Symfony2 Essentials

Symfony is a free and open source PHP MVC web application development framework, which helps you create and maintain web applications and replace recurring coding tasks. Its well-organized structure, clean code, and good programming practices make web development a breeze.

Symfony2 Essentials will guide you through the process of creating a sample web application with Symfony2. You will create a to-do application using a few of the most commonly used Symfony2 components, and discover how to perform these development tasks efficiently. This book introduces you to the Symfony framework with a quick installation guide and a brief explanation of its key features. By the end of this book, you will have learned how to combine a Symfony2 framework with other open source code to speed up the development process.

Who this book is written for

This book is aimed at experienced programmers, especially those familiar with a closely related technology such as Yii or Laravel, but who now want to learn Symfony quickly. This book will also prove beneficial for experienced PHP developers who want to explore and evaluate new frameworks and their possibilities in day-to-day tasks.

What you will learn from this book

- Familiarize yourself with the Symfony framework, its latest features, and how to install it
- Discover the concept of bundles and their application
- Handle translations within Symfony, enable translations, and learn how to handle database translations
- Understand Symfony’s security model, how to secure applications, and implement a custom authentication provider by using FOSUserBundle
- Explore the use of Twig, find the best practices of using it, and discover its common pitfalls
- Create internal commands that will handle sending e-mail reminders for your app
- Develop a plugin for the profiler to provide custom information about your application
- Deploy applications based on Symfony2 using various methods including simple FTP copying, rsync, CI deployment, and more

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'The Symfony Framework – Installation and Configuration'
- A synopsis of the book’s content
- More information on Symfony2 Essentials
About the Author

**Wojciech Bancer** has a master's degree in computer science. He has over 10 years of experience in web application development. In 2007, after passing the Zend exam, he received a Zend Certified Engineer for PHP5 certificate. He started his career as a freelancer and consultant by developing web applications in PHP 4 and PHP 5. He has led many IT projects for clients in Europe and USA. Currently, Wojciech is a managing partner of a software organization and is in charge of the R&D structure of one of the fastest growing iBeacon projects in Europe.
PHP is currently one of the most popular languages in web development. Over time, the language itself has grown and become more mature. However, it still lacks good design patterns and good habits "by design". You can write a very good code with it, but you can also write very bad "spaghetti code".

Symfony2 is currently one of the most popular frameworks to speed up PHP development. It greatly helps you to create clean and reusable code, and it is the first framework that takes modern design patterns like DependencyInjection very seriously. It is also the first framework that uses features of PHP 5.3 — namespaces, closures, and so on. By using this framework, you will be able to deliver more advanced and complicated web applications, suitable even for enterprise requirements.

**What this book covers**

*Chapter 1, The Symfony Framework – Installation and Configuration*, gives a quick overview of Symfony's history, ways to install and configure it, and ways to use the composer — the dependency manager used within Symfony2.

*Chapter 2, Your First Pages*, gives you a quick walkthrough over the basic MVC features available in most solutions nowadays. It will demonstrate to you how to create simple controllers and first views, prepare entities, and load sample data.

*Chapter 3, Twig Templating and Assets Handling*, shows you in details how to handle views, assets managements, and add a frontend library.

*Chapter 4, Forms*, focuses on form creation. It will show you how to create form types, validate forms, and write data to databases.
Chapter 5, *Security and Handling Users*, gives you information about Symfony security features. In this chapter, we will create an example of registration and login forms, and how to handle users with the help of an open source bundle.

Chapter 6, *Translation*, gives information about translation and internalization. You will learn how to create translation files, translate various areas of your application, and choose a good translation strategy.

Chapter 7, *AJAX*, teaches you how to handle basic AJAX calls. We will also learn about bundles that are useful when you write RESTful applications, how to handle various HTTP methods, and how to serialize data in a JSON structure.

Chapter 8, *Command-line Operations*, will provide you with examples on how to create a command-line task, which is often useful with tasks executed in background (cron tasks). You will also learn how to send an e-mail through swiftmailer—a component used by default within the Symfony framework to handle e-mail sending.

Chapter 9, *Symfony2 Profiler and Debugger*, is where you will learn about the Web Debug Toolbar and profiler. We will examine the possibilities offered by these tools and write some code to demonstrate debugging and profiling features.

Chapter 10, *Preparing an Application for Production*, will give you an overview of the tasks that are usually done when your project needs to be deployed. You will also get an overview of various deployment strategies, and you will be able to check your framework and bundles against known security issues.
The Symfony Framework – Installation and Configuration

The Symfony framework is currently one of the most popular PHP frameworks existing within the PHP developer’s environment. Version 2, which was released a few years ago, has been a great improvement, and in my opinion was one of the key elements for making the PHP ecosystem suitable for larger enterprise projects. The framework version 2.0 not only required the modern PHP version (minimal version required for Symfony is PHP 5.3.8), but also uses state-of-the-art technology — namespaces and anonymous functions. Authors also put a lot of efforts to provide long term support and to minimize changes, which break the compatibility between versions. Also, Symfony forced developers to use a few useful design concepts. The key one, introduced in Symfony, was DependencyInjection.

In most cases, the book will refer to the framework as Symfony2. If you want to look over the Internet or Google about this framework, apart from using Symfony keyword you may also try to use the Symfony2 keyword This was the way recommended some time ago by one of the creators to make searching or referencing to the specific framework version easier in future.

Key reasons to choose Symfony2

Symfony2 is recognized in the PHP ecosystem as a very well-written and well-maintained framework. Design patterns that are recommended and forced within the framework allow work to be more efficient in the group, this allows better tests and the creation of reusable code.
Symfony’s knowledge can also be verified through a certificate system, and this allows its developers to be easily found and be more recognized on the market. Last but not least, the Symfony2 components are used as parts of other projects, for example, look at the following:

- Drupal
- phpBB
- Laravel
- eZ Publish and more

Over time, there is a good chance that you will find the parts of the Symfony2 components within other open source solutions.

Bundles and extendable architecture are also some of the key Symfony2 features.

They not only allow you to make your work easier through the easy development of reusable code, but also allows you to find smaller or larger pieces of code that you can embed and use within your project to speed up and make your work faster.

The standards of Symfony2 also make it easier to catch errors and to write high-quality code; its community is growing every year.

The history of Symfony

There are many Symfony versions around, and it’s good to know the differences between them to learn how the framework was evolving during these years.

The first stable Symfony version — 1.0 — was released in the beginning of 2007 and was supported for three years. In mid-2008, version 1.1 was presented, which wasn’t compatible with the previous release, and it was difficult to upgrade any old project to this.

Symfony 1.2 version was released shortly after this, at the end of 2008. Migrating between these versions was much easier, and there were no dramatic changes in the structure. The final versions of Symfony 1’s legacy family was released nearly one year later. Simultaneously, there were two version releases, 1.3 and 1.4. Both were identical, but Symfony 1.4 did not have deprecated features, and it was recommended to start new projects with it. Version 1.4 had 3 years of support.

If you look into the code, version 1.x was very different from version 2. The company that was behind Symfony (the French company, SensioLabs) made a bold move and decided to rewrite the whole framework from scratch.
The first release of Symfony2 wasn't perfect, but it was very promising. It relied on Git submodules (the composer did not exist back then). The 2.1 and 2.2 versions were closer to the one we use now, although it required a lot of effort to migrate to the upper level. Finally, the Symfony 2.3 was released — the first long-term support version within the 2.x branch. After this version, the changes provided within the next major versions (2.4, 2.5, and 2.6) are not so drastic and usually they do not break compatibility.

This book was written based on the latest stable Symfony 2.7.4 version and was tested with PHP 5.5). This Symfony version is marked as the so called long-term support version, and updates for it will be released for 3 years since the first 2.7 version release.

**Installation**

Prior to installing Symfony2, you don't need to have a configured web server. If you have at least PHP version 5.4, you can use the standalone server provided by Symfony2. This server is suitable for development purposes and should not be used for production. It is strongly recommend to work with a Linux/UNIX system for both development and production deployment of Symfony2 framework applications. While it is possible to install and operate on a Windows box, due to its different nature, working with Windows can sometimes force you to maintain a separate fragment of code for this system.

Even if your primary OS is Windows, it is strongly recommended to configure Linux system in a virtual environment. Also, there are solutions that will help you in automating the whole process. As an example, see more on https://www.vagrantup.com/ website.

To install Symfony2, you can use a few methods as follows:

- Use a new Symfony2 installer script (currently, the only officially recommended). Please note that installer requires at least PHP 5.4.
- Use a composer dependency manager to install a Symfony project.
- Download a zip or tgz package and unpack it.
- It does not really matter which method you choose, as they all give you similar results.
Installing Symfony2 using an installer

To install Symfony2 through an installer, go to the Symfony website at http://symfony.com/download, and install the Symfony2 installer by issuing the following commands:

```shell
$ sudo curl -LsS http://symfony.com/installer -o /usr/local/bin/symfony
$ sudo chmod +x /usr/local/bin/symfony
```

After this, you can install Symfony by just typing the following command:

```shell
$ symfony new <new_project_folder>
```

To install the Symfony2 framework for a to-do application, execute the following command:

```shell
$ symfony new todoapp
```

This command installs the latest Symfony2 stable version on the newly created todoapp folder, creates the Symfony2 application, and prepares some basic structure for you to work with.

After the app creation, you can verify that your local PHP is properly configured for Symfony2 by typing the following command:

```shell
$ php app/check.php
```

If everything goes fine, the script should complete with the following message:

```
[OK]
Your system is ready to run Symfony projects
```

Symfony2 is equipped with a standalone server. It makes development easier. If you want to run this, type the following command:

```shell
$ php app/console server:run
```

If everything went alright, you will see a message that your server is working on the IP 127.0.0.1 and port 8000. If there is an error, make sure you are not running anything else that is listening on port 8000. It is also possible to run the server on a different port or IP, if you have such a requirement, by adding the address and port as a parameter, that is:

```shell
$ php app/console server:run 127.0.0.1:8080
```

If everything works, you can now type the following:

http://127.0.0.1:8000/
Now, you will visit Symfony's welcome page.

This page presents you with a nice welcome information and useful documentation link.

**Downloading the example code**

You can download the example code files for all Packt books you have purchased from your account at http://www.packtpub.com. If you have purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files directly e-mailed to you.

---

### The Symfony2 directory structure

Let's dive in to the initial directory structure within the typical Symfony application. Here it is:

- **app**
- **bin**
- **src**
- **vendor**
- **web**

While Symfony2 is very flexible in terms of directory structure, it is recommended to keep the basic structure mentioned earlier.

The following table describes their purpose:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Used for</th>
</tr>
</thead>
<tbody>
<tr>
<td>app</td>
<td>This holds information about general configuration, routing, security configuration, database parameters, and many others. It is also the recommended place for putting new view files. This directory is a starting point.</td>
</tr>
<tr>
<td>bin</td>
<td>It holds some helper executables. It is not really important during the development process, and rarely modified.</td>
</tr>
<tr>
<td>src</td>
<td>This directory holds the project PHP code (usually your bundles).</td>
</tr>
</tbody>
</table>
The Symfony Framework – Installation and Configuration

<table>
<thead>
<tr>
<th>Directory</th>
<th>Used for</th>
</tr>
</thead>
<tbody>
<tr>
<td>vendor</td>
<td>These are third-party libraries used within the project. Usually, this directory contains all the open source third-party bundles, libraries, and other resources. It's worth to mention that it's recommended to keep the files within this directory outside the versioning system. It means that you should not modify them under any circumstances. Fortunately, there are ways to modify the code, if it suits your needs more. This will be demonstrated when we implement user management within our to-do application.</td>
</tr>
<tr>
<td>web</td>
<td>This is the directory that is accessible through the web server. It holds the main entry point to the application (usually the app.php and app_dev.php files), CSS files, JavaScript files, and all the files that need to be available through the web server (user uploadable files).</td>
</tr>
</tbody>
</table>

So, in most cases, you will be usually modifying and creating the PHP files within the src/ directory, the view and configuration files within the app/ directory, and the JS/CSS files within the web/ directory.

The main directory also holds a few files as follows:

- .gitignore
- README.md
- composer.json
- composer.lock

The .gitignore file's purpose is to provide some preconfigured settings for the Git repository, while the composer.json and composer.lock files are the files used by the composer dependency manager.

What is a bundle?

Within the Symfony2 application, you will be using the "bundle" term quite often. Bundle is something similar to plugins. So it can literally hold any code controllers, views, models, and services. A bundle can integrate other non-Symfony2 libraries and hold some JavaScript/CSS code as well. We can say that almost everything is a bundle in Symfony2; even some of the core framework features together form a bundle. A bundle usually implements a single feature or functionality. The code you are writing when you write a Symfony2 application is also a bundle.

There are two types of bundles. The first kind of bundle is the one you write within the application, which is project-specific and not reusable. For this purpose, there is a special bundle called AppBundle created for you when you install the Symfony2 project.
Also, there are reusable bundles that are shared across the various projects either written by you, your team, or provided by a third-party vendors. Your own bundles are usually stored within the src/ directory, while the third-party bundles sit within the vendor/ directory.

The vendor directory is used to store third-party libraries and is managed by the composer. As such, it should never be modified by you.

There are many reusable open source bundles, which help you to implement various features within the application. You can find many of them to help you with User Management, writing RESTful APIs, making better documentation, connecting to Facebook and AWS, and even generating a whole admin panel. There are tons of bundles, and everyday brings new ones.

If you want to explore open source bundles, and want to look around what's available, I recommend you to start with the http://knpbundles.com/ website.

The bundle name is correlated with the PHP namespace. As such, it needs to follow some technical rules, and it needs to end with the Bundle suffix. A few examples of correct names are AppBundle and AcmeDemoBundle, CompanyBlogBundle or CompanySocialForumBundle, and so on.

### Composer

Symfony2 is built based on components, and it would be very difficult to manage the dependencies between them and the framework without a dependency manager. To make installing and managing these components easier, Symfony2 uses a manager called composer.

You can get it from the https://getcomposer.org/ website. The composer makes it easy to install and check all dependencies, download them, and integrate them to your work. If you want to find additional packages that can be installed with the composer, you should visit https://packagist.org/. This site is the main composer repository, and contains information about most of the packages that are installable with the composer.

To install the composer, go to https://getcomposer.org/download/ and see the download instruction. The download instruction should be similar to the following:

```
$ curl -sS https://getcomposer.org/installer | php
```
If the download was successful, you should see the `composer.phar` file in your directory. Move this to the project location in the same place where you have the `composer.json` and `composer.lock` files. You can also install it globally, if you prefer to, with these two commands:

```
$ curl -sS https://getcomposer.org/installer | php
$ sudo mv composer.phar /usr/local/bin/composer
```

You will usually need to use only three composer commands: `require`, `install`, and `update`.

The `require` command is executed when you need to add a new dependency. The `install` command is used to install the package. The `update` command is used when you need to fetch the latest version of your dependencies as specified within the JSON file.

The difference between `install` and `update` is subtle, but very important. If you are executing the `update` command, your `composer.lock` file gets updated with the version of the code you just fetched and downloaded. The `install` command uses the information stored in the `composer.lock` file and the fetch version stored in this file.

When to use `install`? For example, if you deploy the code to the server, you should use `install` rather than `update`, as it will deploy the version of the code stored in `composer.lock`, rather than download the latest version (which may be untested by you). Also, if you work in a team and you just got an update through Git, you should use `install` to fetch the vendor code updated by other developers.

You should use the `update` command if you want to check whether there is an updated version of the package you have installed, that is, whether a new minor version of Symfony2 will be released, then the `update` command will fetch everything.

As an example, let’s install one extra package for user management called FOSUserBundle (FOS is a shortcut of Friends of Symfony). We will only install it here; we will not configure it. We will configure it in the chapter focused on security and user management.

To install FOSUserBundle, we need to know the correct package name and version. The easiest way is to look in the packagist site at https://packagist.org/ and search for the package there. If you type fosuserbundle, the search should return a package called friendsofsymfony/user-bundle as one of the top results. The download counts visible on the right-hand side might be also helpful in determining how popular the bundle is.
If you click on this, you will end up on the page with the detailed information about that bundle, such as homepage, versions, and requirements of the package.

Type the following command:

```bash
$ php composer.phar require friendsofsymfony/user-bundle ^1.3
```

Updating dependencies (including require-dev)

```
- Installing friendsofsymfony/user-bundle (v1.3.6)
  Loading from cache

friendsofsymfony/user-bundle suggests installing willdurand/propel-typehintable-behavior (Needed when using the propel implementation)
Writing lock file
Generating autoload files
...
```

Which version of the package you choose is up to you. If you are interested in package versioning standards, see the composer website at https://getcomposer.org/doc/01-basic-usage.md#package-versions to get more information on it.

The composer holds all the configurable information about dependencies and where to install them in a special JSON file called composer.json. Let’s take a look at this:

```json
{
   "name": "wbancer/todoapp",
   "license": "proprietary",
   "type": "project",
   "autoload": {
      "psr-0": {
         "": "src/",
         "SymfonyStandard": "app/SymfonyStandard/"
      }
   },
   "require": {
      "php": ">=5.3.9",
      "symfony/symfony": "2.7.*",
      "doctrine/orm": "-2.2,>=2.2.3,<2.5",
      // ...
      "incenteev/composer-parameter-handler": "-2.0",
      "friendsofsymfony/user-bundle": "^1.3"
   },
   "require-dev": {
      "sensio/generator-bundle": "-2.3"
   },
...
"scripts": {
    "post-root-package-install": [
        "SymfonyStandard\Composer::hookRootPackageInstall"
    ],
    "post-install-cmd": [
        "// post installation steps"
    ],
    "post-update-cmd": [
        "// post update steps"
    ]
},
"config": {
    "bin-dir": "bin"
},
"extra": {
    "// [...]
}
}

The most important section is the one with the require key. It holds all the information about the packages we want to use within the project. The key scripts contain a set of instructions to run post-install and post-update. The extra key in this case contains some settings specific to the Symfony2 framework. Note that one of the values in here points out to the parameter.yml file. This file is the main file holding the custom machine-specific parameters. The meaning of the other keys is rather obvious.

If you look into the vendor/ directory, you will notice that our package has been installed in the vendor/friendsofsymfony/user-bundle directory.

### The configuration files

Each application has a need to hold some global and machine-specific parameters and configurations. Symfony2 holds configuration within the app/config directory and it is split into a few files as follows:

- config.yml
- config_dev.yml
- config_prod.yml
- config_test.yml
- parameters.yml
All the files except the `parameters.yml` files contain global configuration, while the `parameters.yml` file holds machine-specific information such as database host, database name, user, password, and SMTP configuration.

The default configuration file generated by the `new` Symfony command will be similar to the following one.

This file is auto-generated during the composer install:

```yaml
parameters:
    database_driver: pdo_mysql
    database_host: 127.0.0.1
    database_port: null
    database_name: symfony
    database_user: root
    database_password: null
    mailer_transport: smtp
    mailer_host: 127.0.0.1
    mailer_user: null
    mailer_pass: null
    secret: 93b0eebeff9e229701f74597e10f8ecf4d94d7f
```

As you can see, it mostly holds the parameters related to database, SMTP, locale settings, and secret key that are used internally by Symfony2. Here, you can add your custom parameters using the same syntax. It is a good practice to keep machine-specific data such as passwords, tokens, api-keys, and access keys within this file only. Putting passwords in the general `config.yml` file is considered as a security risk bug.

The global configuration file (`config.yml`) is split into a few other files called `routing*.yml` that contain information about routing on the development and production configuration. The file called as `security.yml` holds information related to authentication and securing the application access. Note that some files contain information for development, production, or test mode. You can define your mode when you run Symfony through the command-line console and when you run it through the web server. In most cases, while developing you will be using the `dev` mode.
The Symfony2 console

To finish this chapter, let's take a look at the Symfony console script. We used it before to fire up the development server, but it offers more. Execute the following:

$ php app/console

You will see a list of supported commands. Each command has a short description. Each of the standard commands come with help, so I will not be describing each of them here, but it is worth to mention a few commonly used ones:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>app/console: cache:clear</td>
<td>Symfony in production uses a lot of caching. Therefore, if you need to change values within a template (Twig) or within configuration files while in production mode, you will need to clear the cache. Cache is also one of the reasons why it's worth to work in the development mode.</td>
</tr>
<tr>
<td>app/console container:debug</td>
<td>Displays all configured public services</td>
</tr>
<tr>
<td>app/console router:debug</td>
<td>Displays all routing configuration along with method, scheme, host, and path.</td>
</tr>
<tr>
<td>app/console security:check</td>
<td>Checks your composer and packages version against known security vulnerabilities. You should run this command regularly.</td>
</tr>
</tbody>
</table>

You can, of course, write your own Symfony2 commands and we will do this within the forthcoming chapters.

Summary

In this chapter, we have demonstrated how to use the Symfony2 installer, test the configuration, run the deployment server, and play around with the Symfony2 command line. We have also installed the composer and learned how to install a package using it.

While we haven't written much code in this chapter, information contained here will be used in all the other chapters. To demonstrate how Symfony2 enables you to make web applications faster, we will try to learn through examples that can be found in real life. To make this task easier, we will try to produce a real to-do web application with modern look and a few working features.

In the next chapter, we will review the default bundle structure, and set up the first routings, controllers, and templates. We will also create some initial database schema and models, and we will present migration and fixtures system.
Where to buy this book

You can buy Symfony2 Essentials from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.