



## What's new in symfony 1.3?

This PDF is brought to you by  
**SENSIOLABS** 

*License:* Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 Unported License  
*Version:* whats-new-1.3-en-2009-12-05

# Table of Contents

Array

# What's new in symfony 1.3?

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

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<sup>1</sup>](#) 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, 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:

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

*Listing  
1-1*

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:

```
$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);
```

*Listing  
1-2*

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). 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.

---

1. <http://www.symfony-project.org/tutorial/1.3/en/upgrade>

# 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.

## l18n 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  
1-3*

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:

- `sfValidatorI18nTimezone`

## Default Error Messages

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

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

*Listing  
1-4*

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 automatically appear in your form until explicitly added (think of a model form where you add a new column to the related table).

*Listing  
1-5*

```
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.

### `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 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:

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

*Listing  
1-6*

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, 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 1-7 $ 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 1-8 $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:

```
Listing 1-9 $ php symfony test:all --xml=log.xml
```

### 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:

```
Listing 1-10 $ php symfony test:all -t
```

## Lime Output Colorization

As of symfony 1.3, 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  
1-11*

```
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  
1-12*

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

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

*Listing  
1-13*

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  
1-14*

```
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  
1-15*

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

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

*Listing  
1-16*

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

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

*Listing  
1-17*

## 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 1-18 $browser->addListener('context.load_factories', array($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 1-19 $browser
->get('/login')
->click('form[action$="/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 1-20 $answer = $this->askAndValidate('What is your 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, there is a built-in task, `symfony:test` that launches the symfony core test suite from the command line, like any other task:

```
Listing 1-21 $ 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, Doctrine is the default configured ORM when executing the `generate:project` task:

```
Listing 1-22 $ php /path/to/symfony generate:project foo
```

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

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

*Listing  
1-23*

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  
1-24*

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<sup>2</sup> 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  
1-25*

### `sfFileSystem::execute()`

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

*Listing  
1-26*

The previous version, which still works:

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

*Listing  
1-27*

---

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

## `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 1-28 $task = new sfDoctrineLoadDataTask($this->dispatcher, $this->formatter);
$task->setConfiguration($this->configuration);
$task->run();
```

The previous version, which still works:

```
Listing 1-29 $task = new sfDoctrineLoadDataTask($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 1-30 $ php symfony project:disable prod
$ php symfony project:enable prod
```

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

```
Listing 1-31 $ php symfony project:disable prod frontend backend
$ php symfony project:enable prod frontend backend
```

These tasks are backward compatible with their previous signature:

```
Listing 1-32 $ 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 1-33 $ 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.

```
Listing 1-34 $ php symfony project:optimize frontend
```

## 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 catch 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://propel.phpdb.org/trac/wiki/Users/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 1-35*

```
classes:
  Article:
    propel_behaviors:
      timestampable: ~
```

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

*Listing 1-36*

```
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 1-37*

```
classes:
  UserGroup:
    propel_behaviors:
      symfony:
        form: false
        filter: false
```

## 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 1-38*

```
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, 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, all plugins were enabled by default, except for the `sfDoctrinePlugin` and the `sfCompat10Plugin` ones:

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

*Listing  
1-39*

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

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

*Listing  
1-40*

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 `sfProtocolousPlugin` 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 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  
1-41

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

## Settings

### `sf_file_link_format`

Symfony 1.3 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`:

Listing  
1-42

```
all:
    .settings:
        file_link_format: txmt://open?url=file://%f&line=%l
```

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

## Doctrine Integration

### 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:

Listing  
1-43

```
UserGroup:
    options:
```

```
symfony:
  form: false
  filter: false
columns:
  user_id:
    type: integer
    primary: true
  group_id:
    type: integer
    primary: true
```

## 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  
1-44*

### 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.

```
$ php symfony doctrine:delete-model-files ModelName
```

*Listing  
1-45*

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.

```
$ php symfony doctrine:clean-model-files
```

*Listing  
1-46*

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.

## Reload Data

It is a common need to want to blow away the databases completely then reload your data fixtures. The `doctrine:build-all-reload` task does this but it also does a bunch of other work, generating models, forms, filters, etc. and this can be time consuming in a large project. Now you can simply use the `doctrine:reload-data` task.

The following command.

```
Listing 1-47 $ php symfony doctrine:reload-data
```

Is equivalent to running these commands:

```
Listing 1-48 $ php symfony doctrine:drop-db
$ php symfony doctrine:build-db
$ php symfony doctrine:insert-sql
$ php symfony doctrine:data-load
```

## 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 1-49 $ 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 1-50 $ 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 1-51 $ 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  
1-52*

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.

## 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  
1-53*

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  
1-54*

## 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  
1-55*

## 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  
1-56*

## 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 outputs in the MySQL command line.

So now the following is possible.

```
$ ./symfony doctrine:dql "FROM Article a" --table
>> doctrine executing dql query
DQL: FROM Article a
```

*Listing  
1-57*

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)
```

## 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 1-58 $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 1-59 $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 1-60 $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 1-61 // 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.

```
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);
        }
    }
}
```

Listing  
1-62

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.

## 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 1-63 $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 1-64 <?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 1-65 <?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:

```
// symfony 1.2
$this->redirect(array('sf_route' => 'article_show', 'sf_subject' =>
    $article));

// symfony 1.3
$this->redirect('article_show', $article);
```

*Listing  
1-66*

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:

```
// symfony 1.2
<?php echo link_to_unless($foo, '@article_show?id='.$article->getId()) ?>

// symfony 1.3
<?php echo link_to_unless($foo, 'article_show', $article) ?>
```

*Listing  
1-67*

## 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  
1-68*

```
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;
        }
    }
}
```