



symfony: simplify professional web development with PHP

Fabien Potencier, Sensio Labs, fabien.potencier@sensio.com

Overview

A framework streamlines application development by automating many of the patterns employed for a given purpose. A framework also adds structure to the code, prompting the developer to write better, more readable, and more maintainable code. Ultimately, a framework makes programming easier, since it packages complex operations into simple statements.

Symfony is a complete framework designed to optimize the development of web applications. For starters, it separates a web application's business rules, server logic, and presentation views. It contains numerous tools and classes aimed at shortening the development time of a complex web application. Additionally, it automates common tasks so that the developer can focus entirely on the specifics of an application. The end result of these advantages means there is no need to reinvent the wheel every time a new web application is built!

Why create yet another PHP framework?

Back in 2003, we spent some time inquiring about the existing open source development tools for web applications in PHP. We found that none fulfilled the requirements needed to develop professional PHP applications. When PHP 5 was released, we decided that the available tools had reached a mature enough stage to be integrated into a full-featured framework. We subsequently spent a year developing the symfony core, basing our work on the Mojavi Model-View-Controller (MVC) framework, the Propel object-relational mapper (ORM), and the Ruby on Rails templating helpers.

We originally built symfony for Sensio's projects, because having an effective framework at our disposal presents an ideal way to develop applications faster and more efficiently.

It also makes web development more intuitive, and the resulting applications are more robust and easier to maintain. After successfully using symfony for a few projects, we decided to release it under an open source license.

Symfony is based on 9 years of experience and more than 200 websites developed at Sensio. We released symfony as an open source project for a year and a half and it is licensed under the MIT license.

One of the main goals of symfony is to try to bring together the PHP world and the enterprise world. We try to demonstrate that PHP can be used instead of Java to develop corporate websites and Intranets.

Main symfony features

A symfony project

The framework installation

There are a lot of different ways to install symfony. Here are the main ones:

PEAR installation: The easy way

```
$ pear channel-discover pear.symfony-project.com
$ pear install symfony/symfony-1.0.0
```

symfony: simplify professional web development with PHP

Sandbox installation: The fastest way

```
$ curl -O http://www.symfony-project.com/get/sf_sandbox-1.0.0.tgz
$ tar xzpf sf_sandbox-1.0.0.tgz
```

Subversion installation: The recommended way

```
$ svn propedit svn:externals
symfony http://svn.symfony-project.com/branches/1.0
```

Application creation

```
$ mkdir ~/sfdemo
$ cd ~/sfdemo

$ symfony init-project sfdemo
$ ./symfony init-app frontend
```

Here, we use the symfony Command Line Interface to create a new project and a new application within this project.

In symfony, a project can have several applications (a frontend and a backend application). Each application can have several modules (a blog module, a forum module, a shopping cart module). And each module has several actions (list, show). Each action then calls the View system to render a template.

Database configuration

Database configuration is done in some YAML configuration files.

```
# config/databases.yml
prod:
  propel:
    param:
      password: PAssWD
all:
  propel:
    class:      sfPropelDatabase
    param:
      dsn:      mysql://root:@localhost/sfdemo
```

Configuration files in symfony are stored in the YAML format.

Now, let's define our database schema:

```
# config/schema.yml
post:
  title:      { type: varchar, size: 255 }
  content:    { type: longvarchar }
  is_published: { type: boolean }
  author_id:  { type: integer, foreignTable: author, foreignReference: id }
  created_at: ~
```

In symfony, we can create tables by hand but it is better to create a schema.yml configuration file. The schema describes the database tables, columns and relationships between tables. One of the goals of the schema.yml configuration file is to be able to abstract the SQL. So, you can use the same schema and code for all the database engines supported by symfony.

Now, it's time to create some test data:

```
# data/fixtures/data.yml
Author:
  fabien:
    first_name: Fabien
    last_name:  Potencier
Post:
  first_post:
    author_id: fabien
    title:     XTech 2007
```

symfony: simplify professional web development with PHP

To create the model classes, convert the schema into SQL, create the tables and load the test data, we need to launch a symfony task:

```
$ ./symfony propel-build-all-load frontend
```

Backend creation

We will now create a backend interface to interact with our database. Every web project needs a backend interface. This interface must have some simple features like list pagination, list filtering and list sorting. We also need a way to create, read, update and delete objects. Symfony simplifies the creation of such an interface, thanks to the admin generator. The admin generator generates modules that can be integrated within an application. The generated code is fully MVC compliant and fully configurable. We can configure most things in a YAML configuration file but we can also extend the generated actions to override some methods.

```
./symfony propel-init-admin frontend post Post
```

Frontend creation

Now, it's time to create the frontend application. The first thing we have to do is to think about our URLs. We want to provide resources available under a well defined URL structure. To achieve this goal, we have to decouple the URL from the module that will handle the request. In symfony, we have a bi-directional routing system that is able to parse an incoming URL and to create a URL based on some configuration. You never write a hardcoded URL in symfony.

```
homepage:
  param: { module: blog, action: recent }
  url:   /

<?php echo url_for('@homepage') ?>
```

Here, we map / URL to the recent action in the blog module and if we need the URL for the homepage, we can use the symfony helper called `url_for()`. When we call this helper with `@homepage` as an argument, it is converted back to `/`. Now if we want to change the action for the homepage, we just have to change the routing configuration. And if we want the recent posts to have a new URL, we also just have to change the configuration. No need to change our templates.

```
homepage:
  param: { module: blog, action: list }
  url:   /
recent:
  param: { module: blog, action: recent }
  url:   /recent
```

Here is another more complex example with a parameter that must match some requirements.

```
post:
  param: { module: blog, action: show }
  requirements:
    id: \d+
  url:   /blog/:id.html

<?php echo link_to(
  $post->getTitle(),
  '@post?id='.$post->getId()
) ?>
```

Functional tests

Symfony is bundled with a unit test framework called lime and with a functional test framework based on lime. The `sfTestBrowser` object simulates a browser without the HTTP overhead. It calls symfony directly. When we use this browser, we can introspect most of the symfony objects:

```
// test/functional/frontend/blogActionsTest.php
$browser = new sfTestBrowser();
```

symfony: simplify professional web development with PHP

```
$browser->initialize();
$browser->
  get('/blog/1.html')->
  isStatusCode(200)->
  checkResponseElement('h1.title', '/XTech 2007/');
```

The `checkResponseElement()` method takes a CSS selector as its first argument. When using the `sfTestBrowser` object, we can also tests Ajax requests, cache, security, routing, ...

Then, we can call the test-functional task of the symfony CLI. As you can see, our tests fail.

```
$ ./symfony test-functional frontend
# get /
ok 1 - status code is 200
not ok 2 - response selector h1 does not match regex /XTech 2007/
# Looks like you failed 1 tests of 2
1..2
```

Our first line of code

We create a new module with a simple `executeShow()` method. This method is the Controller part of the MVC pattern.

```
class blogActions extends sfActions
{
  function executeShow()
  {
    $id = $this->getRequestParameter('id');
    $this->post = PostPeer::retrieveByPk($id);
    $this->forward404Unless($this->post);
  }
}
```

The template is very simple:

```
# apps/frontend/modules/post/templates/showSuccess.php
<h1 class="title"><?php echo $post->getTitle() ?></h1>
<h2>par <?php echo $post->getAuthor()->getFullName() ?></h2>
<p><?php echo $post->getHtmlContent(ESC_RAW) ?></p>
```

This template also demonstrates the XSS security layer. Every variable and every method call is escaped by default, so the title and the author name are escaped. But as the content of the post is stored in HTML, we don't want to escape it.

Deployment

As our website is now finished, it's time to deploy it to the production servers:

```
$ ./symfony test-all
functional/frontend/postActionsTest.....ok
All tests successful.
Files=1, Tests=2

# config/properties.ini
[production]
host=1.2.3.4
user=fabien
dir=/var/www/sfblog
type=rsync

$ ./symfony sync production go
```

The community

If the first two strengths of symfony are the quality of the code base and the quality of the documentation, the third would have to be the community.

symfony: simplify professional web development with PHP

It started like any open source project. There were early adopters who were quite happy with the code and wanted to share it but really, it was limited in the beginning. As time went on, more and more people came, mostly as we released more documentation, screencasts, and tutorials.

Once we reached critical mass, the community started feeding itself. Community members started answering questions posted on the mailing list and in the forums and they started building applications on top of symfony. This allows us to focus on the code and documentation. The community is what keeps symfony alive. A vibrant and contributing community means that the project is alive and will continue to live for a long time.

Bibliography

[POT07] F. Zaninotto & F. Potencier, *The Definitive Guide to symfony*, Apress, 2007.

[SYMFONY] <http://www.symfony-project.com/>

[SENSIO] <http://www.sensio-labs.com/>