from a runtime generated module in `%sf_app_cache_dir%/modules/auto%module%`.

The generator can use a customized theme by using the `--theme` option:

```
./symfony propel:init-admin --theme="custom" frontend article Article
```

Listing 16-211

propel::insert-sql

The `propel::insert-sql` task inserts SQL for current model:

```
$ php symfony propel:insert-sql [--application[="..."]] [--env="..."]
[--connection="..."] [--no-confirmation] [--phing-arg="..."]
```

Listing 16-212

Alias(es): `propel-insert-sql`

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`cli`	The environment
`--connection`	-	The connection name
`--no-confirmation`	-	Do not ask for confirmation
`--phing-arg`	-	Arbitrary phing argument (multiple values allowed)

The `propel:insert-sql` task creates database tables:

Listing 16-213
```
./symfony propel:insert-sql
```

The task connects to the database and executes all SQL statements found in `config/sql/*schema.sql` files.

Before execution, the task will ask you to confirm the execution as it deletes all data in your database.

To bypass the confirmation, you can pass the `--no-confirmation` option:

Listing 16-214
```
./symfony propel:insert-sql --no-confirmation
```

The task read the database configuration from `databases.yml`. You can use a specific application/environment by passing an `--application` or `--env` option.

You can also use the `--connection` option if you want to only load SQL statements for a given connection.

`propel::schema-to-xml`

The `propel::schema-to-xml` task creates schema.xml from schema.yml:

Listing 16-215
```
$ php symfony propel:schema-to-xml
```

Alias(es): `propel-convert-yml-schema`

The `propel:schema-to-xml` task converts YML schemas to XML:

Listing 16-216
```
./symfony propel:schema-to-xml
```

`propel::schema-to-yml`

The `propel::schema-to-yml` task creates schema.yml from schema.xml:

Listing 16-217
```
$ php symfony propel:schema-to-yml
```

Alias(es): `propel-convert-xml-schema`

The `propel:schema-to-yml` task converts XML schemas to YML:

Listing 16-218
```
./symfony propel:schema-to-yml
```

test

test::all

The `test::all` task launches all tests:

```
$ php symfony test:all
```

Listing 16-219

Alias(es): `test-all`

The `test:all` task launches all unit and functional tests:

```
./symfony test:all
```

Listing 16-220

The task launches all tests found in `test/`.

If one or more test fail, you can try to fix the problem by launching them by hand or with the `test:unit` and `test:functional` task.

test::coverage

The `test::coverage` task outputs test code coverage:

```
$ php symfony test:coverage [--detailed] test_name lib_name
```

Listing 16-221

Argument	Default	Description
`test_name`	-	A test file name or a test directory
`lib_name`	-	A lib file name or a lib directory for wich you want to know the coverage

Option (Shortcut)	Default	Description
`--detailed`	-	Output detailed information

The `test:coverage` task outputs the code coverage given a test file or test directory and a lib file or lib directory for which you want code coverage:

```
./symfony test:coverage test/unit/model lib/model
```

Listing 16-222

To output the lines not covered, pass the `--detailed` option:

```
./symfony test:coverage --detailed test/unit/model lib/model
```

Listing 16-223

test::functional

The `test::functional` task launches functional tests:

```
$ php symfony test:functional  application [controller1] ... [controllerN]
```

Listing 16-224

Alias(es): `test-functional`

Argument	Default	Description
`application`	-	The application name
`controller`	-	The controller name

The `test:functional` task launches functional tests for a given application:

Listing 16-225
```
./symfony test:functional frontend
```

The task launches all tests found in `test/functional/%application%`.
You can launch all functional tests for a specific controller by giving a controller name:

Listing 16-226
```
./symfony test:functional frontend article
```

You can also launch all functional tests for several controllers:

Listing 16-227
```
./symfony test:functional frontend article comment
```

`test::unit`

The `test::unit` task launches unit tests:

Listing 16-228
```
$ php symfony test:unit  [name1] ... [nameN]
```

Alias(es): `test-unit`

Argument	Default	Description
`name`	-	The test name

The `test:unit` task launches unit tests:

Listing 16-229
```
./symfony test:unit
```

The task launches all tests found in `test/unit`.
You can launch unit tests for a specific name:

Listing 16-230
```
./symfony test:unit strtolower
```

You can also launch unit tests for several names:

Listing 16-231
```
./symfony test:unit strtolower strtoupper
```

Appendices

Appendix A

License

Attribution-Share Alike 3.0 Unported License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

   a. **"Adaptation"** means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

   b. **"Collection"** means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined below) for the purposes of this License.

   c. **"Creative Commons Compatible License"** means a license that is listed at http://creativecommons.org/compatiblelicenses that has been approved by Creative Commons as being essentially equivalent to this License, including, at a minimum, because that license: (i) contains terms that have the same purpose, meaning and effect as the License Elements of this License; and, (ii) explicitly permits the relicensing of adaptations of works made available under that license under this

License or a Creative Commons jurisdiction license with the same License Elements as this License.

d. **"Distribute"** means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.

e. **"License Elements"** means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, ShareAlike.

f. **"Licensor"** means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

g. **"Original Author"** means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

h. **"Work"** means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

i. **"You"** means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

j. **"Publicly Perform"** means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

k. **"Reproduce"** means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights

   Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant

   Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

   a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;

   b. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";

   c. to Distribute and Publicly Perform the Work including as incorporated in Collections; and,

   d. to Distribute and Publicly Perform Adaptations.

   e. For the avoidance of doubt:

   i. **Non-waivable Compulsory License Schemes**. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;

   ii. **Waivable Compulsory License Schemes**. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor waives the exclusive right to collect such royalties for any exercise by You of the rights granted under this License; and,

   iii. **Voluntary License Schemes**. The Licensor waives the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License.

   The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved.

4. Restrictions

   The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

   a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested. If You create an Adaptation, upon notice from any Licensor You must, to

the extent practicable, remove from the Adaptation any credit as required by Section 4(c), as requested.

b. You may Distribute or Publicly Perform an Adaptation only under the terms of: (i) this License; (ii) a later version of this License with the same License Elements as this License; (iii) a Creative Commons jurisdiction license (either this or a later license version) that contains the same License Elements as this License (e.g., Attribution-ShareAlike 3.0 US)); (iv) a Creative Commons Compatible License. If you license the Adaptation under one of the licenses mentioned in (iv), you must comply with the terms of that license. If you license the Adaptation under the terms of any of the licenses mentioned in (i), (ii) or (iii) (the "Applicable License"), you must comply with the terms of the Applicable License generally and the following provisions: (I) You must include a copy of, or the URI for, the Applicable License with every copy of each Adaptation You Distribute or Publicly Perform; (II) You may not offer or impose any terms on the Adaptation that restrict the terms of the Applicable License or the ability of the recipient of the Adaptation to exercise the rights granted to that recipient under the terms of the Applicable License; (III) You must keep intact all notices that refer to the Applicable License and to the disclaimer of warranties with every copy of the Work as included in the Adaptation You Distribute or Publicly Perform; (IV) when You Distribute or Publicly Perform the Adaptation, You may not impose any effective technological measures on the Adaptation that restrict the ability of a recipient of the Adaptation from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section 4(b) applies to the Adaptation as incorporated in a Collection, but this does not require the Collection apart from the Adaptation itself to be made subject to the terms of the Applicable License.

c. If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and (iv) , consistent with Ssection 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

d. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work

which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

   UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability

   EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

   a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

   b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

   a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

   b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

   c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

Creative Commons Notice

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark “Creative Commons” or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons’ then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of the License.

Creative Commons may be contacted at http://creativecommons.org/.