The symfony Reference Book

symfony 1.3 & 1.4

This PDF is brought to you by

Version: reference-1.4-en-2010-11-09

Table of Contents

About the Author

Fabien Potencier discovered the Web in 1994, at a time when connecting to the Internet was still associated with the harmful strident sounds of a modem. Being a developer by passion, he immediately started to build websites with Perl. But with the release of PHP 5, he decided to switch focus to PHP, and created the symfony framework project in 2004 to help his company leverage the power of PHP for its customers.

Fabien is a serial-entrepreneur, and among other companies, he created Sensio, a services and consulting company specialized in web technologies and Internet marketing, in 1998.

Fabien is also the creator of several other Open-Source projects, a writer, a blogger, a speaker at international conferences, and a happy father of two wonderful kids.

His Website: *http://fabien.potencier.org/*

On Twitter: *http://www.twitter.com/fabpot*

About Sensio Labs

Sensio Labs is a services and consulting company specialized in Open-Source Web technologies and Internet marketing.

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 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 eleven 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.

Which symfony Version?

This book has been written for both symfony 1.3 and symfony 1.4. As writing a single book for two different versions of a software is quite unusual, this section explains what the main differences are between the two versions, and how to make the best choice for your projects.

Both the symfony 1.3 and symfony 1.4 versions have been released at about the same time (at the end of 2009). As a matter of fact, they both have the **exact same feature set**. The only difference between the two versions is how each supports backward compatibility with older symfony versions.

Symfony 1.3 is the release you'll want to use if you need to upgrade a legacy project that uses an older symfony version (1.0, 1.1, or 1.2). It has a backward compatibility layer and all the features that have been deprecated during the 1.3 development period are still available. It means that upgrading is easy, simple, and safe.

If you start a new project today, however, you should use symfony 1.4. This version has the same feature set as symfony 1.3 but all the deprecated features, including the entire compatibility layer, have been removed. This version is cleaner and also a bit faster than symfony 1.3. Another big advantage of using symfony 1.4 is its longer support. Being a Long Term Support release, it will be maintained by the symfony core team for three years (until November 2012).

Of course, you can migrate your projects to symfony 1.3 and then slowly update your code to remove the deprecated features and eventually move to symfony 1.4 in order to benefit from the long term support. You have plenty of time to plan the move as symfony 1.3 will be supported for a year (until November 2010).

As this book does not describe deprecated features, all examples work equally well on both versions.

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 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:

Listing 2-6
```
>
  This is a very long sentence
  that spans several lines in the YAML
  but which will be rendered as a string
  without carriage returns.
```

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

Numbers

Listing 2-7
```
# an integer
12
```

Listing 2-8
```
# an octal
014
```

Listing 2-9
```
# a hexadecimal
0xC
```

Listing 2-10
```
# a float
13.4
```

Listing 2-11
```
# an exponent
1.2e+34
```

Listing 2-12
```
# infinity
.inf
```

Nulls

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

Booleans

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

Dates

YAML uses the ISO-8601 standard to express dates:

Listing 2-13
```
2001-12-14t21:59:43.10-05:00
```

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:

```
1.0:
  version: <?php echo file_get_contents('1.0/VERSION')."\n" ?>
```

Listing 2-28

```
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_maintenance: 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_maintenance: 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 90)*.

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_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
SF_APP_MODULE_DIR	SF_APP_DIR/modules

Constants	Default Value
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 26)*, and benefits from the **configuration cascade mechanism** *(page 26)*.

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 90)*.

Settings

- `.actions`
  - `error_404` *(page 30)*
  - `login` *(page 30)*
  - `secure` *(page 30)*
  - `module_disabled` *(page 30)*
- `.settings`
  - `cache` *(page 32)*
  - `charset` *(page 31)*
  - `check_lock` *(page 34)*
  - `compressed` *(page 34)*
  - `csrf_secret` *(page 31)*
  - `default_culture` *(page 33)*
  - `default_timezone` *(page 32)*
  - `enabled_modules` *(page 31)*
  - `error_reporting` *(page 34)*
  - `escaping_strategy` *(page 30)*
  - `escaping_method` *(page 31)*
  - `etag` *(page 32)*
  - `i18n` *(page 32)*
  - `lazy_cache_key` *(page 33)*
  - `file_link_format` *(page 33)*
  - `logging_enabled` *(page 33)*
  - `no_script_name` *(page 33)*
  - `standard_helpers` *(page 33)*
  - `use_database` *(page 34)*
  - `web_debug` *(page 34)*
  - `web_debug_web_dir` *(page 35)*

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: `true`

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 leave the default value to `true`.

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: a randomly generated secret

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 of your choice.

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: `false`

The `cache` setting enables or disables template caching.

The general configuration of the cache system is done in the `view_cache_manager` *(page 45)* and `view_cache` *(page 46)* sections of the `factories.yml` configuration file. The fined-grained configuration is done in the `cache.yml` *(page 71)* configuration file.

`etag`

Default: `true` 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: `false`

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

The general configuration of the i18n system is to be done in the `i18n` *(page 46)* 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]

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: true for the prod environment of the first application created, false 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 true 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 true if all front controllers are in the same directory (web/). If you want more than one application with no_script_name set to true, move the corresponding front controller(s) under a sub-directory of the web root directory.

lazy_cache_key

Default: true for new projects, false 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.

file_link_format

Default: none

In the debug message, file paths are clickable links if the sf_file_link_format or if the xdebug.file_link_format PHP configuration value is set.

For example, if you want to open files in TextMate, you can use the following value:

```
txmt://open?url=file://%f&line=%l
```

Listing 4-4

The %f placeholder will be replaced with file's absolute path and the %l placeholder will be replaced with the line number.

logging_enabled

Default: true 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: `false` 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: `false`

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

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

use_database

Default: `true`

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

check_lock

Default: `false`

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

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

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

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

`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).

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 26)*, benefits from the **configuration cascade mechanism** *(page 26)*, and can include **constants** *(page 24)*.

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 90)*.

Factories

- `mailer` *(page 40)*
  - `charset` *(page 40)*
  - `delivery_address` *(page 40)*
  - `delivery_strategy` *(page 40)*
  - `spool_arguments` *(page 41)*
  - `spool_class` *(page 41)*
  - `transport` *(page 41)*
- `request` *(page 42)*
  - `formats` *(page 42)*
  - `path_info_array` *(page 42)*
  - `path_info_key` *(page 42)*
  - `relative_url_root` *(page 43)*
- `response` *(page 43)*
  - `charset` *(page 40)*
  - `http_protocol` *(page 43)*
  - `send_http_headers` *(page 43)*
- `user` *(page 43)*
  - `default_culture` *(page 44)*
  - `timeout` *(page 44)*
  - `use_flash` *(page 44)*
- `storage` *(page 44)*
  - `auto_start` *(page 45)*
  - `database` *(page 45)*
  - `db_table` *(page 45)*
  - `db_id_col` *(page 45)*
  - `db_data_col` *(page 45)*
  - `db_time_col` *(page 45)*
  - `session_cache_limiter` *(page 45)*
  - `session_cookie_domain` *(page 45)*
  - `session_cookie_httponly` *(page 45)*
  - `session_cookie_lifetime` *(page 45)*
  - `session_cookie_path` *(page 45)*
  - `session_cookie_secure` *(page 45)*
  - `session_name` *(page 45)*
- `view_cache_manager` *(page 45)*
  - `cache_key_use_vary_headers` *(page 46)*
  - `cache_key_use_host_name` *(page 46)*
- `view_cache` *(page 46)*
- `i18n` *(page 46)*
  - `cache` *(page 47)*
  - `debug` *(page 47)*
  - `source` *(page 47)*

mailer

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

Default configuration:

Listing 5-5
```
mailer:
  class: sfMailer
  param:
    logging:           %SF_LOGGING_ENABLED%
    charset:           %SF_CHARSET%
    delivery_strategy: realtime
    transport:
      class: Swift_SmtpTransport
      param:
        host:       localhost
        port:       25
        encryption: ~
        username:   ~
        password:   ~
```

Default configuration for the `test` *environment*:

Listing 5-6
```
mailer:
  param:
    delivery_strategy: none
```

Default configuration for the `dev` *environment*:

Listing 5-7
```
mailer:
  param:
    delivery_strategy: none
```

charset

The `charset` option defines the charset to use for the mail messages. By default, it uses the `charset` setting from `settings.yml`.

delivery_strategy

The `delivery_strategy` option defines how email messages are delivered by the mailer. Four strategies are available by default, which should suit all the common needs:

- `realtime`: Messages are sent in realtime.
- `single_address`: Messages are sent to a single address.
- `spool`: Messages are stored in a queue.
- `none`: Messages are simply ignored.

delivery_address

The `delivery_address` option defines the recipient of all message when the `delivery_strategy` is set to `single_address`.

spool_class

The `spool_class` option defines the spool class to use when the `delivery_strategy` is set to `spool`:

- `Swift_FileSpool`: Messages are stored on the filesystem.
- `Swift_DoctrineSpool`: Messages are stored in a Doctrine model.
- `Swift_PropelSpool`: Messages are stored in a Propel model.

When the spool is instantiated, the `spool_arguments` option is used as the constructor arguments.

spool_arguments

The `spool_arguments` option defines the constructor arguments of the spool. Here are the options available for the built-in queues classes:

- `Swift_FileSpool`:
  - The absolute path of the queue directory (messages are stored in this directory)
- `Swift_DoctrineSpool`:
  - The Doctrine model to use to store the messages (`MailMessage` by default)
  - The column name to use for message storage (`message` by default)
  - The method to call to retrieve the messages to send (optional).
- `Swift_PropelSpool`:
  - The Propel model to use to store the messages (`MailMessage` by default)
  - The column name to use for message storage (`message` by default)
  - The method to call to retrieve the messages to send (optional). It receives the current Criteria as an argument.

The configuration below shows a typical configuration for a Doctrine spool:

Listing 5-8

```
# configuration in factories.yml
mailer:
  class: sfMailer
  param:
    delivery_strategy: spool
    spool_class:       Swift_DoctrineSpool
    spool_arguments:   [ MailMessage, message, getSpooledMessages ]
```

transport

The `transport` option defines the transport to use to actually send email messages.

The `class` setting can be any class that implements from `Swift_Transport`, and three are provided by default:

- `Swift_SmtpTransport`: Uses a SMTP server to send messages.
- `Swift_SendmailTransport`: Uses `sendmail` to send messages.

- `Swift_MailTransport`: Uses the native PHP `mail()` function to send messages.
- `Swift_NullTransport`: Disables the transport altogether (useful with the `none` strategy to bypass the connection to the mail server).

You can further configure the transport by setting the `param` setting. The "Transport Types"[7] section of the Swift Mailer official documentation describes all you need to know about the built-in transport classes and their different parameters.

request

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

Default configuration:

Listing 5-9

```
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-Types`. It is used by the framework to automatically manage the `Content-Type` of the response, based on the request URI extension.

7. `http://swiftmailer.org/docs/transport-types`

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 5-10

```
response:
  class: sfWebResponse
  param:
    logging:           %SF_LOGGING_ENABLED%
    charset:           %SF_CHARSET%
    send_http_headers: true
```

Default configuration for the `test` *environment*:

Listing 5-11

```
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 5-12

```
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 69)* 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 5-13
```
storage:
  class: sfSessionStorage
  param:
    session_name: symfony
```

Default configuration for the `test` *environment*:

Listing 5-14
```
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()`[8] 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()`[9] 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:

8. http://www.php.net/session_set_cookie_params
9. http://www.php.net/session_cache_limiter

Listing 5-15
```
view_cache_manager:
  class: sfViewCacheManager
  param:
    cache_key_use_vary_headers: true
    cache_key_use_host_name:    true
```

This factory is only created if the `cache` *(page 32)* setting is set to `true`.

Most configuration of this factory is done via the `view_cache` factory, which defines the underlying cache object used by the view cache manager.

`cache_key_use_vary_headers`

The `cache_key_use_vary_headers` option specifies if the cache keys should include the vary headers part. In practice, it says if the page cache should be HTTP header dependent, as specified in `vary` cache parameter (default value: `true`).

`cache_key_use_host_name`

The `cache_key_use_host_name` option specifies if the cache keys should include the host name part. In practice, it says if page cache should be hostname dependent (default value: `true`).

`view_cache`

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

Default configuration:

Listing 5-16
```
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 32)* setting is set to `true`.

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 5-17
```
i18n:
  class: sfI18N
  param:
```

```
    source:               XLIFF
    debug:                false
    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 32)* setting is set to `true`.

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 `true`, 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 5-18

```
routing:
  class: sfPatternRouting
  param:
    load_configuration:              true
    suffix:                          ''
    default_module:                  default
    default_action:                  index
    debug:                           %SF_DEBUG%
    logging:                         %SF_LOGGING_ENABLED%
```

```
    generate_shortest_url:            false
    extra_parameters_as_query_string: false
    cache:                            ~
```

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

Default: none

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.

`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).

`logger`

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

Default configuration:

Listing 5-19

```
logger:
  class: sfAggregateLogger
  param:
    level: debug
    loggers:
      sf_web_debug:
        class: sfWebDebugLogger
        param:
          level: debug
          condition:       %SF_WEB_DEBUG%
          xdebug_logging:  false
          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 5-20

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

If you don't use the `sfAggregateLogger`, don't forget to specify a `null` value for the `loggers` parameter.

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

`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 5-21

```
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 90)*.

Configuration File

The configuration of such a module can be done in the `apps/backend/modules/model/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 singular name set in the generator options. If no explicit value is defined, singular name will default to the underscored version of the model class name (i.e. CamelCase becomes camel_case)

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 record
`executeEdit()`	`edit` view action
`executeUpdate()`	Updates a record
`executeDelete()`	Deletes a record
`executeBatch()`	Executes a batch action
`executeBatchDelete()`	Executes the `_delete` batch action
`processForm()`	Processes the record 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 single 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 59)*
  - `label` *(page 57)*
  - `action` *(page 59)*
  - `credentials` *(page 53)*
- `fields` *(page 57)*
  - `label` *(page 57)*
  - `help` *(page 57)*
  - `attributes` *(page 57)*
  - `credentials` *(page 53)*
  - `renderer` *(page 57)*
  - `renderer_arguments` *(page 58)*
  - `type` *(page 58)*
  - `date_format` *(page 58)*
- `list` *(page 59)*
  - `title` *(page 59)*
  - `display` *(page 59)*
  - `hide` *(page 60)*
  - `layout` *(page 60)*
  - `params` *(page 60)*
  - `sort` *(page 60)*
  - `max_per_page` *(page 61)*
  - `pager_class` *(page 61)*
  - `batch_actions` *(page 61)*
  - `object_actions` *(page 61)*
  - `actions` *(page 61)*
  - `peer_method` *(page 62)*
  - `peer_count_method` *(page 62)*
  - `table_method` *(page 62)*
  - `table_count_method` *(page 62)*
- `filter` *(page 62)*
  - `display` *(page 59)*
  - `class` *(page 63)*
- `form` *(page 63)*
  - `display` *(page 59)*
  - `class` *(page 63)*
- `edit` *(page 63)*
  - `title` *(page 59)*
  - `actions` *(page 61)*
- `new` *(page 64)*
  - `title` *(page 59)*
  - `actions` *(page 61)*

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.

`type`

Default: `Text` for virtual columns

The `type` option defines the type of the column. By default, symfony uses the type defined in your model definition, but if you create a virtual column, you can override the default `Text` type by one of the valid types:

- `ForeignKey`
- `Boolean`
- `Date`
- `Time`
- `Text`
- `Enum` (only available for Doctrine)

`date_format`

Default: `f`

The `date_format` option defines the format to use when displaying dates. It can be any format recognized by the `sfDateFormat` class. This option is not used when the field type is `Date`.

The following tokens can be used for the format:

- `G`: Era
- `y`: year
- `M`: mon
- `d`: mday
- `h`: Hour12
- `H`: hours
- `m`: minutes
- `s`: seconds
- `E`: wday
- `D`: yday
- `F`: DayInMonth
- `w`: WeekInYear
- `W`: WeekInMonth
- `a`: AMPM
- `k`: HourInDay
- `K`: HourInAMPM
- `z`: TimeZone

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.

label

Default: The action key

The `label` 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.

Listing 6-9

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

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:

Listing 6-10
```
config:
  list:
    hide: [created_at, updated_at]
```

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:

Listing 6-11
```
config:
  list:
    params:  |
      %%title%% written by %%author%% and published on %%published_at%%.
```

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. The list of actions is an associative array which keys are the route names and values an array of methods:

```
object_actions: { publish: get, publishBis: [get, post] }
```

Listing 6-13

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 6-14

```
# apps/backend/modules/model/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 26)*, benefits from the **configuration cascade mechanism** *(page 26)*, and can include **constants** *(page 24)*.

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 90)*.

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
      debug:
        realmemoryusage: true
        details:
          time:       { enabled: true }
          slow:       { enabled: true, threshold: 0.1 }
          mem:        { enabled: true }
          mempeak:    { enabled: true }
          memdelta:   { enabled: true }

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	`UTF8`
`persistent`	Whether to create persistent connections	`false`
`options`	A set of Propel options	-
`debug`	Options for the `DebugPDO` class	n/a

The `debug` entry defines all the options described in the Propel documentation[10]. The following YAML shows all the available options:

10. `http://www.propelorm.org/docs/api/1.4/runtime/propel-util/DebugPDO.html#class_details`

Listing 7-4

```
debug:
  realmemoryusage: true
  details:
    time:
      enabled: true
    slow:
      enabled: true
      threshold: 0.001
    memdelta:
      enabled: true
    mempeak:
      enabled: true
    method:
      enabled: true
    mem:
      enabled: true
    querycount:
      enabled: true
```

Doctrine

Default Configuration:

Listing 7-5

```
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	`UTF8`
`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`

Key	Description	Default Value
`validate`	Whether to enable data validation	`false`
`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 43)* factory class (`sfBasicSecurityUser` by default). The enforcement of the authentication and authorization is done by the `security` filter *(page 84)*.

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: false
```

As discussed in the introduction, the `security.yml` file benefits from the **configuration cascade mechanism** *(page 26)*, and can include **constants** *(page 24)*.

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 90)*.

Authentication

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

Listing 8-2

```
default:
  is_secure: false
```

By setting the `is_secure` key to `true` 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: true
```

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: false
```

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:   true
  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 32)* 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 45)* and `view_cache` *(page 46)* 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 `false`:

Listing 9-1

```
default:
  enabled:     false
  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 26)*, and can include **constants** *(page 24)*.

The `cache.yml` configuration file is cached as a PHP file; the process is automatically managed by the `sfCacheConfigHandler` class *(page 90)*.

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: `false`

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 90)*.

Route Classes

- Main Configuration *(page 76)*
  - `class` *(page 76)*
  - `options` *(page 76)*
  - `param` *(page 76)*
  - `params` *(page 76)*
  - `requirements` *(page 76)*
  - `type` *(page 77)*
  - `url` *(page 76)*
- `sfRoute` *(page 77)*
- `sfRequestRoute` *(page 77)*
  - `sf_method` *(page 77)*
- `sfObjectRoute` *(page 77)*
  - `allow_empty` *(page 78)*
  - `convert` *(page 78)*
  - `method` *(page 77)*
  - `model` *(page 77)*
  - `type` *(page 77)*
- `sfPropelRoute` *(page 78)*
  - `method_for_criteria` *(page 78)*
- `sfDoctrineRoute` *(page 78)*
  - `method_for_query` *(page 78)*
- `sfRouteCollection` *(page 78)*
- `sfObjectRouteCollection` *(page 78)*
  - `actions` *(page 78)*
  - `collection_actions` *(page 80)*
  - `column` *(page 79)*
  - `model` *(page 77)*
  - `model_methods` *(page 79)*
  - `module` *(page 79)*
  - `object_actions` *(page 80)*
  - `prefix_path` *(page 79)*
  - `requirements` *(page 76)*
  - `route_class` *(page 80)*
  - `segment_names` *(page 79)*
  - `with_show` *(page 79)*
  - `with_wildcard_routes` *(page 80)*
- `sfPropelRouteCollection` *(page 80)*
- `sfDoctrineRouteCollection` *(page 80)*

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 48)*)
- 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 48)*).

`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. The keys are the action names and the values are the valid methods for that action:

Listing 10-4
```
articles:
  options:
    collection_actions: { filter: post, filterBis: [post, get] }
    # ...
```

object_actions

Default: An empty array

The `object_actions` options defines an associative array of additional actions available for the object routes. The keys are the action names and the values are the valid methods for that action:

Listing 10-5
```
articles:
  options:
    object_actions: { publish: put, publishBis: [post, put] }
    # ...
```

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 24)* and project directories *(page 25)*.

As discussed in the introduction, the `app.yml` file is **environment-aware** *(page 26)*, and benefits from the **configuration cascade mechanism** *(page 26)*.

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 90)*.

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 26)*, and can include **constants** *(page 24)*.

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:     ~
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 90)*.

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 69)*.

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`, ...).

execution

Default configuration:

Listing 12-10

```
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 26)*, and can include **constants** *(page 24)*.

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 90)*.

Layout

Default configuration:

Listing 13-2
```
default:
  has_layout: true
  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 with the `include_stylesheets()` helper.

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 with the `include_javascripts()` helper.

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 31)* 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 26)*, and can include **constants** *(page 24)*.

The `autoload.yml` configuration file is cached as a PHP file; the process is automatically managed by the `sfAutoloadConfigHandler` class *(page 90)*.

The default configuration is fine for most projects:

Listing 14-1

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

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

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

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

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:      true
    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.

When adding your own configuration handlers, you must specify both the class name and the full path to your handler source file under the `class` and the `file` entries respectively. This is required as the configuration is initialized before the autoloading mechanism in sfApplicationConfiguration.

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`

Configuration File	Config Handler Class
factories.yml	sfFactoryConfigHandler
core_compile.yml	sfCompileConfigHandler
filters.yml	sfFilterConfigHandler
routing.yml	sfRoutingConfigHandler
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 26)*, and can include **constants** *(page 24)*.

The core_compile.yml configuration file is cached as a PHP file; the process is automatically managed by the sfCompileConfigHandler class *(page 90)*.

module.yml

The module.yml configuration file allows the configuration of a module. This configuration file is rarely used, and can only contain the entries defined below.

The module.yml file needs to be stored in the config/ sub-directory of a module to be loaded by symfony. The following code shows a typical module.yml content with the default values for all settings:

Listing 14-5

```
all:
  enabled:            true
  view_class:         sfPHP
  partial_view_class: sf
```

If the enabled parameter is set to false, all actions of a module are disabled. They are redirected to the module_disabled_module *(page 30)*/module_disabled_action action (as defined in settings.yml *(page 28)*).

The `view_class` parameter defines the view class used by all actions of the module (without the `View` suffix). It must inherit from `sfView`.

The `partial_view_class` parameter defines the view class used for partials of this module (without the `PartialView` suffix). It must inherit from `sfPartialView`.

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:

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

Listing 15-1

And notify it:

```
$dispatcher->notify($event);
```

Listing 15-2

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:

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

Listing 15-3

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:

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

Listing 15-4

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

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.

context.method_not_found

Notify method: `notifyUntil`

Default notifiers: `sfContext`

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

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

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

Parameter	Description
`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
`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.

debug

`debug.web.load_panels`

Notify method: `notify`

Default notifiers: `sfWebDebug`

The `debug.web.load_panels` event is notified after the call to the `configure` method of the `sfWebDebug` instance. You can use this event to manage your own panels.

`debug.web.view.filter_parameter_html`

Notify method: `filter`

Default notifiers: `sfWebDebugPanelView`

Parameter	Description
`parameter`	The parameter to filter

The `debug.web.view.filter_parameter_html` event filters each parameter rendered by the `sfWebDebugPanelView` panel.

doctrine

`doctrine.configure`

Notify method: `notify`

Default notifiers: `sfDoctrinePluginConfiguration`

The `doctrine.configure` event is notified after the Doctrine plugin has been configured.

`doctrine.filter_model_builder_options`

Notify method: `filter`

Default notifiers: `sfDoctrinePluginConfiguration`

The `doctrine.filter_model_builder_options` event filters the options for the Doctrine schema builder.

`doctrine.filter_cli_config`

Notify method: `filter`

Default notifiers: `sfDoctrinePluginConfiguration`

The `doctrine.filter_cli_config` event filters the configuration array for the Doctrine CLI.

`doctrine.configure_connection`

Notify method: `notify`

Default notifiers: `Doctrine_Manager` through `sfDoctrineDatabase`

Parameter	Description
`connection`	The `Doctrine_Connection` instance
`database`	The `sfDoctrineDatabase` instance

The `doctrine.configure_connection` event is notified when a Doctrine database is initialized for the first time.

`doctrine.admin.delete_object`

Notify method: `notify`

Default notifiers: Admin generator module class

Parameter	Description
`object`	The Doctrine object to delete

The `doctrine.admin.delete_object` event is notified when a Doctrine object is deleted in an admin generator module.

`doctrine.admin.save_object`

Notify method: `notify`

Default notifiers: Admin generator module class

Parameter	Description
`object`	The Doctrine object to delete

The `doctrine.admin.save_object` event is notified when a Doctrine object is saved in an admin generator module.

doctrine.admin.build_query

Notify method: `filter`

Default notifiers: Admin generator module class

The `doctrine.admin.build_query` event is notified when a Doctrine Query is created in an admin generator module.

doctrine.admin.pre_execute

Notify method: `notify`

Default notifiers: Admin generator module class

Parameter	Description
`configuration`	The admin generator configuration object

The `doctrine.admin.pre_execute` event is notified in the `preExecute()` method of admin generator modules.

form

form.post_configure

Notify method: `notify`

Default notifiers: `sfFormSymfony`

The `form.post_configure` event is notified after every form is configured.

form.filter_values

Notify method: `filter`

Default notifiers: `sfFormSymfony`

The `form.filter_values` event filters the merged, tainted parameters and files array just prior to binding.

form.validation_error

Notify method: `notify`

Default notifiers: `sfFormSymfony`

Parameter	Description
`error`	The error instance

The `form.validation_error` event is notified whenever form validation fails.

form.method_not_found

Notify method: `notifyUntil`

Default notifiers: `sfFormSymfony`

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

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

mailer

mailer.configure

Notify method: `notify`

Default notifiers: `sfMailer`

The `mailer.configure` event is notified after the mailer instance has been configured. The mailer instance is the subject of 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.

`propel`

`propel.configure`

Notify method: `notify`
Default notifiers: `sfPropelPluginConfiguration`
The `propel.configure` event is notified after the Propel plugin has been configured.

`propel.filter_phing_args`

Notify method: `filter`
Default notifiers: `sfPropelBaseTask`
The `propel.filter_phing_args` event filters the configuration array for the Propel CLI.

`propel.filter_connection_config`

Notify method: `filter`
Default notifiers: `sfPropelDatabase`

Parameter	Description
`name`	The name of the connection
`database`	The `sfPropelDatabase` instance

The `propel.filter_connection_config` event is notified when a Propel database is initialized for the first time.

`propel.admin.delete_object`

Notify method: `notify`
Default notifiers: Admin generator module class

Parameter	Description
`object`	The Propel object to delete

The `propel.admin.delete_object` event is notified when a Propel object is deleted in an admin generator module.

`propel.admin.save_object`

Notify method: `notify`

Default notifiers: Admin generator module class

Parameter	Description
`object`	The Propel object to delete

The `propel.admin.save_object` event is notified when a Propel object is saved in an admin generator module.

`propel.admin.build_criteria`

Notify method: `filter`

Default notifiers: Admin generator module class

The `propel.admin.build_criteria` event is notified when a Propel Criteria is created in an admin generator module.

`propel.admin.pre_execute`

Notify method: `notify`

Default notifiers: Admin generator module class

Parameter	Description
`configuration`	The admin generator configuration object

The `propel.admin.pre_execute` event is notified in the `preExecute()` method of admin generator modules.

`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 [--xml] [task_name]
```

Listing 16-6

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

Option (Shortcut)	Default	Description
`--xml`	-	To output help as XML

The `help` task displays help for a given task:

```
./symfony help test:all
```

Listing 16-7

You can also output the help as XML by using the `--xml` option:

```
./symfony help test:all --xml
```

Listing 16-8

list

The `list` task lists tasks:

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

Listing 16-9

Argument	Default	Description
`namespace`	-	The namespace name

Option (Shortcut)	Default	Description
`--xml`	-	To output help as XML

The `list` task lists all tasks:

```
./symfony list
```

Listing 16-10

You can also display the tasks for a specific namespace:

```
./symfony list test
```

Listing 16-11

You can also output the information as XML by using the `--xml` option:

```
./symfony list --xml
```

Listing 16-12

app

app::routes

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

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

Listing 16-13

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

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

Listing 16-14
```
./symfony app:routes frontend
```

cache

cache::clear

The cache::clear task clears the cache:

Listing 16-15
```
$ php symfony cache:clear [--app["..."]] [--env["..."]] [--type["..."]]
```

Alias(es): cc

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

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

Listing 16-17
```
./symfony cache:clear --app=frontend --env=prod
```

To clear the cache for all prod environments:

Listing 16-18
```
./symfony cache:clear --env=prod
```

To clear the config cache for all prod environments:

Listing 16-19
```
./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:

```
$ php symfony configure:author  author
```
Listing 16-20

Argument	Default	Description
author	-	The project author

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

```
./symfony configure:author "Fabien Potencier
<fabien.potencier@symfony-project.com>"
```
Listing 16-21

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 16-22

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 16-23

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 16-24

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 16-25

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

```
./symfony configure:database --name=main --class=ProjectDatabase
mysql:host=localhost;dbname=example root mYsEcret
```
Listing 16-26

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

The `doctrine::build` task generate code based on your schema:

Listing 16-27
```
$ php symfony doctrine:build [--application[="..."]] [--env="..."]
[--no-confirmation] [--all] [--all-classes] [--model] [--forms]
[--filters] [--sql] [--db] [--and-migrate] [--and-load[="..."]]
[--and-append[="..."]]
```

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`dev`	The environment
`--no-confirmation`	-	Whether to force dropping of the database
`--all`	-	Build everything and reset the database
`--all-classes`	-	Build all classes
`--model`	-	Build model classes
`--forms`	-	Build form classes
`--filters`	-	Build filter classes
`--sql`	-	Build SQL
`--db`	-	Drop, create, and either insert SQL or migrate the database
`--and-migrate`	-	Migrate the database
`--and-load`	-	Load fixture data (multiple values allowed)
`--and-append`	-	Append fixture data (multiple values allowed)

The `doctrine:build` task generates code based on your schema:

Listing 16-28
```
./symfony doctrine:build
```

You must specify what you would like built. For instance, if you want model and form classes built use the `--model` and `--forms` options:

Listing 16-29
```
./symfony doctrine:build --model --forms
```

You can use the `--all` shortcut option if you would like all classes and SQL files generated and the database rebuilt:

Listing 16-30
```
./symfony doctrine:build --all
```

This is equivalent to running the following tasks:

Listing 16-31
```
./symfony doctrine:drop-db
./symfony doctrine:build-db
```

```
./symfony doctrine:build-model
./symfony doctrine:build-forms
./symfony doctrine:build-filters
./symfony doctrine:build-sql
./symfony doctrine:insert-sql
```

You can also generate only class files by using the `--all-classes` shortcut option. When this option is used alone, the database will not be modified.

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

Listing 16-32

The `--and-migrate` option will run any pending migrations once the builds are complete:

```
./symfony doctrine:build --db --and-migrate
```

Listing 16-33

The `--and-load` option will load data from the project and plugin `data/fixtures/` directories:

```
./symfony doctrine:build --db --and-migrate --and-load
```

Listing 16-34

To specify what fixtures are loaded, add a parameter to the `--and-load` option:

```
./symfony doctrine:build --all --and-load="data/fixtures/dev/"
```

Listing 16-35

To append fixture data without erasing any records from the database, include the `--and-append` option:

```
./symfony doctrine:build --all --and-append
```

Listing 16-36

`doctrine::build-db`

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

```
$ php symfony doctrine:build-db [--application["..."]] [--env="..."]
[database1] ... [databaseN]
```

Listing 16-37

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

Argument	Default	Description
`database`	-	A specific database

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

The `doctrine:build-db` task creates one or more databases based on configuration in `config/databases.yml`:

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

Listing 16-38

You can specify what databases to create by providing their names:

```
./symfony doctrine:build-db slave1 slave2
```

Listing 16-39

doctrine::build-filters

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

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

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

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

Listing 16-41
```
./symfony doctrine:build-filters
```

This task creates form filter classes based on the model. The classes are created in `lib/doctrine/filter`.

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

doctrine::build-forms

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

Listing 16-42
```
$ php symfony doctrine:build-forms [--application[="..."]] [--env="..."]
[--model-dir-name="..."] [--form-dir-name="..."] [--generator-class="..."]
```

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`dev`	The environment
`--model-dir-name`	`model`	The model dir name
`--form-dir-name`	`form`	The form dir name
`--generator-class`	`sfDoctrineFormGenerator`	The generator class

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

Listing 16-43
```
./symfony doctrine:build-forms
```

This task creates form classes based on the model. The classes are created in `lib/doctrine/form`.

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

doctrine::build-model

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

```
$ php symfony doctrine:build-model [--application[="..."]] [--env="..."]
```
Listing 16-44

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

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

```
./symfony doctrine:build-model
```
Listing 16-45

The task read the schema information in `config/doctrine/*.yml` from the project and all enabled 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:

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

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:

```
./symfony doctrine:build-schema
```
Listing 16-47

The task creates a yml file in `config/doctrine`

`doctrine::build-sql`

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

```
$ php symfony doctrine:build-sql [--application[="..."]] [--env="..."]
```
Listing 16-48

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 16-49

The generated SQL is optimized for the database configured in `config/databases.yml`:

```
doctrine.database = mysql
```
Listing 16-50

doctrine::clean-model-files

The `doctrine::clean-model-files` task delete all generated model classes for models which no longer exist in your YAML schema:

Listing 16-51
```
$ php symfony doctrine:clean-model-files [--no-confirmation]
```

Alias(es): `doctrine:clean`

Option (Shortcut)	Default	Description
`--no-confirmation`	-	Do not ask for confirmation

The `doctrine:clean-model-files` task deletes model classes that are not represented in project or plugin schema.yml files:

Listing 16-52
```
./symfony doctrine:clean-model-files
```

doctrine::create-model-tables

The `doctrine::create-model-tables` task drop and recreate tables for specified models.:

Listing 16-53
```
$ php symfony doctrine:create-model-tables [--application[="..."]]
[--env="..."] [models1] ... [modelsN]
```

Argument	Default	Description
`models`	-	The list of models

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

The `doctrine:create-model-tables` Drop and recreate tables for specified models:

Listing 16-54
```
./symfony doctrine:create-model-tables User
```

doctrine::data-dump

The `doctrine::data-dump` task dumps data to the fixtures directory:

Listing 16-55
```
$ php symfony doctrine:data-dump [--application[="..."]] [--env="..."]
[target]
```

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 16-56

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
```

Listing 16-57

`doctrine::data-load`

The `doctrine::data-load` task loads YAML fixture data:

```
$ php symfony doctrine:data-load [--application[="..."]] [--env="..."]
[--append] [dir_or_file1] ... [dir_or_fileN]
```

Listing 16-58

Argument	Default	Description
`dir_or_file`	-	Directory or file to load

Option (Shortcut)	Default	Description
`--application`	1	The application name
`--env`	dev	The environment
`--append`	-	Don't delete current data in the database

The `doctrine:data-load` task loads data fixtures into the database:

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

Listing 16-59

The task loads data from all the files found in `data/fixtures/`.

If you want to load data from specific files or directories, you can append them as arguments:

```
./symfony doctrine:data-load data/fixtures/dev data/fixtures/users.yml
```

Listing 16-60

If you don't want the task to remove existing data in the database, use the `--append` option:

```
./symfony doctrine:data-load --append
```

Listing 16-61

`doctrine::delete-model-files`

The `doctrine::delete-model-files` task delete all the related auto generated files for a given model name.:

```
$ php symfony doctrine:delete-model-files [--no-confirmation] name1 ...
[nameN]
```

Listing 16-62

Argument	Default	Description
`name`	-	The name of the model you wish to delete all related files for.

Option (Shortcut)	Default	Description
`--no-confirmation`	-	Do not ask for confirmation

The `doctrine:delete-model-files` task deletes all files associated with certain models:

Listing 16-63
```
./symfony doctrine:delete-model-files Article Author
```

doctrine::dql

The `doctrine::dql` task execute a DQL query and view the results:

Listing 16-64
```
$ php symfony doctrine:dql [--application[="..."]] [--env="..."]
[--show-sql] [--table] dql_query [parameter1] ... [parameterN]
```

Argument	Default	Description
`dql_query`	-	The DQL query to execute
`parameter`	-	Query parameter

Option (Shortcut)	Default	Description
`--application`	1	The application name
`--env`	dev	The environment
`--show-sql`	-	Show the sql that would be executed
`--table`	-	Return results in table format

The `doctrine:dql` task executes a DQL query and displays the formatted results:

Listing 16-65
```
./symfony doctrine:dql "FROM User"
```

You can show the SQL that would be executed by using the `--show-sql` option:

Listing 16-66
```
./symfony doctrine:dql --show-sql "FROM User"
```

Provide query parameters as additional arguments:

Listing 16-67
```
./symfony doctrine:dql "FROM User WHERE email LIKE ?"
"%symfony-project.com"
```

doctrine::drop-db

The `doctrine::drop-db` task drops database for current model:

Listing 16-68
```
$ php symfony doctrine:drop-db [--application[="..."]] [--env="..."]
[--no-confirmation] [database1] ... [databaseN]
```

Argument	Default	Description
`database`	-	A specific database

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 one or more databases based on configuration in `config/databases.yml`:

Listing 16-69
```
./symfony doctrine:drop-db
```

You will be prompted for confirmation before any databases are dropped unless you provide the `--no-confirmation` option:

```
./symfony doctrine:drop-db --no-confirmation
```

Listing 16-70

You can specify what databases to drop by providing their names:

```
./symfony doctrine:drop-db slave1 slave2
```

Listing 16-71

doctrine::generate-admin

The `doctrine::generate-admin` task generates a Doctrine admin module:

```
$ php symfony doctrine:generate-admin [--module="..."] [--theme="..."]
[--singular="..."] [--plural="..."] [--env="..."]
[--actions-base-class="..."] application route_or_model
```

Listing 16-72

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
`--actions-base-class`	`sfActions`	The base class for the actions

The `doctrine:generate-admin` task generates a Doctrine admin module:

```
./symfony doctrine:generate-admin frontend Article
```

Listing 16-73

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 16-74

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 `with_wildcard_routes` option to the route:

```
article:
  class: sfDoctrineRouteCollection
  options:
    model:                Article
    with_wildcard_routes: true
```

Listing 16-75

doctrine::generate-migration

The doctrine::generate-migration task generate migration class:

Listing 16-76
```
$ php symfony doctrine:generate-migration [--application[="..."]]
[--env="..."] [--editor-cmd="..."] name
```

Argument	Default	Description
name	-	The name of the migration

Option (Shortcut)	Default	Description
--application	1	The application name
--env	dev	The environment
--editor-cmd	-	Open script with this command upon creation

The doctrine:generate-migration task generates migration template

Listing 16-77
```
./symfony doctrine:generate-migration AddUserEmailColumn
```

You can provide an --editor-cmd option to open the new migration class in your editor of choice upon creation:

Listing 16-78
```
./symfony doctrine:generate-migration AddUserEmailColumn --editor-cmd=mate
```

doctrine::generate-migrations-db

The doctrine::generate-migrations-db task generate migration classes from existing database connections:

Listing 16-79
```
$ php symfony doctrine:generate-migrations-db [--application[="..."]]
[--env="..."]
```

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

The doctrine:generate-migrations-db task generates migration classes from existing database connections:

Listing 16-80
```
./symfony doctrine:generate-migrations-db
```

doctrine::generate-migrations-diff

The doctrine::generate-migrations-diff task generate migration classes by producing a diff between your old and new schema.:

Listing 16-81
```
$ php symfony doctrine:generate-migrations-diff [--application[="..."]]
[--env="..."]
```

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

The `doctrine:generate-migrations-diff` task generates migration classes by producing a diff between your old and new schema.

```
./symfony doctrine:generate-migrations-diff
```

Listing 16-82

`doctrine::generate-migrations-models`

The `doctrine::generate-migrations-models` task generate migration classes from an existing set of models:

```
$ php symfony doctrine:generate-migrations-models [--application[="..."]]
[--env="..."]
```

Listing 16-83

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

The `doctrine:generate-migrations-models` task generates migration classes from an existing set of models:

```
./symfony doctrine:generate-migrations-models
```

Listing 16-84

`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="..."] [--actions-base-class="..."]
application module model
```

Listing 16-85

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
`--actions-base-class`	`sfActions`	The base class for the actions

The `doctrine:generate-module` task generates a Doctrine module:

Listing 16-86
```
./symfony doctrine: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 16-87
```
./symfony doctrine:generate-module --generate-in-cache frontend article
Article
```

The generator can use a customized theme by using the `--theme` option:

Listing 16-88
```
./symfony doctrine:generate-module --theme="custom" frontend article
Article
```

This way, you can create your very own module generator with your own conventions.

You can also change the default actions base class (default to sfActions) of the generated modules:

Listing 16-89
```
./symfony doctrine:generate-module --actions-base-class="ProjectActions"
frontend article Article
```

doctrine::generate-module-for-route

The `doctrine::generate-module-for-route` task generates a Doctrine module for a route definition:

Listing 16-90
```
$ php symfony doctrine:generate-module-for-route [--theme="..."]
[--non-verbose-templates] [--singular="..."] [--plural="..."]
[--env="..."] [--actions-base-class="..."] 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
`--actions-base-class`	`sfActions`	The base class for the actions

The `doctrine:generate-module-for-route` task generates a Doctrine module for a route definition:

Listing 16-91
```
./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:

```
$ php symfony doctrine:insert-sql [--application[="..."]] [--env="..."]
```

Listing 16-92

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

The `doctrine:insert-sql` task creates database tables:

```
./symfony doctrine:insert-sql
```

Listing 16-93

The task connects to the database and creates tables for all the `lib/model/doctrine/*.class.php` files.

doctrine::migrate

The `doctrine::migrate` task migrates database to current/specified version:

```
$ php symfony doctrine:migrate [--application[="..."]] [--env="..."]
[--up] [--down] [--dry-run] [version]
```

Listing 16-94

Argument	Default	Description
`version`	-	The version to migrate to

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`dev`	The environment
`--up`	-	Migrate up one version
`--down`	-	Migrate down one version
`--dry-run`	-	Do not persist migrations

The `doctrine:migrate` task migrates the database:

```
./symfony doctrine:migrate
```

Listing 16-95

Provide a version argument to migrate to a specific version:

```
./symfony doctrine:migrate 10
```

Listing 16-96

To migration up or down one migration, use the `--up` or `--down` options:

```
./symfony doctrine:migrate --down
```

Listing 16-97

If your database supports rolling back DDL statements, you can run migrations in dry-run mode using the `--dry-run` option:

Listing 16-98
```
./symfony doctrine:migrate --dry-run
```

generate

generate::app

The `generate::app` task generates a new application:

Listing 16-99
```
$ php symfony generate:app [--escaping-strategy="..."]
[--csrf-secret="..."] app
```

Argument	Default	Description
app	-	The application name

Option (Shortcut)	Default	Description
--escaping-strategy	1	Output escaping strategy
--csrf-secret	1	Secret to use for CSRF protection

The `generate:app` task creates the basic directory structure for a new application in the current project:

Listing 16-100
```
./symfony generate:app frontend
```

This task also creates two front controller scripts in the `web/` directory:

Listing 16-101
```
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`.

By default, the output escaping is enabled (to prevent XSS), and a random secret is also generated to prevent CSRF.

You can disable output escaping by using the `escaping-strategy` option:

Listing 16-102
```
./symfony generate:app frontend --escaping-strategy=false
```

You can enable session token in forms (to prevent CSRF) by defining a secret with the `csrf-secret` option:

Listing 16-103
```
./symfony generate:app frontend --csrf-secret=UniqueSecret
```

You can customize the default skeleton used by the task by creating a `%sf_data_dir%/skeleton/app` directory.

generate::module

The `generate::module` task generates a new module:

Listing 16-104
```
$ php symfony generate:module  application 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:

```
./symfony generate:module frontend article
```

Listing 16-105

The task can also change the author name found in the `actions.class.php` if you have configure it in `config/properties.ini`:

```
name=blog
author=Fabien Potencier <fabien.potencier@sensio.com>
```

Listing 16-106

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 [--orm="..."] [--installer="..."] name
[author]
```

Listing 16-107

Argument	Default	Description
name	-	The project name
author	Your name here	The project author

Option (Shortcut)	Default	Description
--orm	Doctrine	The ORM to use by default
--installer	-	An installer script to execute

The `generate:project` task creates the basic directory structure for a new project in the current directory:

```
./symfony generate:project blog
```

Listing 16-108

If the current directory already contains a symfony project, it throws a `sfCommandException`.

By default, the task configures Doctrine as the ORM. If you want to use Propel, use the `--orm` option:

```
./symfony generate:project blog --orm=Propel
```

Listing 16-109

If you don't want to use an ORM, pass `none` to `--orm` option:

```
./symfony generate:project blog --orm=none
```

Listing 16-110

You can also pass the `--installer` option to further customize the project:

Listing 16-111
```
./symfony generate:project blog --installer=./installer.php
```

You can optionally include a second `author` argument to specify what name to use as author when symfony generates new classes:

Listing 16-112
```
./symfony generate:project blog "Jack Doe"
```

`generate::task`

The `generate::task` task creates a skeleton class for a new task:

Listing 16-113
```
$ php symfony generate:task [--dir="..."] [--use-database="..."]
[--brief-description="..."] task_name
```

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`	`doctrine`	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:

Listing 16-114
```
./symfony generate:task namespace:name
```

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.

Listing 16-115
```
./symfony generate:task namespace:name --dir=plugins/myPlugin/lib/task
```

If you want the task to default to a connection other than `doctrine`, provide the name of this connection with the `--use-database` option:

Listing 16-116
```
./symfony generate:task namespace:name --use-database=main
```

The `--use-database` option can also be used to disable database initialization in the generated task:

Listing 16-117
```
./symfony generate:task namespace:name --use-database=false
```

You can also specify a description:

Listing 16-118
```
./symfony generate:task namespace:name --brief-description="Does
interesting things"
```

i18n

i18n::extract

The `i18n::extract` task extracts i18n strings from php files:

```
$ php symfony i18n:extract [--display-new] [--display-old] [--auto-save]
[--auto-delete] application culture
```

Listing 16-119

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:

```
./symfony i18n:extract frontend fr
```

Listing 16-120

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:

```
./symfony i18n:extract --display-new frontend fr
```

Listing 16-121

To save them in the i18n message catalogue, use the `--auto-save` option:

```
./symfony i18n:extract --auto-save frontend fr
```

Listing 16-122

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:

```
./symfony i18n:extract --display-old frontend fr
```

Listing 16-123

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:

```
./symfony i18n:extract --auto-delete frontend fr
```

Listing 16-124

i18n::find

The `i18n::find` task finds non “i18n ready” strings in an application:

```
$ php symfony i18n:find [--env="..."] application
```

Listing 16-125

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:

Listing 16-126
```
./symfony i18n:find frontend
```

This task is able to find non internationalized strings in pure HTML and in PHP code:

Listing 16-127
```
<p>Non i18n text</p>
<p><?php echo 'Test' ?></p>
```

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:

Listing 16-128
```
$ php symfony log:clear
```

The log:clear task clears all symfony log files:

Listing 16-129
```
./symfony log:clear
```

log::rotate

The log::rotate task rotates an application's log files:

Listing 16-130
```
$ php symfony log:rotate [--history="..."] [--period="..."] application env
```

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
--period	7	The period in days

The log:rotate task rotates application log files for a given environment:

Listing 16-131
```
./symfony log:rotate frontend dev
```

You can specify a period or a history option:

Listing 16-132
```
./symfony log:rotate frontend dev --history=10 --period=7
```

plugin

plugin::add-channel

The `plugin::add-channel` task add a new PEAR channel:

```
$ php symfony plugin:add-channel  name
```

Listing 16-133

Argument	Default	Description
`name`	-	The channel name

The `plugin:add-channel` task adds a new PEAR channel:

```
./symfony plugin:add-channel symfony.plugins.pear.example.com
```

Listing 16-134

plugin::install

The `plugin::install` task installs a plugin:

```
$ php symfony plugin:install [-s|--stability="..."] [-r|--release="..."]
[-c|--channel="..."] [-d|--install_deps] [--force-license] name
```

Listing 16-135

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 16-136

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 16-137

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 16-138

To force installation of all required dependencies, use the `install_deps` flag:

Listing 16-139
```
./symfony plugin:install --install-deps sfGuardPlugin
./symfony plugin:install -d sfGuardPlugin
```

By default, the PEAR channel used is `symfony-plugins` (plugins.symfony-project.org).

You can specify another channel with the `channel` option:

Listing 16-140
```
./symfony plugin:install --channel=mypearchannel sfGuardPlugin
./symfony plugin:install -c mypearchannel sfGuardPlugin
```

You can also install PEAR packages hosted on a website:

Listing 16-141
```
./symfony plugin:install http://somewhere.example.com/
sfGuardPlugin-1.0.0.tgz
```

Or local PEAR packages:

Listing 16-142
```
./symfony plugin:install /home/fabien/plugins/sfGuardPlugin-1.0.0.tgz
```

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:

Listing 16-143
```
$ php symfony plugin:list
```

The `plugin:list` task lists all installed plugins:

Listing 16-144
```
./symfony plugin:list
```

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 16-145
```
$ php symfony plugin:publish-assets [--core-only] [plugins1] ... [pluginsN]
```

Argument	Default	Description
`plugins`	-	Publish this plugin's assets

Option (Shortcut)	Default	Description
`--core-only`	-	If set only core plugins will publish their assets

The `plugin:publish-assets` task will publish web assets from all plugins.

Listing 16-146
```
./symfony plugin:publish-assets
```

In fact this will send the `plugin.post_install` event to each plugin.

You can specify which plugin or plugins should install their assets by passing those plugins' names as arguments:

```
./symfony plugin:publish-assets sfDoctrinePlugin
```

Listing 16-147

plugin::uninstall

The `plugin::uninstall` task uninstalls a plugin:

```
$ php symfony plugin:uninstall [-c|--channel="..."] [-d|--install_deps]
name
```

Listing 16-148

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:

```
./symfony plugin:uninstall sfGuardPlugin
```

Listing 16-149

The default channel is `symfony`.

You can also uninstall a plugin which has a different channel:

```
./symfony plugin:uninstall --channel=mypearchannel sfGuardPlugin
```

Listing 16-150

```
./symfony plugin:uninstall -c mypearchannel sfGuardPlugin
```

Or you can use the `channel/package` notation:

```
./symfony plugin:uninstall mypearchannel/sfGuardPlugin
```

Listing 16-151

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 [-s|--stability="..."] [-r|--release="..."]
[-c|--channel="..."] name
```

Listing 16-152

Argument	Default	Description
`name`	-	The plugin name

Option (Shortcut)	Default	Description
`--stability (-s)`	-	The preferred stability (stable, beta, alpha)
`--release (-r)`	-	The preferred version

Option (Shortcut)	Default	Description
`--channel (-c)`	-	The PEAR channel name

The `plugin:upgrade` task tries to upgrade a plugin:

Listing 16-153
```
./symfony plugin:upgrade sfGuardPlugin
```

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:

Listing 16-154
```
$ php symfony project:clear-controllers
```

The `project:clear-controllers` task clears all non production environment controllers:

Listing 16-155
```
./symfony project:clear-controllers
```

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/`:

Listing 16-156
```
index.php
frontend_dev.php
backend.php
backend_dev.php
```

After executing the `project:clear-controllers` task, two front controller scripts are left in `web/`:

Listing 16-157
```
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 16-158
```
$ php symfony project:deploy [--go] [--rsync-dir="..."]
[--rsync-options[="..."]] server
```

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 --progress`	To options to pass to the rsync executable

The `project:deploy` task deploys a project on a server:

```
./symfony project:deploy production
```

Listing 16-159

The server must be configured in `config/properties.ini`:

```
host=www.example.com
port=22
user=fabien
dir=/var/www/sfblog/
type=rsync
```

Listing 16-160

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:

```
./symfony project:deploy --go production
```

Listing 16-161

Files and directories configured in `config/rsync_exclude.txt` are not deployed:

```
.svn
/web/uploads/*
/cache/*
/log/*
```

Listing 16-162

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 16-163

Last, you can specify the options passed to the rsync executable, using the `rsync-options` option (defaults are `-azC --force --delete --progress`):

```
./symfony project:deploy --go --rsync-options=-avz
```

Listing 16-164

`project::disable`

The `project::disable` task disables an application in a given environment:

```
$ php symfony project:disable  env [app1] ... [appN]
```

Listing 16-165

Argument	Default	Description
`env`	-	The environment name
`app`	-	The application name

The `project:disable` task disables an environment:

Listing 16-166
```
./symfony project:disable prod
```

You can also specify individual applications to be disabled in that environment:

Listing 16-167
```
./symfony project:disable prod frontend backend
```

`project::enable`

The `project::enable` task enables an application in a given environment:

Listing 16-168
```
$ php symfony project:enable  env [app1] ... [appN]
```

Argument	Default	Description
`env`	-	The environment name
`app`	-	The application name

The `project:enable` task enables a specific environment:

Listing 16-169
```
./symfony project:enable frontend prod
```

You can also specify individual applications to be enabled in that environment:

Listing 16-170
```
./symfony project:enable prod frontend backend
```

`project::optimize`

The `project::optimize` task optimizes a project for better performance:

Listing 16-171
```
$ php symfony project:optimize  application [env]
```

Argument	Default	Description
`application`	-	The application name
`env`	`prod`	The environment name

The `project:optimize` optimizes a project for better performance:

Listing 16-172
```
./symfony project:optimize frontend prod
```

This task should only be used on a production server. Don't forget to re-run the task each time the project changes.

`project::permissions`

The `project::permissions` task fixes symfony directory permissions:

Listing 16-173
```
$ php symfony project:permissions
```

The `project:permissions` task fixes directory permissions:

Listing 16-174
```
./symfony project:permissions
```

project::send-emails

The `project::send-emails` task sends emails stored in a queue:

```
$ php symfony project:send-emails [--application[="..."]] [--env="..."]
[--message-limit[="..."]] [--time-limit[="..."]]
```

Listing 16-175

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`dev`	The environment
`--message-limit`	`0`	The maximum number of messages to send
`--time-limit`	`0`	The time limit for sending messages (in seconds)

The `project:send-emails` sends emails stored in a queue:

```
php symfony project:send-emails
```

Listing 16-176

You can limit the number of messages to send:

```
php symfony project:send-emails --message-limit=10
```

Listing 16-177

Or limit to time (in seconds):

```
php symfony project:send-emails --time-limit=10
```

Listing 16-178

project::validate

The `project::validate` task finds deprecated usage in a project:

```
$ php symfony project:validate
```

Listing 16-179

The `project:validate` task detects deprecated usage in your project.

```
./symfony project:validate
```

Listing 16-180

The task lists all the files you need to change before switching to symfony 1.4.

propel

propel::build

The `propel::build` task generate code based on your schema:

```
$ php symfony propel:build [--application[="..."]] [--env="..."]
[--no-confirmation] [--all] [--all-classes] [--model] [--forms]
[--filters] [--sql] [--db] [--and-load[="..."]] [--and-append[="..."]]
```

Listing 16-181

Option (Shortcut)	Default	Description
`--application`	`1`	The application name
`--env`	`dev`	The environment
`--no-confirmation`	`-`	Whether to force dropping of the database

Option (Shortcut)	Default	Description
`--all`	-	Build everything and reset the database
`--all-classes`	-	Build all classes
`--model`	-	Build model classes
`--forms`	-	Build form classes
`--filters`	-	Build filter classes
`--sql`	-	Build SQL
`--db`	-	Drop, create, and insert SQL
`--and-load`	-	Load fixture data (multiple values allowed)
`--and-append`	-	Append fixture data (multiple values allowed)

The `propel:build` task generates code based on your schema:

Listing 16-182
```
./symfony propel:build
```

You must specify what you would like built. For instance, if you want model and form classes built use the `--model` and `--forms` options:

Listing 16-183
```
./symfony propel:build --model --forms
```

You can use the `--all` shortcut option if you would like all classes and SQL files generated and the database rebuilt:

Listing 16-184
```
./symfony propel:build --all
```

This is equivalent to running the following tasks:

Listing 16-185
```
./symfony propel:build-model
./symfony propel:build-forms
./symfony propel:build-filters
./symfony propel:build-sql
./symfony propel:insert-sql
```

You can also generate only class files by using the `--all-classes` shortcut option. When this option is used alone, the database will not be modified.

Listing 16-186
```
./symfony propel:build --all-classes
```

The `--and-load` option will load data from the project and plugin `data/fixtures/` directories:

Listing 16-187
```
./symfony propel:build --db --and-load
```

To specify what fixtures are loaded, add a parameter to the `--and-load` option:

Listing 16-188
```
./symfony propel:build --all --and-load="data/fixtures/dev/"
```

To append fixture data without erasing any records from the database, include the `--and-append` option:

Listing 16-189
```
./symfony propel:build --all --and-append
```

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] [-F|--skip-forms]
[-C|--classes-only] [--phing-arg="..."]
```

Listing 16-190

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 16-191

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 16-192

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 16-193

To build all classes but skip initializing the database, use the `classes-only` option:

```
./symfony propel:build-all --classes-only
```

Listing 16-194

propel::build-all-load

The `propel::build-all-load` task generates Propel model and form classes, SQL, initializes the database, and loads data:

```
$ php symfony propel:build-all-load [--application[="..."]] [--env="..."]
[--connection="..."] [--no-confirmation] [-F|--skip-forms]
[-C|--classes-only] [--phing-arg="..."] [--append] [--dir="..."]
```

Listing 16-195

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

Option (Shortcut)	Default	Description
`--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 16-196
```
./symfony propel:build-all-load
```

The task is equivalent to:

Listing 16-197
```
./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 16-198
```
./symfony propel:buil-all-load --no-confirmation
```

`propel::build-filters`

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

Listing 16-199
```
$ php symfony propel:build-filters [--connection="..."]
[--model-dir-name="..."] [--filter-dir-name="..."] [--application[="..."]]
[--generator-class="..."]
```

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
`--generator-class`	`sfPropelFormFilterGenerator`	The generator class

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

Listing 16-200
```
./symfony propel:build-filters
```

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 16-201

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[="..."]]
[--generator-class="..."]
```

Listing 16-202

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
`--generator-class`	`sfPropelFormGenerator`	The generator class

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

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

Listing 16-203

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 16-204

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 16-205

Option (Shortcut)	Default	Description
`--phing-arg`	-	Arbitrary phing argument (multiple values allowed)

The `propel:build-model` task creates model classes from the schema:

```
./symfony propel:build-model
```

Listing 16-206

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 16-207
```
$ php symfony propel:build-schema [--application[="..."]] [--env="..."]
[--connection="..."] [--xml] [--phing-arg="..."]
```

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 16-208
```
./symfony propel:build-schema
```

By default, the task creates a YML file, but you can also create a XML file:

Listing 16-209
```
./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 16-210
```
$ php symfony propel:build-sql [--phing-arg="..."]
```

Option (Shortcut)	Default	Description
`--phing-arg`	-	Arbitrary phing argument (multiple values allowed)

The `propel:build-sql` task creates SQL statements for table creation:

Listing 16-211
```
./symfony propel:build-sql
```

The generated SQL is optimized for the database configured in `config/propel.ini`:

Listing 16-212
```
propel.database = mysql
```

`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 16-213

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 16-214

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 16-215

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 16-216

If you only want to dump some classes, use the `classes` option:

```
./symfony propel:data-dump --classes="Article,Category"
```

Listing 16-217

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 16-218

`propel::data-load`

The `propel::data-load` task loads YAML fixture data:

```
$ php symfony propel:data-load [--application[="..."]] [--env="..."]
[--append] [--connection="..."] [dir_or_file1] ... [dir_or_fileN]
```

Listing 16-219

Argument	Default	Description
`dir_or_file`	-	Directory or file to load

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

Option (Shortcut)	Default	Description
`--env`	`cli`	The environment
`--append`	-	Don't delete current data in the database
`--connection`	`propel`	The connection name

The `propel:data-load` task loads data fixtures into the database:

Listing 16-220
```
./symfony propel:data-load
```

The task loads data from all the files found in `data/fixtures/`.

If you want to load data from specific files or directories, you can append them as arguments:

Listing 16-221
```
./symfony propel:data-load data/fixtures/dev data/fixtures/users.yml
```

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

Listing 16-222
```
./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 16-223
```
./symfony propel:data-load --append
```

If you want to use a specific database configuration from an application, you can use the `application` option:

Listing 16-224
```
./symfony propel:data-load --application=frontend
```

`propel::generate-admin`

The `propel::generate-admin` task generates a Propel admin module:

Listing 16-225
```
$ php symfony propel:generate-admin [--module="..."] [--theme="..."]
[--singular="..."] [--plural="..."] [--env="..."]
[--actions-base-class="..."] 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
`--plural`	-	The plural name
`--env`	`dev`	The environment
`--actions-base-class`	`sfActions`	The base class for the actions

The `propel:generate-admin` task generates a Propel admin module:

Listing 16-226
```
./symfony propel:generate-admin frontend Article
```

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 16-227

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 `with_wildcard_routes` option to the route:

```
article:
  class: sfPropelRouteCollection
  options:
    model:                Article
    with_wildcard_routes: true
```

Listing 16-228

`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="..."] [--actions-base-class="..."] application module model
```

Listing 16-229

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
`--env`	`dev`	The environment
`--actions-base-class`	`sfActions`	The base class for the actions

The `propel:generate-module` task generates a Propel module:

```
./symfony propel:generate-module frontend article Article
```

Listing 16-230

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 16-231
```
./symfony propel:generate-module --generate-in-cache frontend article
Article
```

The generator can use a customized theme by using the `--theme` option:

Listing 16-232
```
./symfony propel:generate-module --theme="custom" frontend article Article
```

This way, you can create your very own module generator with your own conventions.

You can also change the default actions base class (default to sfActions) of the generated modules:

Listing 16-233
```
./symfony propel:generate-module --actions-base-class="ProjectActions"
frontend article Article
```

`propel::generate-module-for-route`

The `propel::generate-module-for-route` task generates a Propel module for a route definition:

Listing 16-234
```
$ php symfony propel:generate-module-for-route [--theme="..."]
[--non-verbose-templates] [--singular="..."] [--plural="..."]
[--env="..."] [--actions-base-class="..."] 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
`--actions-base-class`	`sfActions`	The base class for the actions

The `propel:generate-module-for-route` task generates a Propel module for a route definition:

Listing 16-235
```
./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:

Listing 16-236
```
$ php symfony propel:graphviz [--phing-arg="..."]
```

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 16-237

`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 16-238

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:

```
./symfony propel:insert-sql
```

Listing 16-239

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:

```
./symfony propel:insert-sql --no-confirmation
```

Listing 16-240

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:

```
$ php symfony propel:schema-to-xml
```

Listing 16-241

The `propel:schema-to-xml` task converts YML schemas to XML:

```
./symfony propel:schema-to-xml
```

Listing 16-242

`propel::schema-to-yml`

The `propel::schema-to-yml` task creates schema.yml from schema.xml:

Listing 16-243
```
$ php symfony propel:schema-to-yml
```

The `propel:schema-to-yml` task converts XML schemas to YML:

Listing 16-244
```
./symfony propel:schema-to-yml
```

symfony

symfony::test

The `symfony::test` task launches the symfony test suite:

Listing 16-245
```
$ php symfony symfony:test [-u|--update-autoloader] [-f|--only-failed]
[--xml="..."] [--rebuild-all]
```

Option (Shortcut)	Default	Description
`--update-autoloader (-u)`	-	Update the sfCoreAutoload class
`--only-failed (-f)`	-	Only run tests that failed last time
`--xml`	-	The file name for the JUnit compatible XML log file
`--rebuild-all`	-	Rebuild all generated fixture files

The `test:all` task launches the symfony test suite:

Listing 16-246
```
./symfony symfony:test
```

test

test::all

The `test::all` task launches all tests:

Listing 16-247
```
$ php symfony test:all [-f|--only-failed] [--xml="..."]
```

Option (Shortcut)	Default	Description
`--only-failed (-f)`	-	Only run tests that failed last time
`--xml`	-	The file name for the JUnit compatible XML log file

The `test:all` task launches all unit and functional tests:

Listing 16-248
```
./symfony test:all
```

The task launches all tests found in `test/`.

If some tests fail, you can use the `--trace` option to have more information about the failures:

Listing 16-249
```
./symfony test:all -t
```

Or you can also try to fix the problem by launching them by hand or with the `test:unit` and `test:functional` task.

Use the `--only-failed` option to force the task to only execute tests that failed during the previous run:

```
./symfony test:all --only-failed
```

Listing 16-250

Here is how it works: the first time, all tests are run as usual. But for subsequent test runs, only tests that failed last time are executed. As you fix your code, some tests will pass, and will be removed from subsequent runs. When all tests pass again, the full test suite is run... you can then rinse and repeat.

The task can output a JUnit compatible XML log file with the `--xml` options:

```
./symfony test:all --xml=log.xml
```

Listing 16-251

`test::coverage`

The `test::coverage` task outputs test code coverage:

```
$ php symfony test:coverage [--detailed] test_name lib_name
```

Listing 16-252

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 16-253

To output the lines not covered, pass the `--detailed` option:

```
./symfony test:coverage --detailed test/unit/model lib/model
```

Listing 16-254

`test::functional`

The `test::functional` task launches functional tests:

```
$ php symfony test:functional [--xml="..."] application [controller1] ... [controllerN]
```

Listing 16-255

Argument	Default	Description
`application`	-	The application name
`controller`	-	The controller name

Option (Shortcut)	Default	Description
`--xml`	-	The file name for the JUnit compatible XML log file

The `test:functional` task launches functional tests for a given application:

Listing 16-256
```
./symfony test:functional frontend
```

The task launches all tests found in `test/functional/%application%`.

If some tests fail, you can use the `--trace` option to have more information about the failures:

Listing 16-257
```
./symfony test:functional frontend -t
```

You can launch all functional tests for a specific controller by giving a controller name:

Listing 16-258
```
./symfony test:functional frontend article
```

You can also launch all functional tests for several controllers:

Listing 16-259
```
./symfony test:functional frontend article comment
```

The task can output a JUnit compatible XML log file with the `--xml` options:

Listing 16-260
```
./symfony test:functional --xml=log.xml
```

`test::unit`

The `test::unit` task launches unit tests:

Listing 16-261
```
$ php symfony test:unit [--xml="..."] [name1] ... [nameN]
```

Argument	Default	Description
`name`	-	The test name

Option (Shortcut)	Default	Description
`--xml`	-	The file name for the JUnit compatible XML log file

The `test:unit` task launches unit tests:

Listing 16-262
```
./symfony test:unit
```

The task launches all tests found in `test/unit`.

If some tests fail, you can use the `--trace` option to have more information about the failures:

Listing 16-263
```
./symfony test:unit -t
```

You can launch unit tests for a specific name:

Listing 16-264
```
./symfony test:unit strtolower
```

You can also launch unit tests for several names:

Listing 16-265
```
./symfony test:unit strtolower strtoupper
```

The task can output a JUnit compatible XML log file with the `--xml` options:

```
./symfony test:unit --xml=log.xml
```

Listing 16-266

Appendices

Appendix A

What's new in symfony 1.3/1.4?

This tutorial is a quick technical introduction for symfony 1.3/1.4. It is for developers who have already worked with symfony 1.2 and who want to quickly learn new features of symfony 1.3/1.4.

First, please note that symfony 1.3 is compatible with PHP 5.2.4 or later.

If you want to upgrade from 1.2, please read the UPGRADE[11] file found in the symfony distribution. You will have there all the information needed to safely upgrade your projects to symfony 1.3.

Mailer

As of symfony 1.3/1.4, there is a new default mailer based on SwiftMailer 4.1.

Sending an email is as simple as using the `composeAndSend()` method from an action:

Listing A-1

```
$this->getMailer()->composeAndSend('from@example.com', 'to@example.com',
'Subject', 'Body');
```

If you need to have more flexibility, you can also use the `compose()` method and send it afterwards. Here is for instance how to add an attachment to the message:

Listing A-2

```
$message = $this->getMailer()->
  compose('from@example.com', 'to@example.com', 'Subject', 'Body')->
  attach(Swift_Attachment::fromPath('/path/to/a/file.zip'))
;
$this->getMailer()->send($message);
```

As the mailer is quite powerful, refer to the documentation for more information.

Security

When a new application is created with the `generate:app` task, the security settings are now enabled by default:

- `escaping_strategy`: The value is now `true` by default (can be disabled with the `--escaping-strategy` option).
- `csrf_secret`: A random password is generated by default, and thus, the CSRF protection is enabled by default (can be disabled with the `--csrf-secret` option).

11. http://www.symfony-project.org/tutorial/1_4/en/upgrade

> It is highly recommended that you change the default generated password, by editing the `settings.yml` configuration file, or by using the `--csrf-secret` option.

Widgets

Default Labels

When a label is auto-generated from the field name, `_id` suffixes are now removed:

- `first_name` => First name (as before)
- `author_id` => Author (was "Author id" before)

`sfWidgetFormInputText`

The `sfWidgetFormInput` class is now abstract. Text input fields are now created with the `sfWidgetFormInputText` class. This change was made to ease introspection of form classes.

I18n widgets

The following widgets have been added:

- `sfWidgetFormI18nChoiceLanguage`
- `sfWidgetFormI18nChoiceCurrency`
- `sfWidgetFormI18nChoiceCountry`
- `sfWidgetFormI18nChoiceTimezone`

The first three of them replace the now deprecated `sfWidgetFormI18nSelectLanguage`, `sfWidgetFormI18nSelectCurrency`, and `sfWidgetFormI18nSelectCountry` widgets.

Fluent Interface

The widgets now implement a fluid interface for the following methods:

- `sfWidgetForm`: `setDefault()`, `setLabel()`, `setIdFormat()`, `setHidden()`
- `sfWidget`: `addRequiredOption()`, `addOption()`, `setOption()`, `setOptions()`, `setAttribute()`, `setAttributes()`
- `sfWidgetFormSchema`: `setDefault()`, `setDefaults()`, `addFormFormatter()`, `setFormFormatterName()`, `setNameFormat()`, `setLabels()`, `setLabel()`, `setHelps()`, `setHelp()`, `setParent()`
- `sfWidgetFormSchemaDecorator`: `addFormFormatter()`, `setFormFormatterName()`, `setNameFormat()`, `setLabels()`, `setHelps()`, `setHelp()`, `setParent()`, `setPositions()`

Validators

`sfValidatorRegex`

The `sfValidatorRegex` has a new `must_match` option. If set to `false`, the regex must not match for the validator to pass.

The `pattern` option of `sfValidatorRegex` can now be an instance of `sfCallable` that returns a regex when called.

`sfValidatorUrl`

The `sfValidatorUrl` has a new `protocols` option. This allows you to specify what protocols to allow:

```
$validator = new sfValidatorUrl(array('protocols' => array('http',
'https')));
```

Listing A-3

The following protocols are allowed by default:

- `http`
- `https`
- `ftp`
- `ftps`

`sfValidatorSchemaCompare`

The `sfValidatorSchemaCompare` class has two new comparators:

- `IDENTICAL`, which is equivalent to `===`;
- `NOT_IDENTICAL`, which is equivalent to `!==`;

`sfValidatorChoice`, `sfValidatorPropelChoice`, `sfValidatorDoctrineChoice`

The `sfValidatorChoice`, `sfValidatorPropelChoice`, `sfValidatorDoctrineChoice` validators have two new options that are enabled only if the `multiple` option is `true`:

- `min` The minimum number of values that need to be selected
- `max` The maximum number of values that need to be selected

I18n validators

The following validators have been added:

- `sfValidatorI18nChoiceTimezone`

Default Error Messages

You can now define default error messages globally by using the `sfValidatorBase::setDefaultMessage()` method:

```
sfValidatorBase::setDefaultMessage('required', 'This field is required.');
```

Listing A-4

The previous code will override the default 'Required.' message for all validators. Note that the default messages must be defined before any validator is created (the configuration class is a good place).

The `setRequiredMessage()` and `setInvalidMessage()` methods are deprecated and call the new `setDefaultMessage()` method.

When symfony displays an error, the error message to use is determined as follows:

- Symfony looks for a message passed when the validator was created (via the second argument of the validator constructor);
- If it is not defined, it looks for a default message defined with the `setDefaultMessage()` method;
- If it is not defined, it falls back to the default message defined by the validator itself (when the message has been added with the `addMessage()` method).

Fluent Interface

The validators now implement a fluid interface for the following methods:

- `sfValidatorSchema`: `setPreValidator()`, `setPostValidator()`
- `sfValidatorErrorSchema`: `addError()`, `addErrors()`
- `sfValidatorBase`: `addMessage()`, `setMessage()`, `setMessages()`, `addOption()`, `setOption()`, `setOptions()`, `addRequiredOption()`

`sfValidatorFile`

An exception is thrown when creating an instance of sfValidatorFile if `file_uploads` is disabled in `php.ini`.

Forms

`sfForm::useFields()`

The new `sfForm::useFields()` method removes all non-hidden fields from a form except the ones given as an argument. It is sometimes easier to explicitly give the fields you want to keep in a form, instead of unsetting all unneeded fields. For instance, when adding new fields to a base form, they won't automagically appear in your form until explicitly added (think of a model form where you add a new column to the related table).

Listing A-5

```
class ArticleForm extends BaseArticleForm
{
  public function configure()
  {
    $this->useFields(array('title', 'content'));
  }
}
```

By default, the array of fields is also used to change the fields order. You can pass `false` as the second argument to `useFields()` to disable the automatic reordering.

`sfForm::getEmbeddedForm($name)`

You can now access a particular embedded form using the `->getEmbeddedForm()` method.

`sfForm::renderHiddenFields()`

The `->renderHiddenFields()` method now renders hidden fields from embedded forms. An argument has been added to disable recursion, useful if you render embedded forms using a formatter.

Listing A-6

```
// render all hidden fields, including those from embedded forms
echo $form->renderHiddenFields();

// render hidden fields without recurring
echo $form->renderHiddenFields(false);
```

sfFormSymfony

The new `sfFormSymfony` class introduces the event dispatcher to symfony forms. You can access the dispatcher from inside your form classes as `self::$dispatcher`. The following form events are now notified by symfony:

- `form.post_configure`: This event is notified after every form is configured
- `form.filter_values`: This event filters the merged, tainted parameters and files array just prior to binding
- `form.validation_error`: This event is notified whenever form validation fails
- `form.method_not_found`: This event is notified whenever an unknown method is called

BaseForm

Every new symfony 1.3/1.4 project includes a `BaseForm` class that you can use to extend the Form component or add project-specific functionality. The forms generated by `sfDoctrinePlugin` and `sfPropelPlugin` automatically extend this class. If you create additional form classes they should now extend `BaseForm` rather than `sfForm`.

sfForm::doBind()

The cleaning of tainted parameters has been isolated in a developer-friendly method, `->doBind()`, which receives the merged array of parameters and files from `->bind()`.

sfForm(Doctrine|Propel)::doUpdateObject()

The Doctrine and Propel form classes now include a developer-friendly `->doUpdateObject()` method. This method receives an array of values from `->updateObject()` that has already been processed by `->processValues()`.

sfForm::enableLocalCSRFProtection() and sfForm::disableLocalCSRFProtection()

Using the `sfForm::enableLocalCSRFProtection()` and `sfForm::disableLocalCSRFProtection()` methods, you can now easily configure the CSRF protection from the `configure()` method of your form classes.

To disable the CSRF protection for a form, add the following line in its `configure()` method:

Listing A-7

```
$this->disableLocalCSRFProtection();
```

By calling the `disableLocalCSRFProtection()`, the CSRF protection will be disabled, even if you pass a CSRF secret when creating a form instance.

Fluent Interface

Some `sfForm` methods now implement a fluent interface: `addCSRFProtection()`, `setValidators()`, `setValidator()`, `setValidatorSchema()`, `setWidgets()`, `setWidget()`, `setWidgetSchema()`, `setOption()`, `setDefault()`, and `setDefaults()`.

Autoloaders

All symfony autoloaders are now case-insensitive. PHP is case-insensitive, now so is symfony.

`sfAutoloadAgain` (EXPERIMENTAL)

A special autoloader has been added that is just for use in debug mode. The new `sfAutoloadAgain` class will reload the standard symfony autoloader and search the filesystem for the class in question. The net effect is that you no longer have to run `symfony cc` after adding a new class to a project.

Tests

Speed up Testing

When you have a large suite of tests, it can be very time consuming to launch all tests every time you make a change, especially if some tests fail. That's because each time you fix a test, you should run the whole test suite again to ensure that you have not break something else. But as long as the failed tests are not fixed, there is no point in re-executing all other tests. As of symfony 1.3/1.4, the `test:all` and `symfony:test` tasks have a `--only-failed` (`-f` as a shortcut) option that forces the task to only re-execute tests that failed during the previous run:

Listing A-8

```
$ php symfony test:all --only-failed
```

Here is how it works: the first time, all tests are run as usual. But for subsequent test runs, only tests that failed last time are executed. As you fix your code, some tests will pass, and will be removed from subsequent runs. When all tests pass again, the full test suite is run… you can then rinse and repeat.

Functional Tests

When a request generates an exception, the `debug()` method of the response tester now outputs a readable text representation of the exception, instead of the normal HTML output. It makes debugging much easier.

`sfTesterResponse` has a new `matches()` method that runs a regex on the whole response content. It is of great help on non XML-like responses, where `checkElement()` is not useable. It also replaces the less-powerful `contains()` method:

Listing A-9

```
$browser->with('response')->begin()->
  matches('/I have \d+ apples/')->    // it takes a regex as an argument
  matches('!/I have \d+ apples/')->   // a ! at the beginning means that
the regex must not match
  matches('!/I have \d+ apples/i')->  // you can also add regex modifiers
end();
```

JUnit Compatible XML Output

The test tasks are now able to output a JUnit compatible XML file by using the `--xml` option:

```
$ php symfony test:all --xml=log.xml
```

Listing A-10

Easy Debugging

To ease the debugging when a test harness reports failed tests, you can now pass the `--trace` option to have a detailed output about the failures:

```
$ php symfony test:all -t
```

Listing A-11

Lime Output Colorization

As of symfony 1.3/1.4, lime does the right thing as far as colorization is concerned. It means, that you can almost always omit the second argument of the lime constructor of `lime_test`:

```
$t = new lime_test(1);
```

Listing A-12

`sfTesterResponse::checkForm()`

The response tester now includes a method to easily verify that all fields in a form have been rendered to the response:

```
$browser->with('response')->begin()->
  checkForm('ArticleForm')->
end();
```

Listing A-13

Or, if you prefer, you can pass a form object:

```
$browser->with('response')->begin()->
  checkForm($browser->getArticleForm())->
end();
```

Listing A-14

If the response includes multiple forms you have the option of providing a CSS selector to pinpoint which portion of the DOM to test:

```
$browser->with('response')->begin()->
  checkForm('ArticleForm', '#articleForm')->
end();
```

Listing A-15

`sfTesterResponse::isValid()`

You can now check whether a response is well-formed XML with the response tester's `->isValid()` method:

```
$browser->with('response')->begin()->
  isValid()->
end();
```

Listing A-16

You also validate the response against its document type be passing `true` as an argument:

```
$browser->with('response')->begin()->
  isValid(true)->
end();
```

Listing A-17

Alternatively, if you have a XSD or RelaxNG schema to validate against, you can provide the path to this file:

Listing A-18
```
$browser->with('response')->begin()->
  isValid('/path/to/schema.xsd')->
end();
```

Listen to `context.load_factories`

You can now add listeners for the `context.load_factories` event to your functional tests. This was not possible in previous versions of symfony.

Listing A-19
```
$browser->addListener('context.load_factories', array($browser,
'listenForNewContext'));
```

A better `->click()`

You can now pass any CSS selector to the `->click()` method, making it much easier to target the element you want semantically.

Listing A-20
```
$browser
  ->get('/login')
  ->click('form[action$="/login"] input[type="submit"]')
;
```

Tasks

The symfony CLI now attempts to detect the width of your terminal window and formats lines to fit. If detection is not possible the CLI defaults to 78 columns wide.

`sfTask::askAndValidate()`

There is a new `sfTask::askAndValidate()` method to ask a question to the user and validates its input:

Listing A-21
```
$answer = $this->askAndValidate('What is you email?', new
sfValidatorEmail());
```

The method also accepts an array of options (see the API doc for more information).

`symfony:test`

From time to time, developers need to run the symfony test suite to check that symfony works well on their specific platform. Until now, they had to know the `prove.php` script bundled with symfony to do that. As of symfony 1.3/1.4, there is a built-in task, `symfony:test` that launches the symfony core test suite from the command line, like any other task:

Listing A-22
```
$ php symfony symfony:test
```

If you were used to run `php test/bin/prove.php`, you should now run the equivalent `php data/bin/symfony symfony:test` command.

project:deploy

The `project:deploy` task has been slightly improved. It now displays the progress of the files transfer in real-time, but only if the `-t` option is passed. If not, the task is silent, except for errors of course. Speaking of errors, if one occurs, the output is on a red background to ease problem identification.

generate:project

As of symfony 1.3/1.4, Doctrine is the default configured ORM when executing the `generate:project` task:

```
$ php /path/to/symfony generate:project foo
```

Listing A-23

To generate a project for Propel, use the `--orm` option:

```
$ php /path/to/symfony generate:project foo --orm=Propel
```

Listing A-24

If you don't want to use Propel or Doctrine, you can pass `none` to the `--orm` option:

```
$ php /path/to/symfony generate:project foo --orm=none
```

Listing A-25

The new `--installer` option allows you to pass a PHP script that can further customize the newly created project. The script is executed in the context of the task, and so can use any of its methods. The more useful ones are the following: `installDir()`, `runTask()`, `ask()`, `askConfirmation()`, `askAndValidate()`, `reloadTasks()`, `enablePlugin()`, and `disablePlugin()`.

More information can be found in this post[12] from the official symfony blog.

You can also include a second "author" argument when generating a project, which specifies a value to use for the `@author` doc tag when symfony generates new classes.

```
$ php /path/to/symfony generate:project foo "Joe Schmo"
```

Listing A-26

sfFileSystem::execute()

The `sfFileSystem::execute()` methods replaces the `sfFileSystem::sh()` method with more powerful features. It takes callbacks for real-time processing of the `stdout` and `stderr` outputs. It also returns both outputs as an array. You can find one example of its usage in the `sfProjectDeployTask` class.

task.test.filter_test_files

The `test:*` tasks now filter test files through the `task.test.filter_test_files` event prior to running them. This event includes `arguments` and `options` parameters.

Enhancements to `sfTask::run()`

You can now pass an associative array of arguments and options to `sfTask::run()`:

```
$task = new sfDoctrineConfigureDatabaseTask($this->dispatcher,
$this->formatter);
$task->run(
```

Listing A-27

12. http://www.symfony-project.org/blog/2009/06/10/new-in-symfony-1-3-project-creation-customization

```
  array('dsn' => 'mysql:dbname=mydb;host=localhost'),
  array('name' => 'master')
);
```

The previous version, which still works:

Listing A-28
```
$task->run(
  array('mysql:dbname=mydb;host=localhost'),
  array('--name=master')
);
```

`sfBaseTask::setConfiguration()`

When calling a task that extends `sfBaseTask` from PHP, you no longer have to pass `--application` and `--env` options to `->run()`. Instead, you can simply set the configuration object directly by calling `->setConfiguration()`.

Listing A-29
```
$task = new sfDoctrineLoadDataTask($this->dispatcher, $this->formatter);
$task->setConfiguration($this->configuration);
$task->run();
```

The previous version, which still works:

Listing A-30
```
$task = new sfDoctrineLoadDataTask($this->dispatcher, $this->formatter);
$task->run(array(), array(
  '--application='.$options['application'],
  '--env='.$options['env'],
));
```

`project:disable` and `project:enable`

You can now wholesale disable or enable an entire environment using the `project:disable` and `project:enable` tasks:

Listing A-31
```
$ php symfony project:disable prod
$ php symfony project:enable prod
```

You can also specify which applications to disable in that environment:

Listing A-32
```
$ php symfony project:disable prod frontend backend
$ php symfony project:enable prod frontend backend
```

These tasks are backward compatible with their previous signature:

Listing A-33
```
$ php symfony project:disable frontend prod
$ php symfony project:enable frontend prod
```

`help` and `list`

The `help` and `list` tasks can now display their information as XML:

Listing A-34
```
$ php symfony list --xml
$ php symfony help test:all --xml
```

The output is based on the new `sfTask::asXml()` method, which returns a XML representation of a task object.

The XML output is mostly useful for third-party tools like IDEs.

`project:optimize`

Running this task reduces the number of disk reads performed during runtime by caching the location of your application's template files. This task should only be used on a production server. Don't forget to re-run the task each time the project changes.

```
$ php symfony project:optimize frontend
```

Listing A-35

`generate:app`

The `generate:app` task now checks for a skeleton directory in your project's `data/skeleton/app` directory before defaulting to the skeleton bundled in the core.

Sending an Email from a Task

You can now easily send an email from a task by using the `getMailer()` method.

Using the Routing in a Task

You can now easily get the routing object from a task by using the `getRouting()` method.

Exceptions

Autoloading

When an exception is thrown during autoloading, symfony now catches them and outputs an error to the user. That should solve some "White screen of death" pages.

Web Debug Toolbar

If possible, the web debug toolbar is now also displayed on exception pages in the development environment.

Propel Integration

Propel has been upgraded to version 1.4. Please visit Propel's site for more information on their upgrade (http://www.propelorm.org/wiki/Documentation/1.4).

Propel Behaviors

The custom builder classes symfony has relied on to extend Propel have been ported to Propel 1.4's new behaviors system.

`propel:insert-sql`

Before `propel:insert-sql` removes all data from a database, it asks for a confirmation. As this task can remove data from several databases, it now also displays the name of the connections of the related databases.

`propel:generate-module`, `propel:generate-admin`, `propel:generate-admin-for-route`

The `propel:generate-module`, `propel:generate-admin`, and `propel:generate-admin-for-route` tasks now takes a `--actions-base-class` option that allows the configuration of the actions base class for the generated modules.

Propel Behaviors

Propel 1.4 introduced an implementation of behaviors in the Propel codebase. The custom symfony builders have been ported into this new system.

If you would like to add native behaviors to your Propel models, you can do so in `schema.yml`:

Listing A-36
```
classes:
  Article:
    propel_behaviors:
      timestampable: ~
```

Or, if you use the old `schema.yml` syntax:

Listing A-37
```
propel:
  article:
    _propel_behaviors:
      timestampable: ~
```

Disabling form generation

You can now disable form generation on certain models by passing parameters to the `symfony` Propel behavior:

Listing A-38
```
classes:
  UserGroup:
    propel_behaviors:
      symfony:
        form: false
        filter: false
```

Note that you have to rebuild the model before that setting is respected, because the behaviour is attached to the model and does only exist after rebuilding it.

Using a different version of Propel

Using a different version of propel is as easy as setting the `sf_propel_runtime_path` and `sf_propel_generator_path` config variables in `ProjectConfiguration`:

Listing A-39
```
// config/ProjectConfiguration.class.php
public function setup()
{
  $this->enablePlugins('sfPropelPlugin');

  sfConfig::set('sf_propel_runtime_path', '/path/to/propel/runtime');
  sfConfig::set('sf_propel_generator_path', '/path/to/propel/generator');
}
```

Routing

Default Requirements

The default `\d+` requirement is now only applied to a `sfObjectRouteCollection` when the `column` option is the default `id`. This means you no longer have to provide an alternate requirement when a non-numeric column is specified (i.e. `slug`).

`sfObjectRouteCollection` options

A new `default_params` option has been added to `sfObjectRouteCollection`. It allows for default parameters to be registered for each generated route:

Listing A-40

```
forum_topic:
  class: sfDoctrineRouteCollection
  options:
    default_params:
      section: forum
```

CLI

Output Colorization

Symfony tries to guess if your console supports colors when you use the symfony CLI tool. But sometimes, symfony guesses wrong; for instance when you use Cygwin (because colorization is always turned off on the Windows platform).

As of symfony 1.3/1.4, you can force the use of colors for the output by passing the global `--color` option.

I18N

Data update

The data used for all I18N operations was updated from the `ICU project`. Symfony comes now with about 330 locale files, which is an increase of about 70 compared to Symfony 1.2. Please note that the updated data might be slightly different from what has been in there before, so for example test cases checking for the tenth item in a language list might fail.

Sorting according to user locale

All sorting on this locale dependent data is now also performed locale dependent. `sfCultureInfo->sortArray()` can be used for that.

Plugins

Before symfony 1.3/1.4, all plugins were enabled by default, except for the `sfDoctrinePlugin` and the `sfCompat10Plugin` ones:

Listing A-41

```
class ProjectConfiguration extends sfProjectConfiguration
{
  public function setup()
  {
    // for compatibility / remove and enable only the plugins you want
    $this->enableAllPluginsExcept(array('sfDoctrinePlugin',
'sfCompat10Plugin'));
  }
}
```

For freshly created projects with symfony 1.3/1.4, plugins must be explicitly enabled in the `ProjectConfiguration` class to be able to use them:

Listing A-42

```
class ProjectConfiguration extends sfProjectConfiguration
{
  public function setup()
  {
    $this->enablePlugins('sfDoctrinePlugin');
  }
}
```

The `plugin:install` task automatically enables the plugin(s) it installs (and `plugin:uninstall` disable them). If you install a plugin via Subversion, you still need to enable it by hand.

If you want to use a core-plugin, like `sfProtoculousPlugin` or `sfCompat10Plugin`, you just need to add the corresponding `enablePlugins()` statement in the `ProjectConfiguration` class.

If you upgrade a project from 1.2, the old behavior will still be active as the upgrade task does not change the `ProjectConfiguration` file. The behavior change is only for new symfony 1.3/1.4 projects.

`sfPluginConfiguration::connectTests()`

You can connect a plugin's tests to the `test:*` tasks by calling that plugin configuration's `->connectTests()` method in the new `setupPlugins()` method:

Listing A-43

```
class ProjectConfiguration extends sfProjectConfiguration
{
  public function setupPlugins()
  {
    $this->pluginConfigurations['sfExamplePlugin']->connectTests();
  }
}
```

Settings

`sf_file_link_format`

Symfony 1.3/1.4 formats file paths as clickable links whenever possible (i.e. the debug exception template). The `sf_file_link_format` is used for this purpose, if set, otherwise symfony will look for the `xdebug.file_link_format` PHP configuration value.

For example, if you want to open files in TextMate, add the following to `settings.yml`:

```
all:
  .settings:
    file_link_format: txmt://open?url=file://%f&line=%l
```

Listing A-44

The `%f` placeholder will be replaced with file's absolute path and the `%l` placeholder will be replaced with the line number.

Doctrine Integration

Doctrine has been upgraded to version 1.2. Please visit Doctrine's site for more information on their upgrade (http://www.doctrine-project.org/documentation/1_2/en).

Generating Form Classes

It is now possible to specify additional options for symfony in your Doctrine YAML schema files. We've added some options to disable the generation of form and filter classes.

For example in a typical many to many reference model, you don't need any form or filter form classes generated. So you can now do the following:

```
UserGroup:
  options:
    symfony:
      form: false
      filter: false
  columns:
    user_id:
      type: integer
      primary: true
    group_id:
      type: integer
      primary: true
```

Listing A-45

Form Classes Inheritance

When you generate forms from your models, your models contain inheritance. The generated child classes will respect the inheritance and generate forms that follow the same inheritance structure.

New Tasks

We have introduced a few new tasks to help you when developing with Doctrine.

Create Model Tables

You can now individually create the tables for a specified array of models. It will drop the tables first then re-create them for you. This is useful if you are developing some new models in an existing project/database and you don't want to blow away the whole database and just want to rebuild a subset of tables.

```
$ php symfony doctrine:create-model-tables Model1 Model2 Model3
```

Listing A-46

Delete Model Files

Often you will change your models, renaming things, remove unused models, etc. in your YAML schema files. When you do this, you then have orphaned model, form and filter classes. You can now manually clean out the generated files related to a model by using the `doctrine:delete-model-files` task.

Listing A-47
```
$ php symfony doctrine:delete-model-files ModelName
```

The above task will find all the related generated files and report them to you before asking you to confirm whether you would like to delete the files or not.

Clean Model Files

You can automate the above process and find out what models exist on the disk but do not exist in your YAML schema files by using the `doctrine:clean-model-files` task.

Listing A-48
```
$ php symfony doctrine:clean-model-files
```

The above command will compare your YAML schema files with the models and files that have been generated and determine what should be removed. These models are then passed on to the `doctrine:delete-model-files` task. It will ask you to confirm the removal of any files before actually deleting anything.

Build whatever

The new `doctrine:build` task allows you to specify what exactly you would like symfony and Doctrine to build. This task replicates the functionality in many of the existing combination-tasks, which have all been deprecated in favor of this more flexible solution.

Here are some possible uses of `doctrine:build`:

Listing A-49
```
$ php symfony doctrine:build --db --and-load
```

This will drop (`:drop-db`) and create (`:build-db`) the database, create the tables configured in `schema.yml` (`:insert-sql`) and load the fixture data (`:data-load`).

Listing A-50
```
$ php symfony doctrine:build --all-classes --and-migrate
```

This will build the model (`:build-model`), forms (`:build-forms`), form filters (`:build-filters`) and run any pending migrations (`:migrate`).

Listing A-51
```
$ php symfony doctrine:build --model --and-migrate --and-append=data/
fixtures/categories.yml
```

This will build the model (`:build-model`), migrate the database (`:migrate`) and append category fixtures data (`:data-load --append --dir=data/fixtures/categories.yml`).

For more information see the `doctrine:build` task's help page.

New option: `--migrate`

The following tasks now include a `--migrate` option, which will replace the nested `doctrine:insert-sql` task with `doctrine:migrate`.

- `doctrine:build-all`
- `doctrine:build-all-load`
- `doctrine:build-all-reload`

- `doctrine:build-all-reload-test-all`
- `doctrine:rebuild-db`
- `doctrine:reload-data`

`doctrine:generate-migration --editor-cmd`

The `doctrine:generate-migration` task now includes a `--editor-cmd` option which will execute once the migration class is created for easy editing.

```
$ php symfony doctrine:generate-migration AddUserEmailColumn
--editor-cmd=mate
```

Listing A-52

This example will generate the new migration class and open the new file in TextMate.

`doctrine:generate-migrations-diff`

This new task will automatically generate complete migration classes for you, based on your old and new schemas.

Create or drop specific connections

You can now specify database connection names when running `doctrine:build-db` and `doctrine:drop-db`:

```
$ php symfony doctrine:drop-db master slave1 slave2
```

Listing A-53

Date Setters and Getters

We've added two new methods for retrieving Doctrine date or timestamp values as PHP DateTime object instances.

```
echo $article->getDateTimeObject('created_at')
  ->format('m/d/Y');
```

Listing A-54

You can also set a dates value by simply calling the `setDateTimeObject` method and passing a valid `DateTime` instance.

```
$article->setDateTimeObject('created_at', new DateTime('09/01/1985'));
```

Listing A-55

`doctrine:migrate --down`

The `doctrine:migrate` now includes `up` and `down` options that will migrate your schema one step in the requested direction.

```
$ php symfony doctrine:migrate --down
```

Listing A-56

`doctrine:migrate --dry-run`

If your database supports rolling back DDL statements (MySQL does not), you can take advantage of the new `dry-run` option.

```
$ php symfony doctrine:migrate --dry-run
```

Listing A-57

Output DQL Task as Table of Data

When you would previously run the `doctrine:dql` command it will just output the data as YAML. We have added a new `--table` option. This option allows you to output the data as a table, similar to how it ouputs in the MySQL command line.

So now the following is possible.

Listing A-58
```
$ ./symfony doctrine:dql "FROM Article a" --table
>> doctrine  executing dql query
DQL: FROM Article a
+----+-----------+----------------+---------------------+---------------------+
| id | author_id | is_on_homepage | created_at          |
updated_at          |
+----+-----------+----------------+---------------------+---------------------+
| 1  | 1         |                | 2009-07-07 18:02:24 | 2009-07-07
18:02:24 |
| 2  | 2         |                | 2009-07-07 18:02:24 | 2009-07-07
18:02:24 |
+----+-----------+----------------+---------------------+---------------------+
(2 results)
```

Pass query parameters to `doctrine:dql`

The `doctrine:dql` task has also been enhanced to accept query parameters as arguments:

Listing A-59
```
$ php symfony doctrine:dql "FROM Article a WHERE name LIKE ?" John%
```

Debugging queries in functional tests

The `sfTesterDoctrine` class now includes a `->debug()` method. This method will output information about that queries that have been run in the current context.

Listing A-60
```
$browser->
  get('/articles')->
  with('doctrine')->debug()
;
```

You can view only the last few queries executed by passing an integer to the method, or show only queries that contain a substring or match a regular expression by passing a string.

Listing A-61
```
$browser->
  get('/articles')->
  with('doctrine')->debug('/from articles/i')
;
```

`sfFormFilterDoctrine`

The `sfFormFilterDoctrine` class can now be seeded a `Doctrine_Query` object via the `query` option:

Listing A-62
```
$filter = new ArticleFormFilter(array(), array(
  'query' => $table->createQuery()->select('title, body'),
));
```

The table method specified via `->setTableMethod()` (or now via the `table_method` option) is no longer required to return a query object. Any of the following are valid `sfFormFilterDoctrine` table methods:

Listing A-63

```
// works in symfony >= 1.2
public function getQuery()
{
  return $this->createQuery()->select('title, body');
}

// works in symfony >= 1.2
public function filterQuery(Doctrine_Query $query)
{
  return $query->select('title, body');
}

// works in symfony >= 1.3
public function modifyQuery(Doctrine_Query $query)
{
  $query->select('title, body');
}
```

Customizing a form filter is now easier. To add a filtering field, all you have to do is add the widget and a method to process it.

Listing A-64

```
class UserFormFilter extends BaseUserFormFilter
{
  public function configure()
  {
    $this->widgetSchema['name'] = new sfWidgetFormInputText();
    $this->validatorSchema['name'] = new
sfValidatorString(array('required' => false));
  }

  public function addNameColumnQuery($query, $field, $value)
  {
    if (!empty($value))
    {
      $query->andWhere(sprintf('CONCAT(%s.f_name, %1$s.l_name) LIKE ?',
$query->getRootAlias()), $value);
    }
  }
}
```

In earlier versions you would have need to extend `getFields()` in addition to creating a widget and method to get this to work.

Configuring Doctrine

You can now listen to the events `doctrine.configure` and `doctrine.configure_connection` to configure Doctrine. This means the Doctrine configuration can be easily customized from a plugin, as long as the plugin is enabled prior to `sfDoctrinePlugin`.

`doctrine:generate-module`, `doctrine:generate-admin`, `doctrine:generate-admin-for-route`

The `doctrine:generate-module`, `doctrine:generate-admin`, and `doctrine:generate-admin-for-route` tasks now takes a `--actions-base-class` option that allows the configuration of the actions base class for the generated modules.

Magic method doc tags

The magic getter and setter methods symfony adds to your Doctrine models are now represented in the doc header of each generated base class. If your IDE supports code completion, you should now see these `getFooBar()` and `setFooBar()` methods show up on model objects, where `FooBar` is a CamelCased field name.

Using a different version of Doctrine

Using a different version of Doctrine is as easy as setting the `sf_doctrine_dir` setting in `ProjectConfiguration`:

Listing A-65

```
// config/ProjectConfiguration.class.php
public function setup()
{
  $this->enablePlugins('sfDoctrinePlugin');

  sfConfig::set('sf_doctrine_dir', '/path/to/doctrine/lib');
}
```

Web Debug Toolbar

`sfWebDebugPanel::setStatus()`

Each panel in the web debug toolbar can specify a status that will affect its title's background color. For example, the background color of the log panel's title changes if any messages with a priority greater than `sfLogger::INFO` are logged.

`sfWebDebugPanel` request parameter

You can now specify a panel to be open on page load by appending a `sfWebDebugPanel` parameter to the URL. For example, appending `?sfWebDebugPanel=config` would cause the web debug toolbar to render with the config panel open.

Panels can also inspect request parameters by accessing the web debug `request_parameters` option:

Listing A-66

```
$requestParameters = $this->webDebug->getOption('request_parameters');
```

Partials

Slots improvements

The `get_slot()` and `include_slot()` helpers now accept a second parameter for specifying the default slot content to return if none is provided by the slot:

Listing A-67

```
<?php echo get_slot('foo', 'bar') // will output 'bar' if slot 'foo' is
not defined ?>
<?php include_slot('foo', 'bar') // will output 'bar' if slot 'foo' is not
defined ?>
```

Pagers

The `sfDoctrinePager` and `sfPropelPager` methods now implement the `Iterator` and `Countable` interfaces.

Listing A-68

```
<?php if (count($pager)): ?>
  <ul>
    <?php foreach ($pager as $article): ?>
      <li><?php echo link_to($article->getTitle(), 'article_show',
$article) ?></li>
    <?php endforeach; ?>
  </ul>
<?php else: ?>
  <p>No results.</p>
<?php endif; ?>
```

View cache

The view cache manager nows accept params in `factories.yml`. Generating the cache key for a view has been refactored in different methods to ease extending the class.

Two params are available in `factories.yml`:

- `cache_key_use_vary_headers` (default: `true`): specify if the cache keys should include the vary headers part. In practice, it says if the page cache should be http header dependent, as specified in `vary` cache parameter.
- `cache_key_use_host_name` (default: `true`): specify if the cache keys should include the host name part. In practice, it says if page cache should be hostname dependent.

Cache more

The view cache manager no longer refuses to cache based on whether there are values in the `$_GET` or `$_POST` arrays. The logic now simply confirms the current request is of the GET method before checking `cache.yml`. This means the following pages are now cacheable:

- `/js/my_compiled_javascript.js?cachebuster123`
- `/users?page=3`

Request

`getContent()`

The content of the request is now accessible via the `getContent()` method.

PUT and DELETE parameters

When a request comes in with either a PUT or a DELETE HTTP method with a content type set to `application/x-www-form-urlencoded`, symfony now parses the raw body and makes the parameters accessible like normal POST parameters.

Actions

`redirect()`

The `sfAction::redirect()` method family is now compatible with the `url_for()` signature introduced in symfony 1.2:

Listing A-69

```
// symfony 1.2
$this->redirect(array('sf_route' => 'article_show', 'sf_subject' =>
$article));

// symfony 1.3/1.4
$this->redirect('article_show', $article);
```

This enhancement was also applied to `redirectIf()` and `redirectUnless()`.

Helpers

`link_to_if()`, `link_to_unless()`

The `link_to_if()` and `link_to_unless()` helpers are now compatible with the `link_to()` signature introduced in symfony 1.2:

Listing A-70

```
// symfony 1.2
<?php echo link_to_unless($foo, '@article_show?id='.$article->getId()) ?>

// symfony 1.3/1.4
<?php echo link_to_unless($foo, 'article_show', $article) ?>
```

Context

You can now listen to `context.method_not_found` to dynamically add methods to `sfContext`. This is useful if you are added a lazy-loading factory, perhaps from a plugin.

Listing A-71

```
class myContextListener
{
  protected
    $factory = null;

  public function listenForMethodNotFound(sfEvent $event)
  {
    $context = $event->getSubject();

    if ('getLazyLoadingFactory' == $event['method'])
    {
      if (null === $this->factory)
```

```
      {
        $this->factory = new
myLazyLoadingFactory($context->getEventDispatcher());
      }

      $event->setReturnValue($this->factory);

      return true;
    }
  }
}
```

Appendix B

Upgrading Projects from 1.2 to 1.3/1.4

This document describes the changes made in symfony 1.3/1.4 and what need to be done to upgrade your symfony 1.2 projects.

If you want more detailed information on what has been changed/added in symfony 1.3/1.4, you can read the What's new?[13] tutorial.

symfony 1.3/1.4 is compatible with PHP 5.2.4 or later. It might also work with PHP 5.2.0 to 5.2.3 but there is no guarantee.

Upgrading to symfony 1.4

There is no upgrade task in symfony 1.4 as this version is the same as symfony 1.3 (minus all the deprecated features). To upgrade to 1.4, you must first upgrade to 1.3, and then switch to the 1.4 release.

Before upgrading to 1.4, you can also validate that your project does not use any deprecated class/method/function/setting/... by running the `project:validate` task:

Listing B-1

```
$ php symfony project:validate
```

The task lists all the files you need to change before switching to symfony 1.4.

Be aware that the task is a glorified regular expression and might gives you many false positives. Also, it cannot detect everything, so it is just a tool that helps you identifying possible problems, not a magic tool. You still need to read the DEPRECATED tutorial carefully.

`sfCompat10Plugin` and `sfProtoculousPlugin` have been removed from 1.4. If you are explicitly disabling them in your project's configuration class files, such as `config/ProjectConfiguration.class.php`, you must remove all mention of them from those files.

13. `http://www.symfony-project.org/tutorial/1_4/en/whats-new`

How to upgrade to symfony 1.3?

To upgrade a project:

- Check that all plugins used by your project are compatible with symfony 1.3
- If you don't use a SCM tool, please make a backup of your project.
- Upgrade symfony to 1.3
- Upgrade the plugins to their 1.3 version
- Launch the `project:upgrade1.3` task from your project directory to perform an automatic upgrade:

  ```
  $ php symfony project:upgrade1.3
  ```

  Listing B-2

  This task can be launched several times without any side effect. Each time you upgrade to a new symfony 1.3 beta / RC or the final symfony 1.3, you have to launch this task.
- You need to rebuild your models and forms due to some changes described below:

  ```
  # Doctrine
  $ php symfony doctrine:build --all-classes

  # Propel
  $ php symfony propel:build --all-classes
  ```

  Listing B-3

- Clear the cache:

  ```
  $ php symfony cache:clear
  ```

  Listing B-4

The remaining sections explain the main changes made in symfony 1.3 that need some kind of upgrade (automatic or not).

Deprecations

During the symfony 1.3 development, we have deprecated and removed some settings, classes, methods, functions, and tasks. Please refer to Deprecations in 1.3[14] for more information.

Autoloading

As of symfony 1.3, the files under the `lib/vendor/` directory are not autoloaded anymore by default. If you want to autoload some `lib/vendor/` sub-directories, add a new entry in the application `autoload.yml` configuration file:

```
autoload:
  vendor_some_lib:
    path:      %SF_LIB_DIR%/vendor/some_lib_dir
    recursive: on
```

Listing B-5

The automatic autoloading of the `lib/vendor/` directory was problematic for several reasons:

14. `http://www.symfony-project.org/tutorial/1_3/en/deprecated`

- If you put a library under the `lib/vendor/` directory that already has an autoload mechanism, symfony will re-parse the files and add a bunch of unneeded information in the cache (see #5893 - http://trac.symfony-project.org/ticket/5893).
- If your symfony directory is not exactly named `lib/vendor/symfony/`, the project autoloader will re-parse the whole symfony directory and some problems might occur (see #6064 - http://trac.symfony-project.org/ticket/6064).

Autoloading in symfony 1.3 is now case-insensitive.

Routing

The `sfPatternRouting::setRoutes()`, `sfPatternRouting::prependRoutes()`, `sfPatternRouting::insertRouteBefore()`, and `sfPatternRouting::connect()` methods do not return the routes as an array as they did in previous versions.

The `lazy_routes_deserialize` option has been removed as it is not needed anymore.

As of symfony 1.3, the cache for the routing is disabled, as this is the best option for most projects as far as performance are concerned. So, if you have not customized the routing cache, it will be automatically disabled for all your applications. If after upgrading to 1.3, your project is slower, you might want to add some routing cache to see if it helps. Here is the symfony 1.2 default configuration you can add back to your `factories.yml`:

Listing B-6

```
routing:
  param:
    cache:
      class: sfFileCache
      param:
        automatic_cleaning_factor: 0
        cache_dir:                 %SF_CONFIG_CACHE_DIR%/routing
        lifetime:                  31556926
        prefix:                    %SF_APP_DIR%/routing
```

JavaScripts and Stylesheets

Removal of the common filter

The `sfCommonFilter` has been deprecated and is not used anymore by default. This filter used to automatically inject the JavaScripts and stylesheets tags into the response content. You now need to manually include these assets by explicitly call the `include_stylesheets()` and `include_javascripts()` helpers in your layout:

Listing B-7

```
<?php include_javascripts() ?>
<?php include_stylesheets() ?>
```

It has been removed for several reasons:

- We already have a better, simple, and more flexible solution (the `include_stylesheets()` and `include_javascripts()` helpers)
- Even if the filter can be easily disabled, it is not an easy task as you must first know about its existence and its “behind the scene” magic work
- Using the helpers provides more fined-grained control over when and where the assets are included in the layout (the stylesheets in the `head` tag, and the JavaScripts just before the end of the `body` tag for instance)

- It is always better to be explicit, rather than implicit (no magic and no WTF effect; see the user mailing-list for a lot of complaints on this issue)
- It provides a small speed improvement

How to upgrade?

- The `common` filter need to be removed from all `filters.yml` configuration files (this is automatically done by the `project:upgrade1.3` task).
- You need to add `include_stylesheets()` and `include_javascripts()` calls in your layout(s) to have the same behavior as before (this is automatically done by the `project:upgrade1.3` task for HTML layouts contained in the `templates/` directories of your applications - they must have a `<head>` tag though; and you need to manually upgrade any other layout, or any page that does not have a layout but still relies on JavaScripts files and/or stylesheets).

The `sfCommonFilter` class is still bundled with symfony 1.3, and so you can still use it in your `filters.yml` if you need to.

Tasks

The following task classes have been renamed:

symfony 1.2	symfony 1.3
`sfConfigureDatabaseTask`	`sfDoctrineConfigureDatabaseTask` or `sfPropelConfigureDatabaseTask`
`sfDoctrineLoadDataTask`	`sfDoctrineDataLoadTask`
`sfDoctrineDumpDataTask`	`sfDoctrineDataDumpTask`
`sfPropelLoadDataTask`	`sfPropelDataLoadTask`
`sfPropelDumpDataTask`	`sfPropelDataDumpTask`

The signature for the `*:data-load` tasks has changed. Specific directories or files must now be provided as arguments. The `--dir` option has been removed.

```
$ php symfony doctrine:data-load data/fixtures/dev
```

Listing B-8

Formatters

The `sfFormatter::format()` third argument has been removed.

Escaping

The `esc_js_no_entities()`, refered to by `ESC_JS_NO_ENTITIES` was updated to correctly handle non-ANSI characters. Before this change all but characters with ANSI value 37 to 177 were escaped. Now it will only escape backslashes `\`, quotes `'` & `"` and linebreaks `\n` & `\r`. However it is unlikely that you previously relied on this broken behaviour.

Doctrine Integration

Required Doctrine Version

The externals to Doctrine have been updated to use the latest and greatest Doctrine 1.2 version. You can read about what is new in Doctrine 1.2 here[15].

Admin Generator Delete

The admin generator batch delete was changed to fetch the records and issue the `delete()` method to each one individually instead of issuing a single DQL query to delete them all. The reason is so that events for deleting each individual record are invoked.

Override Doctrine Plugin Schema

You can override the model included in a plugins YAML schema simply by defining that same model in your local schema. For example, to add an "email" column to sfDoctrineGuardPlugin's `sfGuardUser` model, add the following to `config/doctrine/schema.yml`:

Listing B-9

```
sfGuardUser:
  columns:
    email:
      type: string(255)
```

The package option is a feature of Doctrine and is used for the schemas of symfony plugins. This does not mean the package feature can be used independently to package your models. It must be used directly and only with symfony plugins.

Query logging

The Doctrine integration logs queries run using `sfEventDispatcher` rather than by accessing the logger object directly. Additionally, the subject of these events is either the connection or statement that is running the query. Logging is done by the new `sfDoctrineConnectionProfiler` class, which can be accessed via a `sfDoctrineDatabase` object.

Plugins

If you use the `enableAllPluginsExcept()` method to manage enabled plugins in your `ProjectConfiguration` class, be warned that we now sort the plugins by name to ensure consistency across different platforms.

Widgets

The `sfWidgetFormInput` class is now abstract. Text input fields are now created with the `sfWidgetFormInputText` class. This change was made to ease introspection of form classes.

15. `http://www.doctrine-project.org/upgrade/1_2`

Mailer

Symfony 1.3 has a new mailer factory. When creating a new application, the `factories.yml` has sensible defaults for the `test` and `dev` environments. But if you upgrade an existing project, you might want to update your `factories.yml` with the following configuration for these environments:

Listing B-10

```
mailer:
  param:
    delivery_strategy: none
```

With the previous configuration, emails won't be sent. Of course, they will still be logged, and the `mailer` tester will still work in your functional tests.

If you'd rather want to receive all emails to a single address, you can use the `single_address` delivery strategy (in the `dev` environment for instance):

Listing B-11

```
dev:
  mailer:
    param:
      delivery_strategy: single_address
      delivery_address:  foo@example.com
```

If your project uses an older version of Swiftmailer, you must remove it.

YAML

sfYAML is now more compatible with the 1.2 spec. Here are the changes you might need to do in your configuration files:

- Booleans can now only be represented with the `true` or `false` strings. If you used the alternative strings in the following list, you must replace them with either `true` or `false`:
  - `on`, `y`, `yes`, `+`
  - `off`, `n`, `no`, `-`

The `project:upgrade` task tells you where you use old syntax but does not fix them (to avoid loosing comments for instance). You must fix them by hand.

If you don't want to check all your YAML files, you can force the YAML parser to use the 1.1 YAML specification by using the `sfYaml::setSpecVersion()` method:

Listing B-12

```
sfYaml::setSpecVersion('1.1');
```

Propel

The custom Propel builder classes used in previous versions of symfony have been replaced with new Propel 1.4 behavior classes. To take advantage of this enhancement your project's `propel.ini` file must be updated.

Remove the old builder classes:

Listing B-13
```
; builder settings
propel.builder.peer.class              =
plugins.sfPropelPlugin.lib.builder.SfPeerBuilder
propel.builder.object.class            =
plugins.sfPropelPlugin.lib.builder.SfObjectBuilder
propel.builder.objectstub.class        =
plugins.sfPropelPlugin.lib.builder.SfExtensionObjectBuilder
propel.builder.peerstub.class          =
plugins.sfPropelPlugin.lib.builder.SfExtensionPeerBuilder
propel.builder.objectmultiextend.class =
plugins.sfPropelPlugin.lib.builder.SfMultiExtendObjectBuilder
propel.builder.mapbuilder.class        =
plugins.sfPropelPlugin.lib.builder.SfMapBuilderBuilder
```

And add the new behavior classes:

Listing B-14
```
; behaviors
propel.behavior.default                          = symfony,symfony_i18n
propel.behavior.symfony.class                    =
plugins.sfPropelPlugin.lib.behavior.SfPropelBehaviorSymfony
propel.behavior.symfony_i18n.class               =
plugins.sfPropelPlugin.lib.behavior.SfPropelBehaviorI18n
propel.behavior.symfony_i18n_translation.class =
plugins.sfPropelPlugin.lib.behavior.SfPropelBehaviorI18nTranslation
propel.behavior.symfony_behaviors.class          =
plugins.sfPropelPlugin.lib.behavior.SfPropelBehaviorSymfonyBehaviors
propel.behavior.symfony_timestampable.class      =
plugins.sfPropelPlugin.lib.behavior.SfPropelBehaviorTimestampable
```

The `project:upgrade` task attempts to make this change for you, but may be unable to if you've make local changes to `propel.ini`.

The `BaseFormFilterPropel` class was incorrectly generated in `lib/filter/base` in symfony 1.2. This has been corrected in symfony 1.3; the class is now be generated in `lib/filter`. The `project:upgrade` task will move this file for you.

Tests

The unit test bootstrap file, `test/bootstrap/unit.php`, has been enhanced to better handle autoloading of project class files. The following lines must be added to this script:

Listing B-15
```
$autoload = sfSimpleAutoload::getInstance(sfConfig::get('sf_cache_dir').'/
project_autoload.cache');
$autoload->loadConfiguration(sfFinder::type('file')->name('autoload.yml')->in(array(
  sfConfig::get('sf_symfony_lib_dir').'/config/config',
  sfConfig::get('sf_config_dir'),
)));
$autoload->register();
```

The `project:upgrade` task attempts to make this change for you, but may be unable to if you've make local changes to `test/bootstrap/unit.php`.

Appendix C

Deprecations and removals in 1.3

This document lists all settings, classes, methods, functions, and tasks that have been deprecated or removed in symfony 1.3.

Core Plugins

The following core plugins have been deprecated in symfony 1.3 and will be removed in symfony 1.4:

- `sfCompat10Plugin`: By deprecating this plugin, we also deprecate all other elements in the framework that rely on this plugin to work (1.0 admin generator, and 1.0 form system). It also includes the default admin generator theme for 1.0 located in `lib/plugins/sfPropelPlugin/data/generator/sfPropelAdmin`.
- `sfProtoculousPlugin`: The helpers provided by this plugin do not support unobstrusiveness, and as such should not be used anymore.

Methods and Functions

The following methods and functions have been deprecated in symfony 1.3 or before, and will be removed in symfony 1.4:

- `sfToolkit::getTmpDir()`: You can replace all occurrences of this method by `sys_get_temp_dir()`
- `sfToolkit::removeArrayValueForPath()`, `sfToolkit::hasArrayValueForPath()`, and `getArrayValueForPathByRef()`
- `sfValidatorBase::setInvalidMessage()`: You can replace it by a call to the new `sfValidatorBase::setDefaultMessage()` method
- `sfValidatorBase::setRequiredMessage()`: You can replace it by a call to the new `sfValidatorBase::setDefaultMessage()` method
- `sfTesterResponse::contains()`: You can use the more powerful `matches()` method
- `sfTestFunctionalBase` following methods: `isRedirected()`, `isStatusCode()`, `responseContains()`, `isRequestParameter()`, `isResponseHeader()`, `isUserCulture()`, `isRequestFormat()`, and `checkResponseElement()`: These methods have been deprecated since 1.2, and replaced with the tester classes.

- `sfTestFunctional` following methods: `isCached()`, `isUriCached()`: These methods have been deprecated since 1.2, and replaced with the tester classes.
- `sfFilesystem::sh()`: You can replace all occurrences of this method by calls to the new `sfFilesystem::execute()` method. Be warned that the returned value of this method is an array composed of the `stdout` output and the `stderr` output.
- `sfAction::getDefaultView()`, `sfAction::handleError()`, `sfAction::validate()`: These methods have been deprecated in symfony 1.1, and they was not really useful. As of symfony 1.1, they need the `compat_10` setting set to `on` to work.
- `sfComponent::debugMessage()`: Use the `log_message()` helper instead.
- `sfApplicationConfiguration::loadPluginConfig()`: Use `initializePlugins()` instead.
- `sfLoader::getHelperDirs()` and `sfLoader::loadHelpers()`: Use the same methods from the `sfApplicationConfiguration` object. As all methods of the class `sfLoader` are deprecated, the `sfLoader` class will be removed in symfony 1.4.
- `sfController::sendEmail()`: Use the new mailer feature of Symfony 1.3 instead.
- `sfGeneratorManager::initialize()`: It does nothing.
- `debug_message()`: Use the `log_message()` helper instead.
- `sfWebRequest::getMethodName()`: Use `getMethod()` instead.
- `sfDomCssSelector::getTexts()`: Use `matchAll()->getValues()`
- `sfDomCssSelector::getElements()`: Use `matchAll()`
- `sfVarLogger::getXDebugStack()`: Use `sfVarLogger::getDebugBacktrace()` instead.
- `sfVarLogger`: The logged `debug_stack` value is deprecated in favor of the `debug_backtrace` value.
- `sfContext::retrieveObjects()`: The method is only used by ObjectHelper, which is deprecated

The following methods and functions have been removed in symfony 1.3:

- `sfApplicationConfiguration::checkSymfonyVersion()`: see below for the explanation (`check_symfony_version` setting)

Classes

The following classes have been deprecated in symfony 1.3 and will be removed in symfony 1.4:

- `sfDoctrineLogger`: Use `sfDoctrineConnectionProfiler` instead.
- `sfNoRouting` and `sfPathInfoRouting`
- `sfRichTextEditor`, `sfRichTextEditorFCK`, and `sfRichTextEditorTinyMCE`: They have been replaced by the widget system (see the “Helpers” section below)
- `sfCrudGenerator`, `sfAdminGenerator`, `sfPropelCrudGenerator`, `sfPropelAdminGenerator`: These classes were used by the 1.0 admin generator
- `sfPropelUniqueValidator`, `sfDoctrineUniqueValidator`: These classes were used by the 1.0 form system

- `sfLoader`: see the "Methods and Functions" section
- `sfConsoleRequest`, `sfConsoleResponse`, `sfConsoleController`
- `sfDoctrineDataRetriever`, `sfPropelDataRetriever`: These classes are only used by ObjectHelper, which is deprecated
- `sfWidgetFormI18nSelectLanguage`, `sfWidgetFormI18nSelectCurrency`, and `sfWidgetFormI18nSelectCountry`: Use the corresponding `Choice` widgets (`sfWidgetFormI18nChoiceLanguage`, `sfWidgetFormI18nChoiceCurrency`, and `sfWidgetFormI18nChoiceCountry` respectively) as they act exactly in the same way, except they have more customization possibilities
- `sfWidgetFormChoiceMany`, `sfWidgetFormPropelChoiceMany`, `sfWidgetFormDoctrineChoiceMany`, `sfValidatorChoiceMany`, `sfValidatorPropelChoiceMany`, `sfValidatorPropelDoctrineMany`: Use the same classes but without `Many` at the end, and set the `multiple` option to `true`
- `SfExtensionObjectBuilder`, `SfExtensionPeerBuilder`, `SfMultiExtendObjectBuilder`, `SfNestedSetBuilder`, `SfNestedSetPeerBuilder`, `SfObjectBuilder`, `SfPeerBuilder`: The custom Propel builder classes have been ported to Propel 1.4's new behaviors system

The following classes have been deprecated in symfony 1.3:

- `sfCommonFilter`: see the "Removal of the common filter" section of the UPGRADE_TO_1_3 file for more information about the consequences and how to migrate your code.

Helpers

The following helper groups have been deprecated in symfony 1.3 and will be removed in symfony 1.4:

- All helpers related to the 1.0 form system as provided by the `sfCompat10Plugin`: `DateForm`, `Form`, `ObjectAdmin`, `Object`, and `Validation`

The `form_tag()` helper from the `Form` helper group has been moved to the `Url` helper group, and as such is still available in symfony 1.4.

Loading helpers from the PHP include path has been deprecated in 1.3 and removed in 1.4. Helpers must be located in one of the project, application or module `lib/helper/` directories.

Settings

The following settings (managed in the `settings.yml` configuration file) have been removed from symfony 1.3:

- `check_symfony_version`: This setting was introduced years ago to allow automatic cache cleaning in case of a change of the symfony version. It was mainly useful for shared hosting configuration where the symfony version is shared amongst all customers. As this is bad practice since symfony 1.1 (you need to embed the symfony version in each of your project), the settings does not make sense anymore. Moreover, when the setting is set to `on`, the check adds a small overhead to each request, as we need to get the content of a file.

- `max_forwards`: This settings controls the number of forwards allowed before symfony throws an exception. Making it configurable has no value. If you need more than 5 forwards, you have both a conception problem and a performance one.
- `sf_lazy_cache_key`: Introduced as a big performance improvement in symfony 1.2.6, this setting allowed you to turn on a lazy cache key generation for the view cache. While we think doing it lazy was the best idea, some people might have relied on `sfViewCacheManager::isCacheable()` being called even when the action itself wasn't cacheable. As of symfony 1.3, the behavior is the same as if `sf_lazy_cache_key` was set to `true`.
- `strip_comments`: The `strip_comments` was introduced to be able to disable the comment stripping because of some bugs in the tokenizer of some PHP 5.0.X versions. It was also used later on to avoid large memory consumption when the Tokenizer extension was not compiled with PHP. The first problem is not relevant anymore as the minimum version of PHP needed is 5.2 and the second one has already been fixed by removing the regular expression that simulated the comment stripping.
- `lazy_routes_deserialize`: This option is not needed anymore.

The following settings have been deprecated in symfony 1.3 and will be removed in symfony 1.4:

- `calendar_web_dir`, `rich_text_js_dir`: These settings are used by the Form helper group, which is deprecated in symfony 1.3.
- `validation_error_prefix`, `validation_error_suffix`, `validation_error_class`, `validation_error_id_prefix`: These settings are used by the Validation helper group, which is deprecated in symfony 1.3.
- `is_internal` (in `module.yml`): The `is_internal` flag was used to prevent actions from being called from a browser. This was added to protect email sending in symfony 1.0. As email support does not require this trick anymore, this flag will be removed and not checked anymore in the symfony core code.

Tasks

The following tasks have been removed in symfony 1.3:

- `project:freeze` and `project:unfreeze`: These tasks used to embed the symfony version used by a project inside the project itself. They are not needed anymore as the best practice has been to embed symfony in the project for a very long time now. Moreover, switching from one version of symfony to another is really simple now as you only need to change the path in the `ProjectConfiguration` class. Embedding by hand symfony is also very simple as you just need to copy the whole symfony directory somewhere in your project (`lib/vendor/symfony/` is the recommended one).

The following tasks are deprecated in symfony 1.3, and will be removed in symfony 1.4:

- All symfony 1.0 task aliases.
- `propel:init-admin`: This task generated admin generator modules for symfony 1.0.

The following Doctrine tasks have been merged into `doctrine:build` and will be removed in symfony 1.4:

- `doctrine:build-all`

- `doctrine:build-all-load`
- `doctrine:build-all-reload`
- `doctrine:build-all-reload-test-all`
- `doctrine:rebuild-db`
- `doctrine:reload-data`

Miscellaneous

The following behaviors are deprecated in symfony 1.3, and will be removed in symfony 1.4:

- The `sfParameterHolder::get()`, `sfParameterHolder::has()`, `sfParameterHolder::remove()`, `sfNamespacedParameterHolder::get()`, `sfNamespacedParameterHolder::has()`, and `sfNamespacedParameterHolder::remove()` methods support for the array notation (`[]`) is deprecated and won't be available in symfony 1.4 (better for performance).

The symfony CLI does not accept anymore the global `--dry-run` option as it was not used by any symfony built-in task. If one of your task relies on this option, you can just add it as a local option of your task class.

The Propel templates for the 1.0 admin generator and the 1.0 CRUD will be removed in symfony 1.4 (`plugins/sfPropelPlugin/data/generator/sfPropelAdmin/`).

The "Dynarch calendar" (found in data/web/calendar/) will be removed in symfony 1.4 as it is only used by the Form helper group, which will be also removed in symfony 1.4.

As of symfony 1.3, the unavailable page will only be looked for in the `%SF_APP_CONFIG_DIR%/` and `%SF_CONFIG_DIR%/` directories. If you still have it stored in `%SF_WEB_DIR%/errors/`, you must move it before migrating to symfony 1.4.

The `doc/` directory at the root of a project is not generated anymore, as it was not used by symfony itself. And so the related `sf_doc_dir` has also been removed.

The `sfDoctrinePlugin_doctrine_lib_path` setting, previously used to specify a custom Doctrine lib directory, has been deprecated in 1.3 and removed in 1.4. Use the `sf_doctrine_dir` setting instead.

All symfony `Base*` classes generated classes are not marked as `abstract`.

Appendix D

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/.