



Upgrading Projects from 1.2 to 1.3/ 1.4

This PDF is brought to you by
SENSIOLABS 

License: Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 Unported License
Version: upgrade-1.4-en-2009-12-05

Table of Contents

Array

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?](#)¹ 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:

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

*Listing
1-1*

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.

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

1. <http://www.symfony-project.org/tutorial/14/en/whats-new>

- 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:

Listing 1-2 `$ php symfony project:upgrade1.3`

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:

Listing 1-3

```
# Doctrine
$ php symfony doctrine:build --all-classes

# Propel
$ php symfony propel:build --all-classes
```

- Clear the cache:

Listing 1-4 `$ php symfony cache:clear`

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 1.3² 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:

Listing 1-5

```
autoload:
  vendor_some_lib:
    name:      vendor_some_lib
    path:      %SF_LIB_DIR%/vendor/some_lib_dir
    recursive: on
```

The automatic autoloading of the `lib/vendor/` directory was problematic for several reasons:

- 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>).

2. <http://www.symfony-project.org/tutorial/1.3/en/deprecated>

- 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`:

```
routing:
  param:
    cache:
      class: sfFileCache
      param:
        automatic_cleaning_factor: 0
        cache_dir:                %SF_CONFIG_CACHE_DIR%/routing
        lifetime:                  31556926
        prefix:                    %SF_APP_DIR%/routing
```

Listing
1-6

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:

```
<?php include_javascripts() ?>
<?php include_stylesheets() ?>
```

Listing
1-7

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
<code>sfConfigureDatabaseTask</code>	<code>sfDoctrineConfigureDatabaseTask</code> or <code>sfPropelConfigureDatabaseTask</code>
<code>sfDoctrineLoadDataTask</code>	<code>sfDoctrineDataLoadTask</code>
<code>sfDoctrineDumpDataTask</code>	<code>sfDoctrineDataDumpTask</code>
<code>sfPropelLoadDataTask</code>	<code>sfPropelDataLoadTask</code>
<code>sfPropelDumpDataTask</code>	<code>sfPropelDataDumpTask</code>

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.

Listing 1-8 `$ php symfony doctrine:data-load data/fixtures/dev`

Formatters

The `sfFormatter::format()` third argument has been removed.

Escaping

The `esc_js_no_entities()`, referred 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 here³.

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`:

```
sfGuardUser:
  columns:
    email:
      type: string(255)
```

Listing
1-9



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.

3. <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 1-10

```
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 1-11

```
dev:
  mailer:
    param:
      delivery_strategy: single_address
      delivery_address:  foo@example.com
```

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 losing 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 1-12

```
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 1-13

```
; 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:

```

; 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

```

*Listing
1-14*

The `project:upgrade` task attempts to make this change for you, but may be unable to if you've made 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:

```

$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();

```

*Listing
1-15*

The `project:upgrade` task attempts to make this change for you, but may be unable to if you've made local changes to `test/bootstrap/unit.php`.