Getting Started

symfony 1.2

This PDF is brought to you by

Version: getting-started-1.2-en-2009-10-11

Table of Contents

Chapter 1

Introduction

The symfony framework is a full-stack MVC framework that helps you develop websites faster. It also establishes a set of best practices that will help you to develop maintainable and secure websites. And advocating best practices starts as soon as you want to install the framework itself.

Installing symfony is not very much different to installing any other PHP software but, to make your installation secure from the start, you should not just put all the files under your web root directory as many other installation guides may prescribe. Although it will take slightly more time to install symfony our way, a little extra effort at the beginning is really worth it in the long run. Also, as with any other PHP software installation, there are a lot of small traps that you can fall into that can make your experience harder than it needs to be, so we will try to help you avoid them.

This tutorial teaches you everything you need to get started with a new symfony project. From the web server configuration, installation of symfony itself, to the creation of an application; at the end of the tutorial, you will have a fully-working symfony application, ready to be used for your next project.

Chapter 2

Prerequisites

Before installing symfony, you need to check that your computer has everything installed and configured correctly. Take the time to conscientiously read this chapter and follow all the steps required to check your configuration, as it may save your day further down the road.

Software

First of all, you need to check that your computer has a friendly working environment for web development. At a minimum, you need a web server (Apache, for instance), a database engine (MySQL, PostgreSQL, SQLite, or any PDO[1]-compatible database engine), and PHP 5.2.4 or later.

Command Line Interface

The symfony framework comes bundled with a command line tool that automates a lot of work for you. If you are a Unix-like OS user, you will feel right at home. If you run a Windows system, it will also work fine, but you will just have to type a few commands at the `cmd` prompt.

Unix shell commands can come in handy in a Windows environment. If you would like to use tools like `tar`, `gzip` or `grep` on Windows, you can install Cygwin[2]. The official docs are a little sparse, so a good installation guide can be found here[3]. The adventurous may also like to try Microsoft's Windows Services for Unix[4].

PHP Configuration

As PHP configurations can vary a lot from one OS to another, or even between different Linux distributions, you need to check that your PHP configuration meets the symfony minimum requirements.

First, ensure that you have PHP 5.2.4 at a minimum installed by using the `phpinfo()` built-in function or by running `php -v` on the command line. Be aware that on some configurations,

1. `http://www.php.net/PDO`
2. `http://cygwin.com/`
3. `http://www.soe.ucsc.edu/~you/notes/cygwin-install.html`
4. `http://technet.microsoft.com/en-gb/interopmigration/bb380242.aspx`

you might have two different PHP versions installed: one for the command line, and another for the web.

Then, download the symfony configuration checker script at the following URL:

```
http://sf-to.org/1.2/check.php
```

Listing 2-1

Save the script somewhere under your current web root directory.

Launch the configuration checker script from the command line:

```
$ php check_configuration.php
```

Listing 2-2

If there is a problem with your PHP configuration, the output of the command will give you hints on what to fix and how to fix it.

You should also execute the checker from a browser and fix the issues it might discover. That's because PHP can have a distinct `php.ini` configuration file for these two environments, with different settings.

Don't forget to remove the file from your web root directory afterwards.

Chapter 3

The Sandbox

If your goal is to give symfony a try for a few hours, keep reading this chapter as we will show you the fastest way to get you started. If you want to bootstrap a real world project, you can safely skip this chapter, and jump[5] to the next one right away.

The fastest way to experiment with symfony is to install the symfony sandbox. The sandbox is a dead-easy-to-install pre-packaged symfony project, already configured with some sensible defaults. It is a great way to practice using symfony without the hassle of a proper installation that respects the web best practices.

As the sandbox is pre-configured to use SQLite as a database engine, you need to check that your PHP supports SQLite (see the Prerequisites[6] chapter). You can also read the Configuring the Database[7] section to learn how to change the database used by the sandbox.

You can download the symfony sandbox in `.tgz` or `.zip` format from the symfony installation page[8] or at the following URLs:

Listing 3-1

```
http://www.symfony-project.org/get/sf_sandbox_1_2.tgz

http://www.symfony-project.org/get/sf_sandbox_1_2.zip
```

Un-archive the files somewhere under your web root directory, and you are done. Your symfony project is now accessible by requesting the `web/index.php` script from a browser.

Having all the symfony files under the web root directory is fine for testing symfony on your local computer, but is a really bad idea for a production server as it potentially makes all the internals of your application visible to end users.

You can now finish your installation by reading the Web Server Configuration[9] and the Environments[10] chapters.

As a sandbox is just a normal symfony project where some tasks have been executed for you and some configuration changed, it is quite easy to use it as a starting point for a new project. However, keep in mind that you will probably need to adapt the configuration; for

5. `04-Symfony-Installation#chapter_04`
6. `02-Prerequisites#chapter_02`
7. `05-Project-Setup#chapter_05_sub_configuring_the_database`
8. `http://www.symfony-project.org/installation/1_2`
9. `06-Web-Server-Configuration#chapter_06`
10. `07-Environments#chapter_07`

instance changing the security related settings (see the configuration of XSS and CSRF later in this tutorial).

Chapter 4

Symfony Installation

Project Directory

Before installing symfony, you first need to create a directory that will host all the files related to your project:

Listing 4-1
```
$ mkdir -p /home/sfproject
$ cd /home/sfproject
```

Or on Windows:

Listing 4-2
```
c:\> mkdir c:\dev\sfproject
c:\> cd c:\dev\sfproject
```

Windows users are advised to run symfony and to setup their new project in a path which contains no spaces. Avoid using the `Documents and Settings` directory, including anywhere under `My Documents`.

If you create the symfony project directory under the web root directory, you won't need to configure your web server. Of course, for production environments, we strongly advise you to configure your web server as explained in the web server configuration section.

Symfony Installation

Create a directory to host the symfony framework library files:

Listing 4-3
```
$ mkdir -p lib/vendor
```

Now, you need to install symfony. As the symfony framework has several stable branches, you need to choose the one you want to install by reading the installation page[11] on the symfony website.

Go to the installation page for the version you have just chosen, symfony 1.2[12] for instance.

Under the "**Source Download**" section, you will find the archive in `.tgz` or in `.zip` format. Download the archive, put it under the freshly created `lib/vendor/` directory and un-archive it:

11. `http://www.symfony-project.org/installation`
12. `http://www.symfony-project.org/installation/1_2`

```
$ cd lib/vendor
$ tar zxpf symfony-1.2.2.tgz
$ mv symfony-1.2.2 symfony
$ rm symfony-1.2.2.tgz
```

Listing 4-4

Under Windows, unzipping the zip file can be achieved using Windows Explorer. After you rename the directory to `symfony`, there should be a directory structure similar to `c:\dev\sfproject\lib\vendor\symfony`.

If you use Subversion, it is even better to use the `svn:externals` property to embed symfony into your project in the `lib/vendor/` directory, which benefits from the bug fixes made in the stable branch automatically:

```
http://svn.symfony-project.com/branches/1.2/
```

Listing 4-5

Check that symfony is correctly installed by using the symfony command line to display the symfony version (note the capital `V`):

```
$ cd ../..
$ php lib/vendor/symfony/data/bin/symfony -V
```

Listing 4-6

On Windows:

```
c:\> cd ..\..
c:\> php lib\vendor\symfony\data\bin\symfony -V
```

Listing 4-7

If you are curious about what this command line tool can do for you, type `symfony` to list the available options and tasks:

```
$ php lib/vendor/symfony/data/bin/symfony
```

Listing 4-8

On Windows:

```
c:\> php lib\vendor\symfony\data\bin\symfony
```

Listing 4-9

The symfony command line is the developer's best friend. It provides a lot of utilities that improve your productivity for day-to-day activities like cleaning the cache, generating code, and much more.

Chapter 5

Project Setup

In symfony, **applications** sharing the same data model are regrouped into **projects**. For most projects, you will have two different applications: a frontend and a backend.

Project Creation

From the `sfproject/` directory, run the symfony `generate:project` task to actually create the symfony project:

Listing 5-1
```
$ php lib/vendor/symfony/data/bin/symfony generate:project PROJECT_NAME
```

On Windows:

Listing 5-2
```
c:\> php lib\vendor\symfony\data\bin\symfony generate:project PROJECT_NAME
```

The `generate:project` task generates the default structure of directories and files needed for a symfony project:

Directory	Description
`apps/`	Hosts all project applications
`cache/`	The files cached by the framework
`config/`	The project configuration files
`lib/`	The project libraries and classes
`log/`	The framework log files
`plugins/`	The installed plugins
`test/`	The unit and functional test files
`web/`	The web root directory (see below)

Why does symfony generate so many files? One of the main benefits of using a full-stack framework is to standardize your developments. Thanks to symfony's default structure of files and directories, any developer with some symfony knowledge can take over the maintenance of any symfony project. In a matter of minutes, he will be able to dive into the code, fix bugs, and add new features.

The `generate:project` task has also created a `symfony` shortcut in the project root directory to shorten the number of characters you have to write when running a task.

So, from now on, instead of using the fully qualified path to the symfony program, you can use the `symfony` shortcut.

Application Creation

Now, create the frontend application by running the `generate:app` task:

```
$ php symfony generate:app --escaping-strategy=on
      --csrf-secret=UniqueSecret frontend
```

Listing 5-3

Because the symfony shortcut file is executable, Unix users can replace all occurrences of '`php symfony`' by '`./symfony`' from now on.

On Windows you can copy the '`symfony.bat`' file to your project and use '`symfony`' instead of '`php symfony`':

```
c:\> copy lib\vendor\symfony\data\bin\symfony.bat .
```

Listing 5-4

Based on the application name given as an *argument*, the `generate:app` task creates the default directory structure needed for the application under the `apps/frontend/` directory:

Directory	Description
`config/`	The application configuration files
`lib/`	The application libraries and classes
`modules/`	The application code (MVC)
`templates/`	The global template files

When calling the `generate:app` task, you have also passed two security-related *options*:

- `--escaping-strategy`: Enables output escaping to prevent XSS attacks
- `--csrf-secret`: Enables session tokens in forms to prevent CSRF attacks

By passing these two optional options to the task, you have secured your future development from the two most widespread vulnerabilities found on the web. That's right, symfony will automatically take security measures on your behalf.

If you know nothing about XSS[13] or CSRF[14], take the time to learn more about these security vulnerabilities.

Directory Structure Rights

Before trying to access your newly created project, you need to set the write permissions on the `cache/` and `log/` directories to the appropriate levels, so that your web server can write to them:

```
$ chmod 777 cache/ log/
```

Listing 5-5

Tips for People using a SCM Tool

symfony only ever writes in two directories of a symfony project, `cache/` and `log/`. The content of these directories should be ignored by your SCM (by editing the `svn:ignore` property if you use Subversion for instance).

13. `http://en.wikipedia.org/wiki/Cross-site_scripting`
14. `http://en.wikipedia.org/wiki/CSRF`

The symfony Path

You can get the symfony version used by your project by typing:

Listing 5-6
```
$ php symfony -V
```

The `-V` option also displays the path to the symfony installation directory, which is stored in `config/ProjectConfiguration.class.php`:

Listing 5-7
```
// config/ProjectConfiguration.class.php
require_once '/Users/fabien/symfony-1.2/lib/autoload/
sfCoreAutoload.class.php';
```

For better portability, change the absolute path to the symfony installation to a relative one:

Listing 5-8
```
// config/ProjectConfiguration.class.php
require_once dirname(__FILE__).'/../lib/vendor/symfony/lib/autoload/
sfCoreAutoload.class.php';
```

That way, you can move the project directory anywhere on your machine or another one, and it will just work.

Configuring the Database

One of the first things you might want to do is to configure the database connection for your project. The symfony framework supports all PDO[15]-supported databases (MySQL, PostgreSQL, SQLite, Oracle, MSSQL, ...). On top of PDO, symfony comes bundled with two ORM tools: Propel and Doctrine. Propel is the default one, but switching to Doctrine is quite easy (see the next section for more information).

Configuring the database is as simple as using the `configure:database` task:

Listing 5-9
```
$ php symfony configure:database "mysql:host=localhost;dbname=dbname" root
mYsEcret
```

The `configure:database` task takes three arguments: the PDO DSN[16], the username, and the password to access the database. If you don't need a password to access your database on the development server, just omit the third argument.

Switching to Doctrine

If you decide to use Doctrine instead of Propel, you need to first enable `sfDoctrinePlugin` and disable `sfPropelPlugin`. This can be done by changing the following code in your `config/ProjectConfiguration.class.php`:

Listing 5-10
```
public function setup()
{
  $this->enableAllPluginsExcept(array('sfPropelPlugin',
'sfCompat10Plugin'));
}
```

After making these changes, launch these commands:

Listing 5-11

15. http://www.php.net/PDO
16. http://www.php.net/manual/en/pdo.drivers.php

```
$ php symfony plugin:publish-assets
$ php symfony cc
$ rm web/sfPropelPlugin
$ rm config/propel.ini
$ rm config/schema.yml
$ rm config/databases.yml
```

Then, run the following command to configure your database for Doctrine:

Listing 5-12

```
$ php symfony configure:database --name=doctrine
--class=sfDoctrineDatabase "mysql:host=localhost;dbname=jobeet" root
mYsEcret
```

Chapter 6

Web Server Configuration

The ugly Way

In the previous chapters, you have created a directory that hosts the project. If you have created it somewhere under the web root directory of your web server, you can already access the project in a web browser.

Of course, as there is no configuration, it is very fast to set up, but try to access the `config/databases.yml` file in your browser to understand the bad consequences of such a lazy attitude. If the user knows that your website is developed with symfony, he will have access to a lot of sensitive files.

Never ever use this setup on a production server, and read the next section to learn how to configure your web server properly.

The secure Way

A good web practice is to put under the web root directory only the files that need to be accessed by a web browser, like stylesheets, JavaScripts and images. By default, we recommend to store these files under the `web/` sub-directory of a symfony project.

If you have a look at this directory, you will find some sub-directories for web assets (`css/` and `images/`) and the two front controller files. The front controllers are the only PHP files that need to be under the web root directory. All other PHP files can be hidden from the browser, which is a good idea as far as security is concerned.

Web Server Configuration

Now it is time to change your Apache configuration, to make the new project accessible to the world.

Locate and open the `httpd.conf` configuration file and add the following configuration at the end:

Listing 6-1

```
# Be sure to only have this line once in your configuration
NameVirtualHost 127.0.0.1:8080

# This is the configuration for your project
Listen 127.0.0.1:8080

<VirtualHost 127.0.0.1:8080>
```

```
  DocumentRoot "/home/sfproject/web"
  DirectoryIndex index.php
  <Directory "/home/sfproject/web">
    AllowOverride All
    Allow from All
  </Directory>

  Alias /sf /home/sfproject/lib/vendor/symfony/data/web/sf
  <Directory "/home/sfproject/lib/vendor/symfony/data/web/sf">
    AllowOverride All
    Allow from All
  </Directory>
</VirtualHost>
```

The `/sf` alias gives you access to images and javascript files needed to properly display default symfony pages and the web debug toolbar.

On Windows, you need to replace the `Alias` line with something like:

```
Alias /sf "c:\dev\sfproject\lib\vendor\symfony\data\web\sf"
```

Listing 6-2

And `/home/sfproject/web` should be replaced with:

```
c:\dev\sfproject\web
```

Listing 6-3

This configuration makes Apache listen to port `8080` on your machine, so the website will be accessible at the following URL:

```
http://localhost:8080/
```

Listing 6-4

You can change `8080` to any number, but favour numbers greater than `1024` as they do not require administrator rights.

Configure a dedicated Domain Name

If you are an administrator on your machine, it is better to setup virtual hosts instead of adding a new port each time you start a new project. Instead of choosing a port and add a `Listen` statement, choose a domain name and add a `ServerName` statement:

```
# This is the configuration for your project
<VirtualHost 127.0.0.1:80>
  ServerName sfproject.localhost
  <!-- same configuration as before -->
</VirtualHost>
```

Listing 6-5

The domain name `sfproject.localhost` used in the Apache configuration has to be declared locally. If you run a Linux system, it has to be done in the `/etc/hosts` file. If you run Windows XP, this file is located in the `C:\WINDOWS\system32\drivers\etc\` directory.

Add in the following line:

```
127.0.0.1 sfproject.localhost
```

Listing 6-6

Test the New Configuration

Restart Apache, and check that you now have access to the new application by opening a browser and typing `http://localhost:8080/index.php/`, or `http://sfproject.localhost/index.php/` depending on the Apache configuration you chose in the previous section.

If you have the Apache `mod_rewrite` module installed, you can remove the `index.php/` part of the URL. This is possible thanks to the rewriting rules configured in the `web/.htaccess` file.

You should also try to access the application in the development environment (see the next section for more information about environments). Type in the following URL:

Listing 6-7

```
http://sfproject.localhost/frontend_dev.php/
```

The web debug toolbar should show in the top right corner, including small icons proving that your `sf/` alias configuration is correct.

The setup is a little different if you want to run symfony on an IIS server in a Windows environment. Find how to configure it in the related tutorial[17].

17. `http://www.symfony-project.com/cookbook/1_0/web_server_iis`

Chapter 7

The Environments

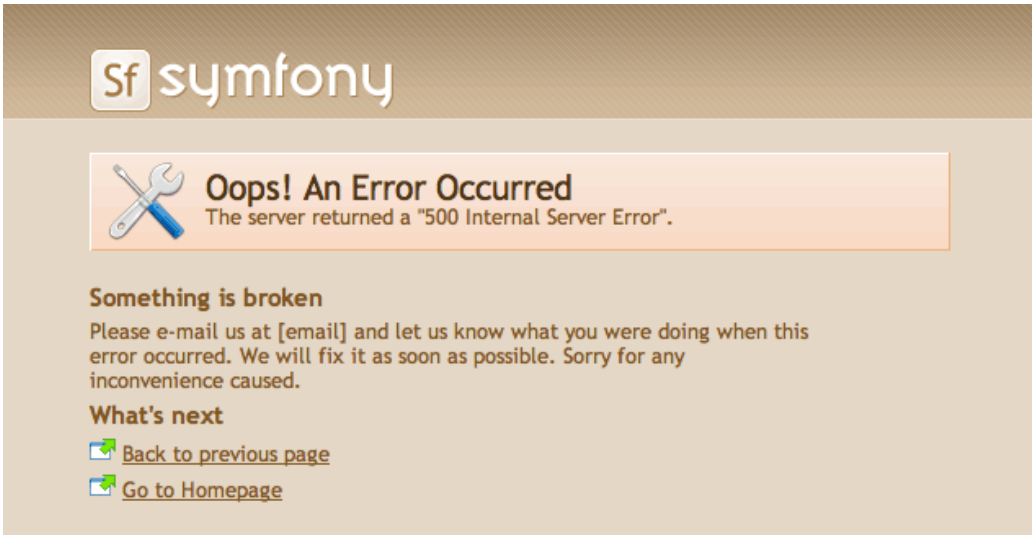
If you have a look at the `web/` directory, you will find two PHP files: `index.php` and `frontend_dev.php`. These files are called **front controllers**; all requests to the application are made through them. But why do we have two front controllers for each application?

Both files point to the same application but for different **environments**. When you develop an application, except if you develop directly on the production server, you need several environments:

- The **development environment**: This is the environment used by **web developers** when they work on the application to add new features, fix bugs, …
- The **test environment**: This environment is used to automatically test the application.
- The **staging environment**: This environment is used by the **customer** to test the application and report bugs or missing features.
- The **production environment**: This is the environment **end users** interact with.

What makes an environment unique? In the development environment for instance, the application needs to log all the details of a request to ease debugging, but the cache system must be disabled as all changes made to the code must be taken into account right away. So, the development environment must be optimized for the developer. The best example is certainly when an exception occurs. To help the developer debug the issue faster, symfony displays the exception with all the information it has about the current request right into the browser:

But on the production environment, the cache layer must be activated and, of course, the application must display customized error messages instead of raw exceptions. So, the production environment must be optimized for performance and the user experience.



If you open the front controller files, you will see that their content is the same except for the environment setting:

Listing 7-1

```
// web/index.php
<?php

require_once(dirname(__FILE__).'/../config/
ProjectConfiguration.class.php');

$configuration =
ProjectConfiguration::getApplicationConfiguration('frontend', 'prod',
false);
sfContext::createInstance($configuration)->dispatch();
```

The web debug toolbar is also a great example of the usage of environment. It is present on all pages in the development environment and gives you access to a lot of information by clicking on the different tabs: the current application configuration, the logs for the current request, the SQL statements executed on the database engine, memory information, and time information.

Chapter 8

What's next?

If you have followed the instructions from the previous chapters, you should now have a fully-functional symfony project, and are ready to experiment with symfony.

You can start reading more documentation on the main documentation page[18] of your version.

On this page, you will find the Jobeet tutorial, which is probably the best way to learn symfony. It explains in great detail the development of a web application from start to finish, and also teaches you the best practices of a web development. You can also buy it as a printed book.

The symfony framework has a lot of great features and a lot of free documentation. That said, one of the most valuable assets an Open-Source project can have is its community, and symfony has one of the most active and friendly communities around. If you start using symfony for your projects, consider joining the symfony community:

- Subscribe to the user mailing-list[19]
- Subscribe to the official blog feed[20]
- Subscribe to the symfony planet feed[21]
- Come and chat on the #symfony IRC[22] channel on freenode

18. http://www.symfony-project.org/doc/1_2/
19. http://groups.google.com/group/symfony-users
20. http://feeds.feedburner.com/symfony/blog
21. http://feeds.feedburner.com/symfony/planet
22. irc://irc.freenode.net/symfony