Symfony

How to contribute to Symfony

Version: 3.3

generated on October 23, 2017

How to contribute to Symfony (3.3)

If you find typos or errors, feel free to report them by creating a ticket on the Symfony ticketing system (*http://github.com/symfony/symfony-docs/issues*). Based on tickets and users feedback, this book is continuously updated.

Contents at a Glance

Chapter 1
Reporting a Bug

Whenever you find a bug in Symfony, we kindly ask you to report it. It helps us make a better Symfony.

If you think you've found a security issue, please use the special *procedure* instead.

Before submitting a bug:

- Double-check the official *documentation* to see if you're not misusing the framework;
- Ask for assistance on *Stack Overflow*[1], on the #support channel of *the Symfony Slack*[2] or on the #symfony *IRC channel*[3] if you're not sure if your issue really is a bug.

If your problem definitely looks like a bug, report it using the official bug *tracker*[4] and follow some basic rules:

- Use the title field to clearly describe the issue;
- Describe the steps needed to reproduce the bug with short code examples (providing a unit test that illustrates the bug is best);
- If the bug you experienced is not obvious or affects more than one layer, providing a simple failing unit test may not be sufficient. In this case, please *provide a reproducer*;
- Give as much detail as possible about your environment (OS, PHP version, Symfony version, enabled extensions, ...);
- If you want to provide a stack trace you got on an HTML page, be sure to provide the plain text version, which should appear at the bottom of the page. *Do not* provide it as a screenshot, since search engines will not be able to index the text inside them. Same goes for errors encountered in a terminal, do not take a screenshot, but copy/paste the contents. If the stack trace is long, consider enclosing it in a *<details> HTML tag*[5]. **Be wary that stack traces may contain sensitive information, and if it is the case, be sure to redact them prior to posting your stack trace.**
- *(optional)* Attach a *patch*.

1. http://stackoverflow.com/questions/tagged/symfony2
2. https://symfony.com/slack-invite
3. https://symfony.com/irc
4. https://github.com/symfony/symfony/issues
5. https://developer.mozilla.org/en-US/docs/Web/HTML/Element/details

Chapter 2
Creating a Bug Reproducer

The main Symfony code repository receives thousands of issues reports per year. Some of those issues are so obvious or easy to understand, that Symfony Core developers can fix them without any other information. However, other issues are much harder to understand because developers can't easily reproduce them in their computers. That's when we'll ask you to create a "bug reproducer", which is the minimum amount of code needed to make the bug appear when executed.

Reproducing Simple Bugs

If you are reporting a bug related to some Symfony component used outside the Symfony framework, it's enough to share a small PHP script that when executed shows the bug:

Listing 2-1
```php
// First, run "composer require symfony/validator"
// Then, execute this file:
<?php
require_once __DIR__.'/vendor/autoload.php';
use Symfony\Component\Validator\Constraints;

$wrongUrl = 'http://example.com/exploit.html?<script>alert(1);</script>';
$urlValidator = new Constraints\UrlValidator();
$urlConstraint = new Constraints\Url();

// The URL is wrong, so var_dump() should display an error, but it displays
// "null" instead because there is no context to build a validator violation
var_dump($urlValidator->validate($wrongUrl, $urlConstraint));
```

Reproducing Complex Bugs

If the bug is related to the Symfony Framework or if it's too complex to create a PHP script, it's better to reproduce the bug by forking the Symfony Standard edition. To do so:

1. Go to *https://github.com/symfony/symfony-standard* and click on the **Fork** button to make a fork of that repository or go to your already forked copy.
2. Clone the forked repository into your computer: `git clone git://github.com/YOUR-GITHUB-USERNAME/symfony-standard.git`
3. Browse the project and create a new branch (e.g. `issue_23567`, `reproduce_23657`, etc.)

4. Now you must add the minimum amount of code to reproduce the bug. This is the trickiest part and it's explained a bit more later.
5. Add, commit and push all your changes.
6. Add a comment in your original issue report to share the URL of your forked project (e.g. `https://github.com/YOUR-GITHUB-USERNAME/symfony-standard/tree/issue_23567`) and, if necessary, explain the steps to reproduce (e.g. "browse this URL", "fill in this data in the form and submit it", etc.)

Adding the Minimum Amount of Code Possible

The key to create a bug reproducer is to solely focus on the feature that you suspect is failing. For example, imagine that you suspect that the bug is related to a route definition. Then, after forking the Symfony Standard Edition:

1. Don't edit any of the default Symfony configuration options.
2. Don't copy your original application code and don't use the same structure of bundles, controllers, actions, etc. as in your original application.
3. Open the default controller class of the AppBundle and add your routing definition using annotations.
4. Don't create or modify any other file.
5. Execute the `server:run` command and browse the previously defined route to see if the bug appears or not.
6. If you can see the bug, you're done and you can already share the code with us.
7. If you can't see the bug, you must keep making small changes. For example, if your original route was defined using XML, forget about the previous route annotation and define the route using XML instead. Or maybe your application uses bundle inheritance and that's where the real bug is. Then, forget about AppBundle and quickly generate a new AppParentBundle, make AppBundle inherit from it and test if the route is working.

In short, the idea is to keep adding small and incremental changes to the default Symfony Standard edition until you can reproduce the bug.

Chapter 3
Submitting a Patch

Patches are the best way to provide a bug fix or to propose enhancements to Symfony.

Step 1: Setup your Environment

Install the Software Stack

Before working on Symfony, setup a friendly environment with the following software:

- Git;
- PHP version 5.5.9 or above.

Before Symfony 2.7, the minimal PHP version was 5.3.3. Before Symfony 3.0, minimal version was 5.3.9. Please keep this in mind, if you are working on a bug fix for earlier versions of Symfony.

Configure Git

Set up your user information with your real name and a working email address:

Listing 3-1
```
$ git config --global user.name "Your Name"
$ git config --global user.email you@example.com
```

If you are new to Git, you are highly recommended to read the excellent and free *ProGit*[1] book.

1. http://git-scm.com/book

If your IDE creates configuration files inside the project's directory, you can use global `.gitignore` file (for all projects) or `.git/info/exclude` file (per project) to ignore them. See *GitHub's documentation*[2].

Windows users: when installing Git, the installer will ask what to do with line endings, and suggests replacing all LF with CRLF. This is the wrong setting if you wish to contribute to Symfony! Selecting the as-is method is your best choice, as Git will convert your line feeds to the ones in the repository. If you have already installed Git, you can check the value of this setting by typing:

Listing 3-2
```
$ git config core.autocrlf
```

This will return either "false", "input" or "true"; "true" and "false" being the wrong values. Change it to "input" by typing:

Listing 3-3
```
$ git config --global core.autocrlf input
```

Replace --global by --local if you want to set it only for the active repository

Get the Symfony Source Code

Get the Symfony source code:

- Create a *GitHub*[3] account and sign in;
- Fork the *Symfony repository*[4] (click on the "Fork" button);
- After the "forking action" has completed, clone your fork locally (this will create a `symfony` directory):

Listing 3-4
```
$ git clone git@github.com:USERNAME/symfony.git
```

- Add the upstream repository as a remote:

Listing 3-5
```
$ cd symfony
$ git remote add upstream git://github.com/symfony/symfony.git
```

Check that the current Tests Pass

Now that Symfony is installed, check that all unit tests pass for your environment as explained in the dedicated *document*.

Step 2: Work on your Patch

The License

Before you start, you must know that all the patches you are going to submit must be released under the *MIT license*, unless explicitly specified in your commits.

2. https://help.github.com/articles/ignoring-files
3. https://github.com/join
4. https://github.com/symfony/symfony

Choose the right Branch

Before working on a patch, you must determine on which branch you need to work:

- `2.7`, if you are fixing a bug for an existing feature or want to make a change that falls into the *list of acceptable changes in patch versions* (you may have to choose a higher branch if the feature you are fixing was introduced in a later version);

> - `master`, if you are adding a new feature.

All bug fixes merged into maintenance branches are also merged into more recent branches on a regular basis. For instance, if you submit a patch for the `2.7` branch, the patch will also be applied by the core team on the `master` branch.

Create a Topic Branch

Each time you want to work on a patch for a bug or on an enhancement, create a topic branch:

Listing 3-6
```
$ git checkout -b BRANCH_NAME master
```

Or, if you want to provide a bugfix for the `2.7` branch, first track the remote `2.7` branch locally:

Listing 3-7
```
$ git checkout -t origin/2.7
```

Then create a new branch off the `2.7` branch to work on the bugfix:

Listing 3-8
```
$ git checkout -b BRANCH_NAME 2.7
```

Use a descriptive name for your branch (`ticket_XXX` where `XXX` is the ticket number is a good convention for bug fixes).

The above checkout commands automatically switch the code to the newly created branch (check the branch you are working on with `git branch`).

Work on your Patch

Work on the code as much as you want and commit as much as you want; but keep in mind the following:

- Read about the Symfony *conventions* and follow the coding *standards* (use `git diff --check` to check for trailing spaces -- also read the tip below);
- Add unit tests to prove that the bug is fixed or that the new feature actually works;
- Try hard to not break backward compatibility (if you must do so, try to provide a compatibility layer to support the old way) -- patches that break backward compatibility have less chance to be merged;
- Do atomic and logically separate commits (use the power of `git rebase` to have a clean and logical history);
- Never fix coding standards in some existing code as it makes the code review more difficult;
- Write good commit messages (see the tip below).

When submitting pull requests, *fabbot*[5] checks your code for common typos and verifies that you are using the PHP coding standards as defined in *PSR-1*[6] and *PSR-2*[7].

A status is posted below the pull request description with a summary of any problems it detects or any Travis CI build failures.

A good commit message is composed of a summary (the first line), optionally followed by a blank line and a more detailed description. The summary should start with the Component you are working on in square brackets (`[DependencyInjection]`, `[FrameworkBundle]`, ...). Use a verb (`fixed ...`, `added ...`, ...) to start the summary and don't add a period at the end.

Prepare your Patch for Submission

When your patch is not about a bug fix (when you add a new feature or change an existing one for instance), it must also include the following:

- An explanation of the changes in the relevant `CHANGELOG` file(s) (the `[BC BREAK]` or the `[DEPRECATION]` prefix must be used when relevant);
- An explanation on how to upgrade an existing application in the relevant `UPGRADE` file(s) if the changes break backward compatibility or if you deprecate something that will ultimately break backward compatibility.

Step 3: Submit your Patch

Whenever you feel that your patch is ready for submission, follow the following steps.

Rebase your Patch

Before submitting your patch, update your branch (needed if it takes you a while to finish your changes):

Listing 3-9
```
$ git checkout master
$ git fetch upstream
$ git merge upstream/master
$ git checkout BRANCH_NAME
$ git rebase master
```

Replace `master` with the branch you selected previously (e.g. `2.7`) if you are working on a bugfix

When doing the `rebase` command, you might have to fix merge conflicts. `git status` will show you the *unmerged* files. Resolve all the conflicts, then continue the rebase:

Listing 3-10
```
$ git add ... # add resolved files
$ git rebase --continue
```

Check that all tests still pass and push your branch remotely:

Listing 3-11
```
$ git push --force origin BRANCH_NAME
```

5. http://fabbot.io
6. http://www.php-fig.org/psr/psr-1/
7. http://www.php-fig.org/psr/psr-2/

Make a Pull Request

You can now make a pull request on the `symfony/symfony` GitHub repository.

Take care to point your pull request towards `symfony:2.7` if you want the core team to pull a bugfix based on the `2.7` branch.

To ease the core team work, always include the modified components in your pull request message, like in:

Listing 3-12
```
[Yaml] fixed something
[Form] [Validator] [FrameworkBundle] added something
```

The default pull request description contains a table which you must fill in with the appropriate answers. This ensures that contributions may be reviewed without needless feedback loops and that your contributions can be included into Symfony as quickly as possible.

Some answers to the questions trigger some more requirements:

- If you answer yes to "Bug fix?", check if the bug is already listed in the Symfony issues and reference it/them in "Fixed tickets";
- If you answer yes to "New feature?", you must submit a pull request to the documentation and reference it under the "Doc PR" section;
- If you answer yes to "BC breaks?", the patch must contain updates to the relevant `CHANGELOG` and `UPGRADE` files;
- If you answer yes to "Deprecations?", the patch must contain updates to the relevant `CHANGELOG` and `UPGRADE` files;
- If you answer no to "Tests pass", you must add an item to a todo-list with the actions that must be done to fix the tests;
- If the "license" is not MIT, just don't submit the pull request as it won't be accepted anyway.

If some of the previous requirements are not met, create a todo-list and add relevant items:

Listing 3-13
```
- [ ] fix the tests as they have not been updated yet
- [ ] submit changes to the documentation
- [ ] document the BC breaks
```

If the code is not finished yet because you don't have time to finish it or because you want early feedback on your work, add an item to todo-list:

Listing 3-14
```
- [ ] finish the code
- [ ] gather feedback for my changes
```

As long as you have items in the todo-list, please prefix the pull request title with "[WIP]".

In the pull request description, give as much details as possible about your changes (don't hesitate to give code examples to illustrate your points). If your pull request is about adding a new feature or modifying an existing one, explain the rationale for the changes. The pull request description helps the code review and it serves as a reference when the code is merged (the pull request description and all its associated comments are part of the merge commit message).

In addition to this "code" pull request, you must also send a pull request to the *documentation repository*[8] to update the documentation when appropriate.

8. `https://github.com/symfony/symfony-docs`

Rework your Patch

Based on the feedback on the pull request, you might need to rework your patch. Before re-submitting the patch, rebase with `upstream/master` or `upstream/2.7`, don't merge; and force the push to the origin:

Listing 3-15
```
$ git rebase -f upstream/master
$ git push --force origin BRANCH_NAME
```

When doing a `push --force`, always specify the branch name explicitly to avoid messing other branches in the repo (`--force` tells Git that you really want to mess with things so do it carefully).

Moderators earlier asked you to "squash" your commits. This means you will convert many commits to one commit. This is no longer necessary today, because Symfony project uses a proprietary tool which automatically squashes all commits before merging.

Chapter 4
Maintenance

During the lifetime of a minor version, new releases (patch versions) are published on a monthly basis. This document describes the boundaries of acceptable changes.

Bug fixes are accepted under the following conditions:

- The change does not break valid unit tests;
- New unit tests cover the bug fix;
- The current buggy behavior is not widely used as a "feature".

When documentation (or phpdoc) is not in sync with the code, code behavior should always be considered as being the correct one.

Besides bug fixes, other minor changes can be accepted in a patch version:

- **Performance improvement**: Performance improvement should only be accepted if the changes are local (located in one class) and only for algorithmic issues (any such patches must come with numbers that show a significant improvement on real-world code);
- **Newer versions of PHP/HHVM**: Fixes that add support for newer versions of PHP or HHVM are acceptable if they don't break the unit test suite;
- **Newer versions of popular OSes**: Fixes that add support for newer versions of popular OSes (Linux, MacOS and Windows) are acceptable if they don't break the unit test suite;
- **Translations**: Translation updates and additions are accepted;
- **External data**: Updates for external data included in Symfony can be updated (like ICU for instance);
- **Version updates for Composer dependencies**: Changing the minimal version of a dependency is possible, bumping to a major one or increasing PHP minimal version is not;
- **Coding standard and refactoring**: Coding standard fixes or code refactoring are not recommended but can be accepted for consistency with the existing code base, if they are not too invasive, and if merging them on master would not lead to complex branch merging;
- **Tests**: Tests that increase the code coverage can be added.

Anything not explicitly listed above should be done on the next minor or major version instead (aka the *master* branch). For instance, the following changes are never accepted in a patch version:

- **New features**;

- **Backward compatibility breaks**: Note that backward compatibility breaks can be done when fixing a security issue if it would not be possible to fix it otherwise;
- **Support for external platforms**: Adding support for new platforms (like Google App Engine) cannot be done in patch versions;
- **Exception messages**: Exception messages must not be changed as some automated systems might rely on them (even if this is not recommended);
- **Adding new Composer dependencies**;
- **Support for newer major versions of Composer dependencies**: Taking into account support for newer versions of an existing dependency is not acceptable.
- **Web design**: Changing the web design of built-in pages like exceptions, the toolbar or the profiler is not allowed.

This policy is designed to enable a continuous upgrade path that allows one to move forward with newest Symfony versions in the safest way. One should be able to move PHP versions, OS or Symfony versions almost independently. That's the reason why supporting the latest PHP versions or OS features is considered as bug fixes.

Chapter 5
Symfony Core Team

The **Symfony Core** team is the group of developers that determine the direction and evolution of the Symfony project. Their votes rule if the features and patches proposed by the community are approved or rejected.

All the Symfony Core members are long-time contributors with solid technical expertise and they have demonstrated a strong commitment to drive the project forward.

This document states the rules that govern the Symfony Core team. These rules are effective upon publication of this document and all Symfony Core members must adhere to said rules and protocol.

Core Organization

Symfony Core members are divided into three groups. Each member can only belong to one group at a time. The privileges granted to a group are automatically granted to all higher priority groups.

The Symfony Core groups, in descending order of priority, are as follows:

1. **Project Leader**

- Elects members in any other group;
- Merges pull requests in all Symfony repositories.

2. **Mergers**

- Merge pull requests for the component or components on which they have been granted privileges.

3. **Deciders**

- Decide to merge or reject a pull request.

Active Core Members

- **Project Leader**:
    - **Fabien Potencier** (*fabpot*[1]).

1. https://github.com/fabpot/

- **Mergers** (`@symfony/mergers` on GitHub):
    - **Tobias Schultze** (*Tobion*[2]) can merge into the *Routing*[3], *OptionsResolver*[4] and *PropertyAccess*[5] components;
    - **Romain Neutron** (*romainneutron*[6]) can merge into the *Process*[7] component;
    - **Nicolas Grekas** (*nicolas-grekas*[8]) can merge into the *Cache*[9], *Debug*[10], *Process*[11], *PropertyAccess*[12], *VarDumper*[13] components, *PhpUnitBridge*[14] and the *DebugBundle*[15];
    - **Christophe Coevoet** (*stof*[16]) can merge into all components, bridges and bundles;
    - **Kévin Dunglas** (*dunglas*[17]) can merge into the *PropertyInfo*[18] and the *Serializer*[19] component;
    - **Jakub Zalas** (*jakzal*[20]) can merge into the *DomCrawler*[21] and *Intl*[22] components;
    - **Christian Flothmann** (*xabbuh*[23]) can merge into the *Yaml*[24] component;
    - **Javier Eguiluz** (*javiereguiluz*[25]) can merge into the *WebProfilerBundle*[26];
    - **Grégoire Pineau** (*lyrixx*[27]) can merge into the *Workflow*[28] component;
    - **Ryan Weaver** (*weaverryan*[29]) can merge into the *Security*[30] component and the *SecurityBundle*[31];
    - **Robin Chalas** (*chalasr*[32]) can merge into the *Console*[33] and *Security*[34] components and the *SecurityBundle*[35];
    - **Maxime Steinhausser** (*ogizanagi*[36]) can merge into *Config*[37], *Console*[38], *Form*[39], *Serializer*[40], *DependencyInjection*[41], and *HttpKernel*[42] components.

---

2. https://github.com/Tobion/
3. https://github.com/symfony/routing
4. https://github.com/symfony/options-resolver
5. https://github.com/symfony/property-access
6. https://github.com/romainneutron/
7. https://github.com/symfony/process
8. https://github.com/nicolas-grekas/
9. https://github.com/symfony/cache
10. https://github.com/symfony/debug
11. https://github.com/symfony/process
12. https://github.com/symfony/property-access
13. https://github.com/symfony/var-dumper
14. https://github.com/symfony/phpunit-bridge
15. https://github.com/symfony/debug-bundle
16. https://github.com/stof/
17. https://github.com/dunglas/
18. https://github.com/symfony/property-info
19. https://github.com/symfony/serializer
20. https://github.com/jakzal/
21. https://github.com/symfony/dom-crawler
22. https://github.com/symfony/intl
23. https://github.com/xabbuh/
24. https://github.com/symfony/yaml
25. https://github.com/javiereguiluz/
26. https://github.com/symfony/web-profiler-bundle
27. https://github.com/lyrixx/
28. https://github.com/symfony/workflow
29. https://github.com/weaverryan/
30. https://github.com/symfony/security
31. https://github.com/symfony/security-bundle
32. https://github.com/chalasr/
33. https://github.com/symfony/console
34. https://github.com/symfony/security
35. https://github.com/symfony/security-bundle
36. https://github.com/ogizanagi/
37. https://github.com/symfony/config
38. https://github.com/symfony/console
39. https://github.com/symfony/form
40. https://github.com/symfony/serializer
41. https://github.com/symfony/dependency-injection
42. https://github.com/symfony/http-kernel

- **Deciders** (`@symfony/deciders` on GitHub):
    - **Jordi Boggiano** (*seldaek*[43]);
    - **Lukas Kahwe Smith** (*lsmith77*[44]).

Former Core Members

They are no longer part of the Core Team, but we are very grateful for all their Symfony contributions:

> - **Bernhard Schussek** (*webmozart*[45]);
> - **Abdellatif AitBoudad** (*aitboudad*[46]).

Core Membership Application

At present, new Symfony Core membership applications are not accepted.

Core Membership Revocation

A Symfony Core membership can be revoked for any of the following reasons:

- Refusal to follow the rules and policies stated in this document;
- Lack of activity for the past six months;
- Willful negligence or intent to harm the Symfony project;
- Upon decision of the **Project Leader**.

Should new Symfony Core memberships be accepted in the future, revoked members must wait at least 12 months before re-applying.

Code Development Rules

Symfony project development is based on pull requests proposed by any member of the Symfony community. Pull request acceptance or rejection is decided based on the votes cast by the Symfony Core members.

Pull Request Voting Policy

- `-1` votes must always be justified by technical and objective reasons;
- `+1` votes do not require justification, unless there is at least one `-1` vote;
- Core members can change their votes as many times as they desire during the course of a pull request discussion;
- Core members are not allowed to vote on their own pull requests.

Pull Request Merging Policy

A pull request **can be merged** if:

- It is a minor change [1];

43. https://github.com/Seldaek/
44. https://github.com/lsmith77/
45. https://github.com/webmozart/
46. https://github.com/aitboudad/

- Enough time was given for peer reviews (at least 2 days for "regular" pull requests, and 4 days for pull requests with "a significant impact");
- At least the component's **Merger** or two other Core members voted `+1` and no Core member voted `-1`.

Pull Request Merging Process

All code must be committed to the repository through pull requests, except for minor changes [1] which can be committed directly to the repository.

Mergers must always use the command-line `gh` tool provided by the **Project Leader** to merge the pull requests.

Release Policy

The **Project Leader** is also the release manager for every Symfony version.

Symfony Core Rules and Protocol Amendments

The rules described in this document may be amended at anytime at the discretion of the **Project Leader**.[47]

47.

[1] *(1, 2)* Minor changes comprise typos, DocBlock fixes, code standards violations, and minor CSS, JavaScript and HTML modifications.

Chapter 6
Security Issues

This document explains how Symfony security issues are handled by the Symfony core team (Symfony being the code hosted on the main `symfony/symfony` Git repository).

Reporting a Security Issue

If you think that you have found a security issue in Symfony, don't use the bug tracker and don't publish it publicly. Instead, all security issues must be sent to **security [at] symfony.com**. Emails sent to this address are forwarded to the Symfony core-team private mailing-list.

Resolving Process

For each report, we first try to confirm the vulnerability. When it is confirmed, the core-team works on a solution following these steps:

1. Send an acknowledgement to the reporter;
2. Work on a patch;
3. Get a CVE identifier from *mitre.org*[1];
4. Write a security announcement for the official Symfony *blog*[2] about the vulnerability. This post should contain the following information:

    - a title that always include the "Security release" string;
    - a description of the vulnerability;
    - the affected versions;
    - the possible exploits;
    - how to patch/upgrade/workaround affected applications;
    - the CVE identifier;
    - credits.

5. Send the patch and the announcement to the reporter for review;
6. Apply the patch to all maintained versions of Symfony;

1. `https://cveform.mitre.org/`
2. `https://symfony.com/blog/`

7. Package new versions for all affected versions;
8. Publish the post on the official Symfony *blog*[3] (it must also be added to the "*Security Advisories*[4]" category);
9. Update the security advisory list (see below).
10. Update the public *security advisories database*[5] maintained by the FriendsOfPHP organization and which is used by the `security:check` command.

Releases that include security issues should not be done on Saturday or Sunday, except if the vulnerability has been publicly posted.

While we are working on a patch, please do not reveal the issue publicly.

The resolution takes anywhere between a couple of days to a month depending on its complexity and the coordination with the downstream projects (see next paragraph).

Collaborating with Downstream Open-Source Projects

As Symfony is used by many large Open-Source projects, we standardized the way the Symfony security team collaborates on security issues with downstream projects. The process works as follows:

1. After the Symfony security team has acknowledged a security issue, it immediately sends an email to the downstream project security teams to inform them of the issue;
2. The Symfony security team creates a private Git repository to ease the collaboration on the issue and access to this repository is given to the Symfony security team, to the Symfony contributors that are impacted by the issue, and to one representative of each downstream projects;
3. All people with access to the private repository work on a solution to solve the issue via pull requests, code reviews, and comments;
4. Once the fix is found, all involved projects collaborate to find the best date for a joint release (there is no guarantee that all releases will be at the same time but we will try hard to make them at about the same time). When the issue is not known to be exploited in the wild, a period of two weeks seems like a reasonable amount of time.

The list of downstream projects participating in this process is kept as small as possible in order to better manage the flow of confidential information prior to disclosure. As such, projects are included at the sole discretion of the Symfony security team.

As of today, the following projects have validated this process and are part of the downstream projects included in this process:

- Drupal (releases typically happen on Wednesdays)
- eZPublish

Security Advisories

3. `https://symfony.com/blog/`
4. `https://symfony.com/blog/category/security-advisories`
5. `https://github.com/FriendsOfPHP/security-advisories`

You can check your Symfony application for known security vulnerabilities using the `security:check` command (see *How to Check for Known Security Vulnerabilities in Your Dependencies*).

This section indexes security vulnerabilities that were fixed in Symfony releases, starting from Symfony 1.0.0:

- Jul 17, 2017, *CVE-2017-11365: Empty passwords validation issue*[6] (2.7.30, 2.7.31, 2.8.23, 2.8.24, 3.2.10, 3.2.11, 3.3.3, and 3.3.4)
- May 9, 2016: *CVE-2016-2403: Unauthorized access on a misconfigured Ldap server when using an empty password*[7] (2.8.0-2.8.5, 3.0.0-3.0.5)
- May 9, 2016: *CVE-2016-4423: Large username storage in session*[8] (2.3.0-2.3.40, 2.7.0-2.7.12, 2.8.0-2.8.5, 3.0.0-3.0.5)
- January 18, 2016: *CVE-2016-1902: SecureRandom's fallback not secure when OpenSSL fails*[9] (2.3.0-2.3.36, 2.6.0-2.6.12, 2.7.0-2.7.8)
- November 23, 2015: *CVE-2015-8125: Potential Remote Timing Attack Vulnerability in Security Remember-Me Service*[10] (2.3.35, 2.6.12 and 2.7.7)
- November 23, 2015: *CVE-2015-8124: Session Fixation in the "Remember Me" Login Feature*[11] (2.3.35, 2.6.12 and 2.7.7)
- May 26, 2015: *CVE-2015-4050: ESI unauthorized access*[12] (Symfony 2.3.29, 2.5.12 and 2.6.8)
- April 1, 2015: *CVE-2015-2309: Unsafe methods in the Request class*[13] (Symfony 2.3.27, 2.5.11 and 2.6.6)
- April 1, 2015: *CVE-2015-2308: Esi Code Injection*[14] (Symfony 2.3.27, 2.5.11 and 2.6.6)
- September 3, 2014: *CVE-2014-6072: CSRF vulnerability in the Web Profiler*[15] (Symfony 2.3.19, 2.4.9 and 2.5.4)
- September 3, 2014: *CVE-2014-6061: Security issue when parsing the Authorization header*[16] (Symfony 2.3.19, 2.4.9 and 2.5.4)
- September 3, 2014: *CVE-2014-5245: Direct access of ESI URLs behind a trusted proxy*[17] (Symfony 2.3.19, 2.4.9 and 2.5.4)
- September 3, 2014: *CVE-2014-5244: Denial of service with a malicious HTTP Host header*[18] (Symfony 2.3.19, 2.4.9 and 2.5.4)
- July 15, 2014: *Security releases: Symfony 2.3.18, 2.4.8, and 2.5.2 released*[19] (*CVE-2014-4931*[20])
- October 10, 2013: *Security releases: Symfony 2.0.25, 2.1.13, 2.2.9, and 2.3.6 released*[21] (*CVE-2013-5958*[22])
- August 7, 2013: *Security releases: Symfony 2.0.24, 2.1.12, 2.2.5, and 2.3.3 released*[23] (*CVE-2013-4751*[24] and *CVE-2013-4752*[25])

---

6. https://symfony.com/blog/cve-2017-11365-empty-passwords-validation-issue
7. https://symfony.com/blog/cve-2016-2403-unauthorized-access-on-a-misconfigured-ldap-server-when-using-an-empty-password
8. https://symfony.com/blog/cve-2016-4423-large-username-storage-in-session
9. https://symfony.com/blog/cve-2016-1902-securerandom-s-fallback-not-secure-when-openssl-fails
10. https://symfony.com/blog/cve-2015-8125-potential-remote-timing-attack-vulnerability-in-security-remember-me-service
11. https://symfony.com/blog/cve-2015-8124-session-fixation-in-the-remember-me-login-feature
12. https://symfony.com/blog/cve-2015-4050-esi-unauthorized-access
13. https://symfony.com/blog/cve-2015-2309-unsafe-methods-in-the-request-class
14. https://symfony.com/blog/cve-2015-2308-esi-code-injection
15. https://symfony.com/blog/cve-2014-6072-csrf-vulnerability-in-the-web-profiler
16. https://symfony.com/blog/cve-2014-6061-security-issue-when-parsing-the-authorization-header
17. https://symfony.com/blog/cve-2014-5245-direct-access-of-esi-urls-behind-a-trusted-proxy
18. https://symfony.com/blog/cve-2014-5244-denial-of-service-with-a-malicious-http-host-header
19. https://symfony.com/blog/security-releases-cve-2014-4931-symfony-2-3-18-2-4-8-and-2-5-2-released
20. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-4931
21. https://symfony.com/blog/security-releases-cve-2013-5958-symfony-2-0-25-2-1-13-2-2-9-and-2-3-6-released
22. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-5958
23. https://symfony.com/blog/security-releases-symfony-2-0-24-2-1-12-2-2-5-and-2-3-3-released
24. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-4751
25. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-4752

- January 17, 2013: *Security release: Symfony 2.0.22 and 2.1.7 released*[26] (*CVE-2013-1348*[27] and *CVE-2013-1397*[28])
- December 20, 2012: *Security release: Symfony 2.0.20 and 2.1.5*[29] (*CVE-2012-6431*[30] and *CVE-2012-6432*[31])
- November 29, 2012: *Security release: Symfony 2.0.19 and 2.1.4*[32]
- November 25, 2012: *Security release: symfony 1.4.20 released*[33] (*CVE-2012-5574*[34])
- August 28, 2012: *Security Release: Symfony 2.0.17 released*[35]
- May 30, 2012: *Security Release: symfony 1.4.18 released*[36] (*CVE-2012-2667*[37])
- February 24, 2012: *Security Release: Symfony 2.0.11 released*[38]
- November 16, 2011: *Security Release: Symfony 2.0.6*[39]
- March 21, 2011: *symfony 1.3.10 and 1.4.10: security releases*[40]
- June 29, 2010: *Security Release: symfony 1.3.6 and 1.4.6*[41]
- May 31, 2010: *symfony 1.3.5 and 1.4.5*[42]
- February 25, 2010: *Security Release: 1.2.12, 1.3.3 and 1.4.3*[43]
- February 13, 2010: *symfony 1.3.2 and 1.4.2*[44]
- April 27, 2009: *symfony 1.2.6: Security fix*[45]
- October 03, 2008: *symfony 1.1.4 released: Security fix*[46]
- May 14, 2008: *symfony 1.0.16 is out*[47]
- April 01, 2008: *symfony 1.0.13 is out*[48]
- March 21, 2008: *symfony 1.0.12 is (finally) out !*[49]
- June 25, 2007: *symfony 1.0.5 released (security fix)*[50]

26. https://symfony.com/blog/security-release-symfony-2-0-22-and-2-1-7-released
27. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-1348
28. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-1397
29. https://symfony.com/blog/security-release-symfony-2-0-20-and-2-1-5-released
30. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-6431
31. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-6432
32. https://symfony.com/blog/security-release-symfony-2-0-19-and-2-1-4
33. https://symfony.com/blog/security-release-symfony-1-4-20-released
34. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-5574
35. https://symfony.com/blog/security-release-symfony-2-0-17-released
36. https://symfony.com/blog/security-release-symfony-1-4-18-released
37. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-2667
38. https://symfony.com/blog/security-release-symfony-2-0-11-released
39. https://symfony.com/blog/security-release-symfony-2-0-6
40. https://symfony.com/blog/symfony-1-3-10-and-1-4-10-security-releases
41. https://symfony.com/blog/security-release-symfony-1-3-6-and-1-4-6
42. https://symfony.com/blog/symfony-1-3-5-and-1-4-5
43. https://symfony.com/blog/security-release-1-2-12-1-3-3-and-1-4-3
44. https://symfony.com/blog/symfony-1-3-2-and-1-4-2
45. https://symfony.com/blog/symfony-1-2-6-security-fix
46. https://symfony.com/blog/symfony-1-1-4-released-security-fix
47. https://symfony.com/blog/symfony-1-0-16-is-out
48. https://symfony.com/blog/symfony-1-0-13-is-out
49. https://symfony.com/blog/symfony-1-0-12-is-finally-out
50. https://symfony.com/blog/symfony-1-0-5-released-security-fix

Chapter 7
Running Symfony Tests

The Symfony project uses a third-party service which automatically runs tests for any submitted *patch*. If the new code breaks any test, the pull request will show an error message with a link to the full error details.

In any case, it's a good practice to run tests locally before submitting a *patch* for inclusion, to check that you have not broken anything.

Before Running the Tests

To run the Symfony test suite, install the external dependencies used during the tests, such as Doctrine, Twig and Monolog. To do so, *install Composer* and execute the following:

Listing 7-1
```
1 $ composer update
```

Running the Tests

Then, run the test suite from the Symfony root directory with the following command:

Listing 7-2
```
1 $ php ./phpunit symfony
```

The output should display `OK`. If not, read the reported errors to figure out what's going on and if the tests are broken because of the new code.

The entire Symfony suite can take up to several minutes to complete. If you want to test a single component, type its path after the `phpunit` command, e.g.:

Listing 7-3
```
1 $ php ./phpunit src/Symfony/Component/Finder/
```

On Windows, install the *Cmder*[1], *ConEmu*[2], *ANSICON*[3] or *Mintty*[4] free applications to see colored test results.

1. http://cmder.net/
2. https://conemu.github.io/
3. https://github.com/adoxa/ansicon/releases
4. https://mintty.github.io/

Chapter 8
Our Backward Compatibility Promise

Ensuring smooth upgrades of your projects is our first priority. That's why we promise you backward compatibility (BC) for all minor Symfony releases. You probably recognize this strategy as *Semantic Versioning*[1]. In short, Semantic Versioning means that only major releases (such as 2.0, 3.0 etc.) are allowed to break backward compatibility. Minor releases (such as 2.5, 2.6 etc.) may introduce new features, but must do so without breaking the existing API of that release branch (2.x in the previous example).

This promise was introduced with Symfony 2.3 and does not apply to previous versions of Symfony.

However, backward compatibility comes in many different flavors. In fact, almost every change that we make to the framework can potentially break an application. For example, if we add a new method to a class, this will break an application which extended this class and added the same method, but with a different method signature.

Also, not every BC break has the same impact on application code. While some BC breaks require you to make significant changes to your classes or your architecture, others are fixed as easily as changing the name of a method.

That's why we created this page for you. The section "Using Symfony Code" will tell you how you can ensure that your application won't break completely when upgrading to a newer version of the same major release branch.

The second section, "Working on Symfony Code", is targeted at Symfony contributors. This section lists detailed rules that every contributor needs to follow to ensure smooth upgrades for our users.

Experimental Features and code marked with the `@internal` tags are excluded from our Backward Compatibility promise.

Also note that backward compatibility breaks are tolerated if they are required to fix a security issue.

1. `http://semver.org/`

Using Symfony Code

If you are using Symfony in your projects, the following guidelines will help you to ensure smooth upgrades to all future minor releases of your Symfony version.

Using our Interfaces

All interfaces shipped with Symfony can be used in type hints. You can also call any of the methods that they declare. We guarantee that we won't break code that sticks to these rules.

The exception to this rule are interfaces tagged with `@internal`. Such interfaces should not be used or implemented.

If you implement an interface, we promise that we won't ever break your code.

The following table explains in detail which use cases are covered by our backward compatibility promise:

Use Case	**Backward Compatibility**
If you...	**Then we guarantee BC...**
Type hint against the interface	Yes
Call a method	Yes
If you implement the interface and...	**Then we guarantee BC...**
Implement a method	Yes
Add an argument to an implemented method	Yes
Add a default value to an argument	Yes

Using our Classes

All classes provided by Symfony may be instantiated and accessed through their public methods and properties.

Classes, properties and methods that bear the tag `@internal` as well as the classes located in the various `*\\Tests\\` namespaces are an exception to this rule. They are meant for internal use only and should not be accessed by your own code.

To be on the safe side, check the following table to know which use cases are covered by our backward compatibility promise:

Use Case	**Backward Compatibility**
If you...	**Then we guarantee BC...**
Type hint against the class	Yes
Create a new instance	Yes
Extend the class	Yes
Access a public property	Yes
Call a public method	Yes

Use Case	Backward Compatibility
If you extend the class and...	**Then we guarantee BC...**
Access a protected property	Yes
Call a protected method	Yes
Override a public property	Yes
Override a protected property	Yes
Override a public method	Yes
Override a protected method	Yes
Add a new property	No
Add a new method	No
Add an argument to an overridden method	Yes
Add a default value to an argument	Yes
Call a private method (via Reflection)	No
Access a private property (via Reflection)	No

Working on Symfony Code

Do you want to help us improve Symfony? That's great! However, please stick to the rules listed below in order to ensure smooth upgrades for our users.

Changing Interfaces

This table tells you which changes you are allowed to do when working on Symfony's interfaces:

Type of Change	Change Allowed
Remove entirely	No
Change name or namespace	No
Add parent interface	Yes [2]
Remove parent interface	No
Methods	
Add method	No
Remove method	No
Change name	No
Move to parent interface	Yes
Add argument without a default value	No
Add argument with a default value	No
Remove argument	Yes [3]
Add default value to an argument	No
Remove default value of an argument	No

Type of Change	Change Allowed
Add type hint to an argument	No
Remove type hint of an argument	No
Change argument type	No
Change return type	No
Constants	
Add constant	Yes
Remove constant	No
Change value of a constant	Yes [1] [5]

Changing Classes

This table tells you which changes you are allowed to do when working on Symfony's classes:

Type of Change	Change Allowed
Remove entirely	No
Make final	No
Make abstract	No
Change name or namespace	No
Change parent class	Yes [4]
Add interface	Yes
Remove interface	No
Public Properties	
Add public property	Yes
Remove public property	No
Reduce visibility	No
Move to parent class	Yes
Protected Properties	
Add protected property	Yes
Remove protected property	No
Reduce visibility	No
Move to parent class	Yes
Private Properties	
Add private property	Yes
Remove private property	Yes
Constructors	
Add constructor without mandatory arguments	Yes [1]
Remove constructor	No

Type of Change	Change Allowed
Reduce visibility of a public constructor	No
Reduce visibility of a protected constructor	No
Move to parent class	Yes
Public Methods	
Add public method	Yes
Remove public method	No
Change name	No
Reduce visibility	No
Move to parent class	Yes
Add argument without a default value	No
Add argument with a default value	No
Remove argument	Yes [3]
Add default value to an argument	No
Remove default value of an argument	No
Add type hint to an argument	No
Remove type hint of an argument	No
Change argument type	No
Change return type	No
Protected Methods	
Add protected method	Yes
Remove protected method	No
Change name	No
Reduce visibility	No
Move to parent class	Yes
Add argument without a default value	No
Add argument with a default value	No
Remove argument	Yes [3]
Add default value to an argument	No
Remove default value of an argument	No
Add type hint to an argument	No
Remove type hint of an argument	No
Change argument type	No
Change return type	No
Private Methods	
Add private method	Yes
Remove private method	Yes

Type of Change	Change Allowed
Change name	Yes
Add argument without a default value	Yes
Add argument with a default value	Yes
Remove argument	Yes
Add default value to an argument	Yes
Remove default value of an argument	Yes
Add type hint to an argument	Yes
Remove type hint of an argument	Yes
Change argument type	Yes
Change return type	Yes
Static Methods	
Turn non static into static	No
Turn static into non static	No
Constants	
Add constant	Yes
Remove constant	No
Change value of a constant	Yes [1] [5]

2 3 4 5 6

2.

[1] *(1, 2, 3)* Should be avoided. When done, this change must be documented in the UPGRADE file.

3.

[2] The added parent interface must not introduce any new methods that don't exist in the interface already.

4.

[3] *(1, 2, 3)* Only the last argument(s) of a method may be removed, as PHP does not care about additional arguments that you pass to a method.

5.

[4] When changing the parent class, the original parent class must remain an ancestor of the class.

6.

[5] *(1, 2)* The value of a constant may only be changed when the constants aren't used in configuration (e.g. Yaml and XML files), as these do not support constants and have to hardcode the value. For instance, event name constants can't change the value without introducing a BC break. Additionally, if a constant will likely be used in objects that are serialized, the value of a constant should not be changed.

Chapter 9
Experimental Features

All Symfony features benefit from our *Backward Compatibility Promise* to give developers the confidence to upgrade to new versions safely and more often.

But sometimes, a new feature is controversial. Or finding a good API is not easy. In such cases, we prefer to gather feedback from real-world usage, adapt the API, or remove it altogether. Doing so is not possible with a no BC-break approach.

To avoid being bound to our backward compatibility promise, such features can be marked as **experimental** and their classes and methods must be marked with the `@experimental` tag.

A feature can be marked as being experimental for only one minor version, and can never be introduced in an LTS version. The core team can decide to extend the experimental period for another minor version on a case by case basis.

To ease upgrading projects using experimental features, the changelog must explain backward incompatible changes and explain how to upgrade code.

Chapter 10
Coding Standards

When contributing code to Symfony, you must follow its coding standards. To make a long story short, here is the golden rule: **Imitate the existing Symfony code**. Most open-source Bundles and libraries used by Symfony also follow the same guidelines, and you should too.

Remember that the main advantage of standards is that every piece of code looks and feels familiar, it's not about this or that being more readable.

Symfony follows the standards defined in the *PSR-0*[1], *PSR-1*[2], *PSR-2*[3] and *PSR-4*[4] documents.

Since a picture - or some code - is worth a thousand words, here's a short example containing most features described below:

Listing 10-1

```php
<?php

/*
 * This file is part of the Symfony package.
 *
 * (c) Fabien Potencier <fabien@symfony.com>
 *
 * For the full copyright and license information, please view the LICENSE
 * file that was distributed with this source code.
 */

namespace Acme;

/**
 * Coding standards demonstration.
 */
class FooBar
{
    const SOME_CONST = 42;

    /**
     * @var string
     */
    private $fooBar;

```

1. http://www.php-fig.org/psr/psr-0/
2. http://www.php-fig.org/psr/psr-1/
3. http://www.php-fig.org/psr/psr-2/
4. http://www.php-fig.org/psr/psr-4/

```php
    /**
     * @param string $dummy Some argument description
     */
    public function __construct($dummy)
    {
        $this->fooBar = $this->transformText($dummy);
    }

    /**
     * @return string
     *
     * @deprecated
     */
    public function someDeprecatedMethod()
    {
        @trigger_error(sprintf('The %s() method is deprecated since version 2.8 and will be removed in 3.0.
Use Acme\Baz::someMethod() instead.', __METHOD__), E_USER_DEPRECATED);

        return Baz::someMethod();
    }

    /**
     * Transforms the input given as first argument.
     *
     * @param bool|string $dummy   Some argument description
     * @param array       $options An options collection to be used within the transformation
     *
     * @return string|null The transformed input
     *
     * @throws \RuntimeException When an invalid option is provided
     */
    private function transformText($dummy, array $options = array())
    {
        $defaultOptions = array(
            'some_default' => 'values',
            'another_default' => 'more values',
        );

        foreach ($options as $option) {
            if (!in_array($option, $defaultOptions)) {
                throw new \RuntimeException(sprintf('Unrecognized option "%s"', $option));
            }
        }

        $mergedOptions = array_merge(
            $defaultOptions,
            $options
        );

        if (true === $dummy) {
            return null;
        }

        if ('string' === $dummy) {
            if ('values' === $mergedOptions['some_default']) {
                return substr($dummy, 0, 5);
            }

            return ucwords($dummy);
        }
    }

    /**
     * Performs some basic check for a given value.
     *
     * @param mixed $value     Some value to check against
     * @param bool  $theSwitch Some switch to control the method's flow
     *
     * @return bool|void The resultant check if $theSwitch isn't false, void otherwise
     */
    private function reverseBoolean($value = null, $theSwitch = false)
```

```
97      {
98          if (!$theSwitch) {
99              return;
100         }
101
102         return !$value;
103     }
    }
```

Structure

- Add a single space after each comma delimiter;
- Add a single space around binary operators (`==`, `&&`, ...), with the exception of the concatenation (`.`) operator;
- Place unary operators (`!`, `--`, ...) adjacent to the affected variable;
- Always use *identical comparison*[5] unless you need type juggling;
- Use *Yoda conditions*[6] when checking a variable against an expression to avoid an accidental assignment inside the condition statement (this applies to `==`, `!=`, `===`, and `!==`);
- Add a comma after each array item in a multi-line array, even after the last one;
- Add a blank line before `return` statements, unless the return is alone inside a statement-group (like an `if` statement);
- Use `return null;` when a function explicitly returns `null` values and use `return;` when the function returns `void` values;
- Use braces to indicate control structure body regardless of the number of statements it contains;
- Define one class per file - this does not apply to private helper classes that are not intended to be instantiated from the outside and thus are not concerned by the *PSR-0*[7] and *PSR-4*[8] autoload standards;
- Declare the class inheritance and all the implemented interfaces on the same line as the class name;
- Declare class properties before methods;
- Declare public methods first, then protected ones and finally private ones. The exceptions to this rule are the class constructor and the `setUp()` and `tearDown()` methods of PHPUnit tests, which must always be the first methods to increase readability;
- Declare all the arguments on the same line as the method/function name, no matter how many arguments there are;
- Use parentheses when instantiating classes regardless of the number of arguments the constructor has;
- Exception and error message strings must be concatenated using *`sprintf`*[9];
- Calls to *`trigger_error`*[10] with type `E_USER_DEPRECATED` must be switched to opt-in via `@` operator. Read more at Deprecations;
- Do not use `else`, `elseif`, `break` after `if` and `case` conditions which return or throw something;
- Do not use spaces around `[` offset accessor and before `]` offset accessor.

Naming Conventions

- Use camelCase, not underscores, for variable, function and method names, arguments;
- Use underscores for configuration options and parameters;
- Use namespaces for all classes;

---

5. http://php.net/manual/en/language.operators.comparison.php
6. https://en.wikipedia.org/wiki/Yoda_conditions
7. http://www.php-fig.org/psr/psr-0/
8. http://www.php-fig.org/psr/psr-4/
9. http://php.net/manual/en/function.sprintf.php
10. http://php.net/manual/en/function.trigger-error.php

- Prefix abstract classes with `Abstract`. Please note some early Symfony classes do not follow this convention and have not been renamed for backward compatibility reasons. However all new abstract classes must follow this naming convention;
- Suffix interfaces with `Interface`;
- Suffix traits with `Trait`;
- Suffix exceptions with `Exception`;
- Use alphanumeric characters and underscores for file names;
- For type-hinting in PHPDocs and casting, use `bool` (instead of `boolean` or `Boolean`), `int` (instead of `integer`), `float` (instead of `double` or `real`);
- Don't forget to look at the more verbose *Conventions* document for more subjective naming considerations.

Service Naming Conventions

- A service name contains groups, separated by dots;
- The DI alias of the bundle is the first group (e.g. `fos_user`);
- Use lowercase letters for service and parameter names (except when referring to environment variables with the `%env(VARIABLE_NAME)%` syntax);
- A group name uses the underscore notation.

Documentation

- Add PHPDoc blocks for all classes, methods, and functions (though you may be asked to remove PHPDoc that do not add value);
- Group annotations together so that annotations of the same type immediately follow each other, and annotations of a different type are separated by a single blank line;
- Omit the `@return` tag if the method does not return anything;
- The `@package` and `@subpackage` annotations are not used.

License

- Symfony is released under the MIT license, and the license block has to be present at the top of every PHP file, before the namespace.

Chapter 11
Conventions

The *Coding Standards* document describes the coding standards for the Symfony projects and the internal and third-party bundles. This document describes coding standards and conventions used in the core framework to make it more consistent and predictable. You are encouraged to follow them in your own code, but you don't need to.

Method Names

When an object has a "main" many relation with related "things" (objects, parameters, ...), the method names are normalized:

- `get()`
- `set()`
- `has()`
- `all()`
- `replace()`
- `remove()`
- `clear()`
- `isEmpty()`
- `add()`
- `register()`
- `count()`
- `keys()`

The usage of these methods are only allowed when it is clear that there is a main relation:

- a `CookieJar` has many `Cookie` objects;
- a Service `Container` has many services and many parameters (as services is the main relation, the naming convention is used for this relation);
- a Console `Input` has many arguments and many options. There is no "main" relation, and so the naming convention does not apply.

For many relations where the convention does not apply, the following methods must be used instead (where XXX is the name of the related thing):

Main Relation	Other Relations
`get()`	`getXXX()`
`set()`	`setXXX()`
n/a	`replaceXXX()`
`has()`	`hasXXX()`
`all()`	`getXXXs()`
`replace()`	`setXXXs()`
`remove()`	`removeXXX()`
`clear()`	`clearXXX()`
`isEmpty()`	`isEmptyXXX()`
`add()`	`addXXX()`
`register()`	`registerXXX()`
`count()`	`countXXX()`
`keys()`	n/a

While "setXXX" and "replaceXXX" are very similar, there is one notable difference: "setXXX" may replace, or add new elements to the relation. "replaceXXX", on the other hand, cannot add new elements. If an unrecognized key is passed to "replaceXXX" it must throw an exception.

Deprecations

From time to time, some classes and/or methods are deprecated in the framework; that happens when a feature implementation cannot be changed because of backward compatibility issues, but we still want to propose a "better" alternative. In that case, the old implementation can simply be **deprecated**.

A feature is marked as deprecated by adding a `@deprecated` phpdoc to relevant classes, methods, properties, ...:

Listing 11-1
```
/**
 * @deprecated since version 2.8, to be removed in 3.0. Use XXX instead.
 */
```

The deprecation message should indicate the version when the class/method was deprecated, the version when it will be removed, and whenever possible, how the feature was replaced.

A PHP `E_USER_DEPRECATED` error must also be triggered to help people with the migration starting one or two minor versions before the version where the feature will be removed (depending on the criticality of the removal):

Listing 11-2
```
@trigger_error('XXX() is deprecated since version 2.8 and will be removed in 3.0. Use XXX instead.',
E_USER_DEPRECATED);
```

Without the *@-silencing operator*[1], users would need to opt-out from deprecation notices. Silencing swaps this behavior and allows users to opt-in when they are ready to cope with them (by adding a custom error handler like the one used by the Web Debug Toolbar or by the PHPUnit bridge).

When deprecating a whole class the `trigger_error()` call should be placed between the namespace and the use declarations, like in this example from *ArrayParserCache*[2]:

1. https://php.net/manual/en/language.operators.errorcontrol.php

Listing 11-3

```php
namespace Symfony\Component\ExpressionLanguage\ParserCache;

@trigger_error('The '.__NAMESPACE__.'\ArrayParserCache class is deprecated since version 3.2 and will be
removed in 4.0. Use the Symfony\Component\Cache\Adapter\ArrayAdapter class instead.', E_USER_DEPRECATED);

use Symfony\Component\ExpressionLanguage\ParsedExpression;

/**
 * @author Adrien Brault <adrien.brault@gmail.com>
 *
 * @deprecated ArrayParserCache class is deprecated since version 3.2 and will be removed in 4.0. Use the
Symfony\Component\Cache\Adapter\ArrayAdapter class instead.
 */
class ArrayParserCache implements ParserCacheInterface
```

2. https://github.com/symfony/symfony/blob/3.2/src/Symfony/Component/ExpressionLanguage/ParserCache/ArrayParserCache.php

Chapter 12
Git

This document explains some conventions and specificities in the way we manage the Symfony code with Git.

Pull Requests

Whenever a pull request is merged, all the information contained in the pull request (including comments) is saved in the repository.

You can easily spot pull request merges as the commit message always follows this pattern:

Listing 12-1
```
merged branch USER_NAME/BRANCH_NAME (PR #1111)
```

The PR reference allows you to have a look at the original pull request on GitHub: *https://github.com/symfony/symfony/pull/1111*. But all the information you can get on GitHub is also available from the repository itself.

The merge commit message contains the original message from the author of the changes. Often, this can help understand what the changes were about and the reasoning behind the changes.

Moreover, the full discussion that might have occurred back then is also stored as a Git note (before March 22 2013, the discussion was part of the main merge commit message). To get access to these notes, add this line to your `.git/config` file:

Listing 12-2
```
fetch = +refs/notes/*:refs/notes/*
```

After a fetch, getting the GitHub discussion for a commit is then a matter of adding `--show-notes=github-comments` to the `git show` command:

Listing 12-3
```
$ git show HEAD --show-notes=github-comments
```

Chapter 13
Symfony License

Symfony is released under the MIT license.

According to *Wikipedia*[1]:

> "It is a permissive license, meaning that it permits reuse within proprietary software on the condition that the license is distributed with that software. The license is also GPL-compatible, meaning that the GPL permits combination and redistribution with software that uses the MIT License."

The License

1. https://en.wikipedia.org/wiki/MIT_License

Chapter 14
Documentation Format

The Symfony documentation uses *reStructuredText*[1] as its markup language and *Sphinx*[2] for generating the documentation in the formats read by the end users, such as HTML and PDF.

reStructuredText

reStructuredText is a plaintext markup syntax similar to Markdown, but much stricter with its syntax. If you are new to reStructuredText, take some time to familiarize with this format by reading the existing *Symfony documentation*[3] source code.

If you want to learn more about this format, check out the *reStructuredText Primer*[4] tutorial and the *reStructuredText Reference*[5].

If you are familiar with Markdown, be careful as things are sometimes very similar but different:

- Lists starts at the beginning of a line (no indentation is allowed);
- Inline code blocks use double-ticks (``like this``).

Sphinx

Sphinx[6] is a build system that provides tools to create documentation from reStructuredText documents. As such, it adds new directives and interpreted text roles to the standard reST markup. Read more about the *Sphinx Markup Constructs*[7].

1. http://docutils.sourceforge.net/rst.html
2. http://sphinx-doc.org/
3. https://github.com/symfony/symfony-docs
4. http://sphinx-doc.org/rest.html
5. http://docutils.sourceforge.net/docs/user/rst/quickref.html
6. http://sphinx-doc.org/
7. http://sphinx-doc.org/markup/

Syntax Highlighting

PHP is the default syntax highlighter applied to all code blocks. You can change it with the `code-block` directive:

Listing 14-1
```
.. code-block:: yaml

    { foo: bar, bar: { foo: bar, bar: baz } }
```

Besides all of the major programming languages, the syntax highlighter supports all kinds of markup and configuration languages. Check out the list of *supported languages*[8] on the syntax highlighter website.

Configuration Blocks

Whenever you include a configuration sample, use the `configuration-block` directive to show the configuration in all supported configuration formats (`PHP`, `YAML` and `XML`). Example:

Listing 14-2
```
.. configuration-block::

    .. code-block:: yaml

        # Configuration in YAML

    .. code-block:: xml

        <!-- Configuration in XML -->

    .. code-block:: php

        // Configuration in PHP
```

The previous reST snippet renders as follow:

Listing 14-3
```
# Configuration in YAML
```

The current list of supported formats are the following:

Markup Format	Use It to Display
`html`	HTML
`xml`	XML
`php`	PHP
`yaml`	YAML
`twig`	Pure Twig markup
`html+twig`	Twig markup blended with HTML
`html+php`	PHP code blended with HTML
`ini`	INI
`php-annotations`	PHP Annotations

8. `http://pygments.org/languages/`

Adding Links

The most common type of links are **internal links** to other documentation pages, which use the following syntax:

Listing 14-4
```
:doc:`/absolute/path/to/page`
```

The page name should not include the file extension (`.rst`). For example:

Listing 14-5
```
:doc:`/controller`

:doc:`/components/event_dispatcher`

:doc:`/configuration/environments`
```

The title of the linked page will be automatically used as the text of the link. If you want to modify that title, use this alternative syntax:

Listing 14-6
```
:doc:`Spooling Email </email/spool>`
```

Although they are technically correct, avoid the use of relative internal links such as the following, because they break the references in the generated PDF documentation:

Listing 14-7
```
:doc:`controller`

:doc:`event_dispatcher`

:doc:`environments`
```

Links to the API follow a different syntax, where you must specify the type of the linked resource (`namespace`, `class` or `method`):

Listing 14-8
```
:namespace:`Symfony\\Component\\BrowserKit`

:class:`Symfony\\Component\\Routing\\Matcher\\ApacheUrlMatcher`

:method:`Symfony\\Component\\HttpKernel\\Bundle\\Bundle::build`
```

Links to the PHP documentation follow a pretty similar syntax:

Listing 14-9
```
:phpclass:`SimpleXMLElement`

:phpmethod:`DateTime::createFromFormat`

:phpfunction:`iterator_to_array`
```

New Features or Behavior Changes

If you're documenting a brand new feature or a change that's been made in Symfony, you should precede your description of the change with a `.. versionadded:: 2.X` directive and a short description:

Listing 14-10
```
.. versionadded:: 2.7
    The ``askHiddenResponse()`` method was introduced in Symfony 2.7.

You can also ask a question and hide the response. This is particularly [...]
```

If you're documenting a behavior change, it may be helpful to *briefly* describe how the behavior has changed:

Listing 14-11
```
.. versionadded:: 2.7
    The ``include()`` function is a new Twig feature that's available in
    Symfony 2.7. Prior, the ``{% include %}`` tag was used.
```

Whenever a new minor version of Symfony is released (e.g. 2.4, 2.5, etc), a new branch of the documentation is created from the `master` branch. At this point, all the `versionadded` tags for Symfony versions that have reached end-of-maintenance will be removed. For example, if Symfony 2.5 were released today, and 2.2 had recently reached its end-of-maintenance, the 2.2 `versionadded` tags would be removed from the new `2.5` branch.

Chapter 15
Symfony Documentation License

The Symfony documentation is licensed under a Creative Commons Attribution-Share Alike 3.0 Unported License (*CC BY-SA 3.0*[1]).

You are free:

- to *Share* — to copy, distribute and transmit the work;
- to *Remix* — to adapt the work.

Under the following conditions:

- *Attribution* — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work);
- *Share Alike* — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

With the understanding that:

- *Waiver* — Any of the above conditions can be waived if you get permission from the copyright holder;
- *Public Domain* — Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license;
- *Other Rights* — In no way are any of the following rights affected by the license:
    - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
    - The author's moral rights;
    - Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.
- *Notice* — For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

This is a human-readable summary of the *Legal Code (the full license)*[2].

1. http://creativecommons.org/licenses/by-sa/3.0/
2. http://creativecommons.org/licenses/by-sa/3.0/legalcode

Chapter 16

Contributing to the Documentation

Before Your First Contribution

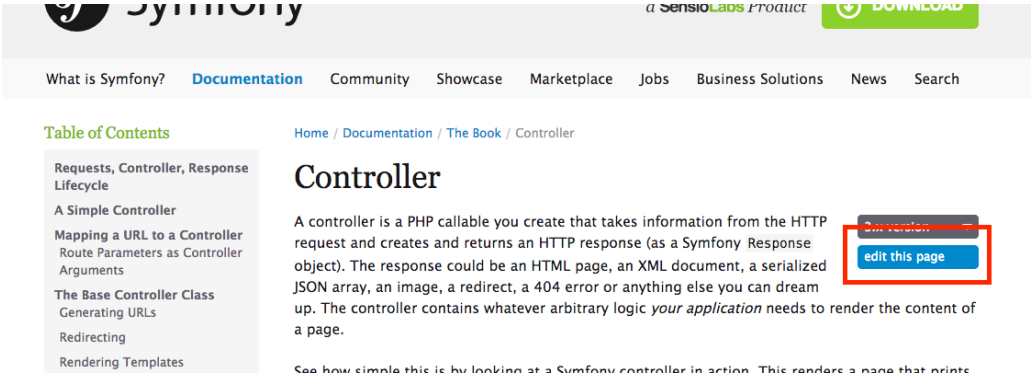
Before contributing, you need to:

- Sign up for a free *GitHub*[1] account, which is the service where the Symfony documentation is hosted.
- Be familiar with the *reStructuredText*[2] markup language, which is used to write Symfony docs. Read *this article* for a quick overview.

Fast Online Contributions

If you're making a relatively small change - like fixing a typo or rewording something - the easiest way to contribute is directly on GitHub! You can do this while you're reading the Symfony documentation.

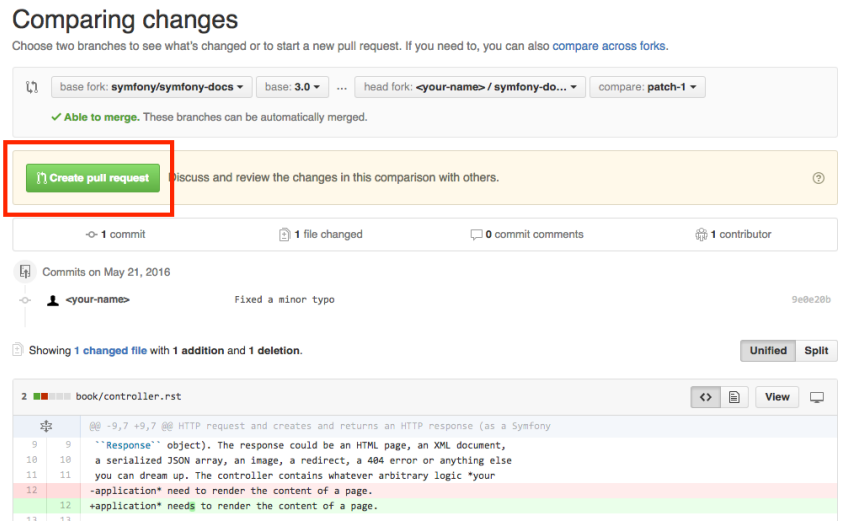
Step 1. Click on the **edit this page** button on the upper right corner and you'll be redirected to GitHub:



1. https://github.com/
2. http://docutils.sourceforge.net/rst.html

Step 2. Edit the contents, describe your changes and click on the **Propose file change** button.

Step 3. GitHub will now create a branch and a commit for your changes (forking the repository first if this is your first contribution) and it will also display a preview of your changes:



If everything is correct, click on the **Create pull request** button.

Step 4. GitHub will display a new page where you can do some last-minute changes to your pull request before creating it. For simple contributions, you can safely ignore these options and just click on the **Create pull request** button again.

Congratulations! You just created a pull request to the official Symfony documentation! The community will now review your pull request and (possibly) suggest tweaks.

If your contribution is large or if you prefer to work on your own computer, keep reading this guide to learn an alternative way to send pull requests to the Symfony Documentation.

Your First Documentation Contribution

In this section, you'll learn how to contribute to the Symfony documentation for the first time. The next section will explain the shorter process you'll follow in the future for every contribution after your first one.

Let's imagine that you want to improve the Setup guide. In order to make your changes, follow these steps:

Step 1. Go to the official Symfony documentation repository located at *github.com/symfony/symfony-docs*[3] and click on the **Fork** button to fork the repository to your personal account. This is only needed the first time you contribute to Symfony.

Step 2. Clone the forked repository to your local machine (this example uses the `projects/symfony-docs/` directory to store the documentation; change this value accordingly):

Listing 16-1
```
$ cd projects/
$ git clone git://github.com/YOUR-GITHUB-USERNAME/symfony-docs.git
```

Step 3. Add the original Symfony docs repository as a "Git remote" executing this command:

Listing 16-2

3. `https://github.com/symfony/symfony-docs`

```
$ cd symfony-docs/
$ git remote add upstream https://github.com/symfony/symfony-docs.git
```

If things went right, you'll see the following when listing the "remotes" of your project:

Listing 16-3
```
$ git remote -v
origin  git@github.com:YOUR-GITHUB-USERNAME/symfony-docs.git (fetch)
origin  git@github.com:YOUR-GITHUB-USERNAME/symfony-docs.git (push)
upstream  https://github.com/symfony/symfony-docs.git (fetch)
upstream  https://github.com/symfony/symfony-docs.git (push)
```

Fetch all the commits of the upstream branches by executing this command:

Listing 16-4
```
$ git fetch upstream
```

The purpose of this step is to allow you work simultaneously on the official Symfony repository and on your own fork. You'll see this in action in a moment.

Step 4. Create a dedicated **new branch** for your changes. Use a short and memorable name for the new branch (if you are fixing a reported issue, use `fix_XXX` as the branch name, where `XXX` is the number of the issue):

Listing 16-5
```
$ git checkout -b improve_install_article upstream/2.7
```

In this example, the name of the branch is `improve_install_article` and the `upstream/2.7` value tells Git to create this branch based on the `2.7` branch of the `upstream` remote, which is the original Symfony Docs repository.

Fixes should always be based on the **oldest maintained branch** which contains the error. Nowadays this is the `2.7` branch. If you are instead documenting a new feature, switch to the first Symfony version that included it, e.g. `upstream/3.1`. Not sure? That's ok! Just use the `upstream/master` branch.

Step 5. Now make your changes in the documentation. Add, tweak, reword and even remove any content and do your best to comply with the *Documentation Standards*. Then commit your changes!

Listing 16-6
```
# if the modified content existed before
$ git add setup.rst
$ git commit setup.rst
```

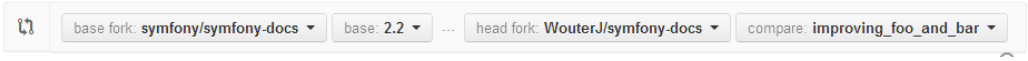
Step 6. Push the changes to your forked repository:

Listing 16-7
```
$ git push origin improve_install_article
```

The `origin` value is the name of the Git remote that corresponds to your forked repository and `improve_install_article` is the name of the branch you created previously.

Step 7. Everything is now ready to initiate a **pull request**. Go to your forked repository at `https//github.com/YOUR-GITHUB-USERNAME/symfony-docs` and click on the **Pull Requests** link located in the sidebar.

Then, click on the big **New pull request** button. As GitHub cannot guess the exact changes that you want to propose, select the appropriate branches where changes should be applied:



In this example, the **base fork** should be `symfony/symfony-docs` and the **base** branch should be the `2.7`, which is the branch that you selected to base your changes on. The **head fork** should be your forked

copy of `symfony-docs` and the **compare** branch should be `improve_install_article`, which is the name of the branch you created and where you made your changes.

Step 8. The last step is to prepare the **description** of the pull request. A short phrase or paragraph describing the proposed changes is enough to ensure that your contribution can be reviewed.

Step 9. Now that you've successfully submitted your first contribution to the Symfony documentation, **go and celebrate!** The documentation managers will carefully review your work in short time and they will let you know about any required change.

In case you are asked to add or modify something, don't create a new pull request. Instead, make sure that you are on the correct branch, make your changes and push the new changes:

Listing 16-8
```
$ cd projects/symfony-docs/
$ git checkout improve_install_article

# ... do your changes

$ git push
```

Step 10. After your pull request is eventually accepted and merged in the Symfony documentation, you will be included in the Symfony Documentation Contributors list. Moreover, if you happen to have a *SensioLabsConnect*[4] profile, you will get a cool *Symfony Documentation Badge*[5].

Your Next Documentation Contributions

Check you out! You've made your first contribution to the Symfony documentation! Somebody throw a party! Your first contribution took a little extra time because you needed to learn a few standards and setup your computer. But from now on, your contributions will be much easier to complete.

Here is a **checklist** of steps that will guide you through your next contribution to the Symfony docs:

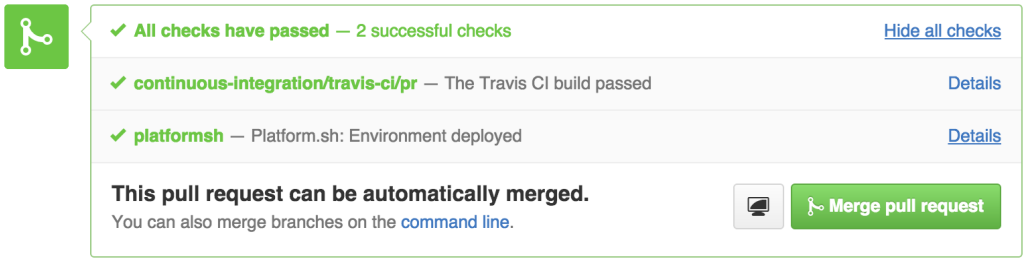
Listing 16-9
```
# create a new branch based on the oldest maintained version
$ cd projects/symfony-docs/
$ git fetch upstream
$ git checkout -b my_changes upstream/2.7

# ... do your changes

# (optional) add your changes if this is a new content
$ git add xxx.rst

# commit your changes and push them to your fork
$ git commit xxx.rst
$ git push origin my_changes

# ... go to GitHub and create the Pull Request

# (optional) make the changes requested by reviewers and commit them
$ git commit xxx.rst
$ git push
```

After completing your next contributions, also watch your ranking improve on the list of *Symfony Documentation Contributors*[6]. You guessed right: after all this hard work, it's **time to celebrate again!**

4. https://connect.sensiolabs.com/
5. https://connect.sensiolabs.com/badge/36/symfony-documentation-contributor
6. https://symfony.com/contributors/doc

Review your changes

Every GitHub Pull Request is automatically built and deployed by *Platform.sh*[7] on a single environment that you can access on your browser to review your changes.



To access the *Platform.sh*[8] environment URL, go to your Pull Request page on GitHub, click on the **Show all checks** link and finally, click on the `Details` link displayed for Platform.sh service.

Only Pull Requests to maintained branches are automatically built by Platform.sh. Check the *roadmap*[9] for maintained branches.

Build the Documentation Locally

Alternatively you can build the documentation on your own computer for testing purposes following these steps:

1. Install *pip*[10] as explained in the *pip installation*[11] article;
2. Install *Sphinx*[12] and *Sphinx Extensions for PHP and Symfony*[13] (depending on your system, you may need to execute this command as root user):

Listing 16-10
```
$ pip install sphinx~=1.3.0 git+https://github.com/fabpot/sphinx-php.git
```

3. Run the following command to build the documentation in HTML format:

Listing 16-11
```
$ cd _build/
$ make html
```

The generated documentation is available in the `_build/html` directory.

7. https://platform.sh
8. https://platform.sh
9. https://symfony.com/roadmap
10. https://pip.pypa.io/en/stable/
11. https://pip.pypa.io/en/stable/installing/
12. http://sphinx-doc.org/
13. https://github.com/fabpot/sphinx-php

Frequently Asked Questions

Why Do my Changes Take so Long to Be Reviewed and/or Merged?

Please be patient. It can take up to several days before your pull request can be fully reviewed. After merging the changes, it could take again several hours before your changes appear on the symfony.com website.

Why Should I Use the Oldest Maintained Branch Instead of the Master Branch?

Consistent with Symfony's source code, the documentation repository is split into multiple branches, corresponding to the different versions of Symfony itself. The `master` branch holds the documentation for the development branch of the code.

Unless you're documenting a feature that was introduced after Symfony 2.7, your changes should always be based on the `2.7` branch. Documentation managers will use the necessary Git-magic to also apply your changes to all the active branches of the documentation.

What If I Want to Submit my Work without Fully Finishing It?

You can do it. But please use one of these two prefixes to let reviewers know about the state of your work:

- `[WIP]` (Work in Progress) is used when you are not yet finished with your pull request, but you would like it to be reviewed. The pull request won't be merged until you say it is ready.
- `[WCM]` (Waiting Code Merge) is used when you're documenting a new feature or change that hasn't been accepted yet into the core code. The pull request will not be merged until it is merged in the core code (or closed if the change is rejected).

Would You Accept a Huge Pull Request with Lots of Changes?

First, make sure that the changes are somewhat related. Otherwise, please create separate pull requests. Anyway, before submitting a huge change, it's probably a good idea to open an issue in the Symfony Documentation repository to ask the managers if they agree with your proposed changes. Otherwise, they could refuse your proposal after you put all that hard work into making the changes. We definitely don't want you to waste your time!

Chapter 17
Documentation Standards

Contributions must follow these standards to match the style and tone of the rest of the Symfony documentation.

Sphinx

- The following characters are chosen for different heading levels: level 1 is `=` (equal sign), level 2 `-` (dash), level 3 `~` (tilde), level 4 `.` (dot) and level 5 `"` (double quote);
- Each line should break approximately after the first word that crosses the 72nd character (so most lines end up being 72-78 characters);
- The `::` shorthand is *preferred* over `.. code-block:: php` to begin a PHP code block (read *the Sphinx documentation*[1] to see when you should use the shorthand);
- Inline hyperlinks are **not** used. Separate the link and their target definition, which you add on the bottom of the page;
- Inline markup should be closed on the same line as the open-string;

Example

Listing 17-1

```
Example
=======

When you are working on the docs, you should follow the
`Symfony Documentation`_ standards.

Level 2
-------

A PHP example would be::

    echo 'Hello World';

Level 3
~~~~~~~

```

1. http://sphinx-doc.org/rest.html#source-code

```
.. code-block:: php

    echo 'You cannot use the :: shortcut here';

.. _`Symfony Documentation`: https://symfony.com/doc
```

Code Examples

- The code follows the *Symfony Coding Standards* as well as the *Twig Coding Standards*[2];
- The code examples should look real for a web application context. Avoid abstract or trivial examples (`foo`, `bar`, `demo`, etc.);
- The code should follow the *Symfony Best Practices*. Unless the example requires a custom bundle, make sure to always use the `AppBundle` bundle to store your code;
- Use `Acme` when the code requires a vendor name;
- Use `example.com` as the domain of sample URLs and `example.org` and `example.net` when additional domains are required. All of these domains are *reserved by the IANA*[3].
- To avoid horizontal scrolling on code blocks, we prefer to break a line correctly if it crosses the 85th character;
- When you fold one or more lines of code, place `...` in a comment at the point of the fold. These comments are: `// ...` (php), `# ...` (yaml/bash), `{# ... #}` (twig), `<!-- ... -->` (xml/html), `; ...` (ini), `...` (text);
- When you fold a part of a line, e.g. a variable value, put `...` (without comment) at the place of the fold;
- Description of the folded code: (optional)
    - If you fold several lines: the description of the fold can be placed after the `...`;
    - If you fold only part of a line: the description can be placed before the line;
- If useful to the reader, a PHP code example should start with the namespace declaration;
- When referencing classes, be sure to show the `use` statements at the top of your code block. You don't need to show *all* `use` statements in every example, just show what is actually being used in the code block;
- If useful, a `codeblock` should begin with a comment containing the filename of the file in the code block. Don't place a blank line after this comment, unless the next line is also a comment;
- You should put a `$` in front of every bash line.

Formats

Configuration examples should show all supported formats using configuration blocks. The supported formats (and their orders) are:

- **Configuration** (including services): YAML, XML, PHP
- **Routing**: Annotations, YAML, XML, PHP
- **Validation**: Annotations, YAML, XML, PHP
- **Doctrine Mapping**: Annotations, YAML, XML, PHP
- **Translation**: XML, YAML, PHP

Example

Listing 17-2

2. `http://twig.sensiolabs.org/doc/coding_standards.html`
3. `http://tools.ietf.org/html/rfc2606#section-3`

```php
// src/Foo/Bar.php
namespace Foo;

use Acme\Demo\Cat;
// ...

class Bar
{
    // ...

    public function foo($bar)
    {
        // set foo with a value of bar
        $foo = ...;

        $cat = new Cat($foo);

        // ... check if $bar has the correct value

        return $cat->baz($bar, ...);
    }
}
```

In YAML you should put a space after `{` and before `}` (e.g. `{ _controller: ... }`), but this should not be done in Twig (e.g. `{'hello' : 'value'}`).

Files and Directories

- When referencing directories, always add a trailing slash to avoid confusions with regular files (e.g. "execute the `console` script located at the `app/` directory").
- When referencing file extensions explicitly, you should include a leading dot for every extension (e.g. "XML files use the `.xml` extension").
- When you list a Symfony file/directory hierarchy, use `your-project/` as the top level directory. E.g.

Listing 17-3
```
your-project/
├─ app/
├─ src/
├─ vendor/
└─ ...
```

English Language Standards

Symfony documentation uses the United States English dialect, commonly called *American English*[4]. The *American English Oxford Dictionary*[5] is used as the vocabulary reference.

In addition, documentation follows these rules:

- **Section titles**: use a variant of the title case, where the first word is always capitalized and all other words are capitalized, except for the closed-class words (read Wikipedia article about *headings and titles*[6]).

4. https://en.wikipedia.org/wiki/American_English
5. http://en.oxforddictionaries.com/definition/american_english/
6. https://en.wikipedia.org/wiki/Letter_case#Headings_and_publication_titles

E.g.: The Vitamins are in my Fresh California Raisins

- **Punctuation**: avoid the use of *Serial (Oxford) Commas*[7];
- **Pronouns**: avoid the use of *nosism*[8] and always use *you* instead of *we*. (i.e. avoid the first person point of view: use the second instead);
- **Gender-neutral language**: when referencing a hypothetical person, such as *"a user with a session cookie"*, use gender-neutral pronouns (they/their/them). For example, instead of:

    - he or she, use they
    - him or her, use them
    - his or her, use their
    - his or hers, use theirs
    - himself or herself, use themselves

7. https://en.wikipedia.org/wiki/Serial_comma
8. https://en.wikipedia.org/wiki/Nosism

Chapter 18
Translations

The official Symfony documentation is published only in English, but some community groups maintain *unofficial translations*[1].

1. https://symfony.com/blog/discontinuing-the-symfony-community-translations

Chapter 19
The Release Process

This document explains the **release process** of the Symfony project (i.e. the code hosted on the main `symfony/symfony` *Git repository*[1]).

Symfony manages its releases through a *time-based model* and follows the *Semantic Versioning*[2] strategy:

- A new Symfony minor version (e.g. 2.8, 3.2, 4.1) comes out every *six months*: one in *May* and one in *November*;
- A new Symfony major version (e.g., 3.0, 4.0) comes out every *two years* and it's released at the same time of the last minor version of the previous major version.

Development

The full development period for any major or minor version lasts six months and is divided into two phases:

- **Development**: *Four months* to add new features and to enhance existing ones;
- **Stabilization**: *Two months* to fix bugs, prepare the release, and wait for the whole Symfony ecosystem (third-party libraries, bundles, and projects using Symfony) to catch up.

During the development phase, any new feature can be reverted if it won't be finished in time or if it won't be stable enough to be included in the current final release.

Maintenance

Each Symfony version is maintained for a fixed period of time, depending on the type of the release. This maintenance is divided into:

- *Bug fixes and security fixes*: During this period, all issues can be fixed. The end of this period is referenced as being the *end of maintenance* of a release.

1. `https://github.com/symfony/symfony`
2. `http://semver.org/`

- *Security fixes only*: During this period, only security related issues can be fixed. The end of this period is referenced as being the *end of life* of a release.

The *maintenance document* describes the boundaries of acceptable changes during maintenance.

Symfony Versions

Standard Versions

A **Standard Minor Version** is maintained for an *eight month* period for bug fixes, and for a *fourteen month* period for security issue fixes.

In Symfony 2.x branch, the number of minor versions wasn't constrained, so that branch ended up with nine minor versions (from 2.0 to 2.8). Starting from 3.x branch, the number of minor versions is limited to five (from X.0 to X.4).

Long Term Support Versions

Every two years, a new **Long Term Support Version** (usually abbreviated as "LTS") is published. Each LTS version is supported for a *three year* period for bug fixes, and for a *four year* period for security issue fixes.

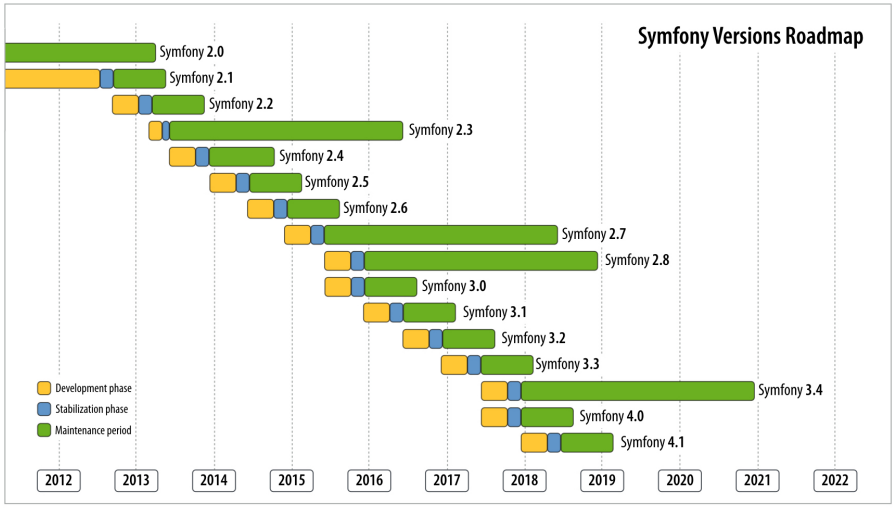
Paid support after the three year support provided by the community can also be bought from *SensioLabs*[3].

In the Symfony 2.x branch, the LTS versions are 2.3, 2.7 and 2.8. Starting from the 3.x branch, only the last minor version of each branch is considered LTS (e.g. 3.4, 4.4, 5.4, etc.)

Schedule

Below is the schedule for the first few versions that use this release model:

3. `http://sensiolabs.com/`



- **Yellow** represents the Development phase
- **Blue** represents the Stabilization phase
- **Green** represents the Maintenance period

If you want to learn more about the timeline of any given Symfony version, use the online *timeline calculator*[4].

Whenever an important event related to Symfony versions happens (a version reaches end of maintenance or a new patch version is released for instance), you can automatically receive an email notification if you subscribed on the *roadmap notification*[5] page.

Version	Feature Freeze	Release	End of Maintenance	End of Life
2.0	05/2011	07/2011	03/2013 (20 months)	09/2013
2.1	07/2012	09/2012	05/2013 (9 months)	11/2013
2.2	01/2013	03/2013	11/2013 (8 months)	05/2014
2.3	03/2013	05/2013	05/2016 (36 months)	05/2017
2.4	09/2013	11/2013	09/2014 (10 months [1])	01/2015
2.5	03/2014	05/2014	01/2015 (8 months)	07/2015
2.6	09/2014	11/2014	07/2015 (8 months)	01/2016
2.7	03/2015	05/2015	05/2018 (36 months)	05/2019
2.8	09/2015	11/2015	11/2018 (36 months [2])	11/2019
3.0	09/2015	11/2015	07/2016 (8 months) [3])	01/2017
3.1	03/2016	05/2016	01/2017 (8 months)	07/2017
3.2	09/2016	11/2016	07/2017 (8 months)	01/2018
3.3	03/2017	05/2017	01/2018 (8 months)	07/2018
3.4	09/2017	11/2017	11/2020 (36 months)	11/2021

4. `https://symfony.com/roadmap#checker`
5. `https://symfony.com/roadmap`

Version	Feature Freeze	Release	End of Maintenance	End of Life
4.0	09/2017	11/2017	07/2018 (8 months)	01/2019
4.1	03/2018	05/2018	01/2019 (8 months)	07/2019
4.2	09/2018	11/2018	07/2019 (8 months)	01/2020
4.3	03/2019	05/2019	01/2020 (8 months)	07/2020
4.4	09/2019	11/2019	11/2022 (36 months)	11/2023
5.0	09/2019	11/2019	07/2020 (8 months)	01/2021
...	...	...	...	...

6 7 8

Backward Compatibility

Our *Backward Compatibility Promise* is very strict and allows developers to upgrade with confidence from one minor version of Symfony to the next one.

Whenever keeping backward compatibility is not possible, the feature, the enhancement or the bug fix will be scheduled for the next major version.

However, *Experimental Features* are **not** part of the backward compatibility promise and their APIs can change from one minor version to the next. The changelog must explain the changes and how to upgrade.

Deprecations

When a feature implementation cannot be replaced with a better one without breaking backward compatibility, there is still the possibility to deprecate the old implementation and add a new preferred one along side. Read the conventions document to learn more about how deprecations are handled in Symfony.

Rationale

This release process was adopted to give more *predictability* and *transparency*. It was discussed based on the following goals:

- Shorten the release cycle (allow developers to benefit from the new features faster);

---

6. [1] Symfony 2.4 maintenance has been *extended to September 2014*[9].

7. [2] Symfony 2.8 is the last version of the Symfony 2.x branch.

8. [3] Symfony 3.0 is the first version to use the new release process based on five minor releases.

9. `https://symfony.com/blog/extended-maintenance-for-symfony-2-4`

- Give more visibility to the developers using the framework and Open-Source projects using Symfony;
- Improve the experience of Symfony core contributors: everyone knows when a feature might be available in Symfony;
- Coordinate the Symfony timeline with popular PHP projects that work well with Symfony and with projects using Symfony;
- Give time to the Symfony ecosystem to catch up with the new versions (bundle authors, documentation writers, translators, ...).

The six month period was chosen as two releases fit in a year. It also allows for plenty of time to work on new features and it allows for non-ready features to be postponed to the next version without having to wait too long for the next cycle.

The dual maintenance mode was adopted to make every Symfony user happy. Fast movers, who want to work with the latest and the greatest, use the standard version: a new version is published every six months, and there is a two months period to upgrade. Companies wanting more stability use the LTS versions: a new version is published every two years and there is a year to upgrade.

Chapter 20
Community Reviews

Symfony is an open-source project driven by a large community. If you don't feel ready to contribute code or patches, reviewing issues and pull requests (PRs) can be a great start to get involved and give back. In fact, people who "triage" issues are the backbone to Symfony's success!

Why Reviewing Is Important

Community reviews are essential for the development of the Symfony framework, since there are many more pull requests and bug reports than there are members in the Symfony core team to review, fix and merge them.

On the *Symfony issue tracker*[1], you can find many items in a *Needs Review*[2] status:

- **Bug Reports**: Bug reports need to be checked for completeness. Is any important information missing? Can the bug be *easily* reproduced?
- **Pull Requests**: Pull requests contain code that fixes a bug or implements new functionality. Reviews of pull requests ensure that they are implemented properly, are covered by test cases, don't introduce new bugs and maintain backward compatibility.

Note that **anyone who has some basic familiarity with Symfony and PHP can review bug reports and pull requests**. You don't need to be an expert to help.

Be Constructive

Before you begin, remember that you are looking at the result of someone else's hard work. A good review comment thanks the contributor for their work, identifies what was done well, identifies what should be improved and suggests a next step.

1. `https://github.com/symfony/symfony/issues`
2. `https://github.com/symfony/symfony/labels/Status%3A%20Needs%20Review`

Create a GitHub Account

Symfony uses *GitHub*[3] to manage bug reports and pull requests. If you want to do reviews, you need to *create a GitHub account*[4] and log in.

The Bug Report Review Process

A good way to get started with reviewing is to pick a bug report from the *bug reports in need of review*[5].

The steps for the review are:

1. **Is the Report Complete?**

   Good bug reports contain a link to a fork of the *Symfony Standard Edition*[6] (the "reproduction project") that reproduces the bug. If it doesn't, the report should at least contain enough information and code samples to reproduce the bug.

2. **Reproduce the Bug**

   Download the reproduction project and test whether the bug can be reproduced on your system. If the reporter did not provide a reproduction project, create one by *forking*[7] the *Symfony Standard Edition*[8].

3. **Update the Issue Status**

   At last, add a comment to the bug report. **Thank the reporter for reporting the bug**. Include the line `Status: <status>` in your comment to trigger our *Carson Bot*[9] which updates the status label of the issue. You can set the status to one of the following:

   Needs Work If the bug *does not* contain enough information to be reproduced, explain what information is missing and move the report to this status.

   Works for me If the bug *does* contain enough information to be reproduced but works on your system, or if the reported bug is a feature and not a bug, provide a short explanation and move the report to this status.

   Reviewed If you can reproduce the bug, move the report to this status. If you created a reproduction project, include the link to the project in your comment.

Example

Here is a sample comment for a bug report that could be reproduced:

Listing 20-1
```
Thank you @weaverryan for creating this bug report! This indeed looks
like a bug. I reproduced the bug in the "kernel-bug" branch of
https://github.com/webmozart/symfony-standard.

Status: Reviewed
```

3. https://github.com
4. https://help.github.com/articles/signing-up-for-a-new-github-account/
5. https://github.com/symfony/symfony/issues?utf8=%E2%9C%93&q=is%3Aopen+is%3Aissue+label%3A%22Bug%22+label%3A%22Status%3A+Needs+Review%22+
6. https://github.com/symfony/symfony-standard
7. https://help.github.com/articles/fork-a-repo/
8. https://github.com/symfony/symfony-standard
9. https://github.com/carsonbot/carsonbot

The Pull Request Review Process

The process for reviewing pull requests (PRs) is similar to the one for bug reports. Reviews of pull requests usually take a little longer since you need to understand the functionality that has been fixed or added and find out whether the implementation is complete.

It is okay to do partial reviews! If you do a partial review, comment how far you got and leave the PR in "Needs Review" state.

Pick a pull request from the *PRs in need of review*[10] and follow these steps:

1. **Is the PR Complete**?

   Every pull request must contain a header that gives some basic information about the PR. You can find the template for that header in the Contribution Guidelines.

2. **Is the Base Branch Correct?**

   GitHub displays the branch that a PR is based on below the title of the pull request. Is that branch correct?

   - Bugs should be fixed in the oldest, maintained version that contains the bug. Check *Symfony's Release Schedule* to find the oldest currently supported version.
   - New features should always be added to the current development version. Check the *Symfony Roadmap*[11] to find the current development version.

3. **Reproduce the Problem**

   Read the issue that the pull request is supposed to fix. Reproduce the problem on a clean *Symfony Standard Edition*[12] project and try to understand why it exists. If the linked issue already contains such a project, install it and run it on your system.

4. **Review the Code**

   Read the code of the pull request and check it against some common criteria:

   - Does the code address the issue the PR is intended to fix/implement?
   - Does the PR stay within scope to address *only* that issue?
   - Does the PR contain automated tests? Do those tests cover all relevant edge cases?
   - Does the PR contain sufficient comments to easily understand its code?
   - Does the code break backward compatibility? If yes, does the PR header say so?
   - Does the PR contain deprecations? If yes, does the PR header say so? Does the code contain `trigger_error()` statements for all deprecated features?
   - Are all deprecations and backward compatibility breaks documented in the latest UPGRADE-X.X.md file? Do those explanations contain "Before"/"After" examples with clear upgrade instructions?

   Eventually, some of these aspects will be checked automatically.

5. **Test the Code**

   Take your project from step 3 and test whether the PR works properly. Replace the Symfony project in the `vendor` directory by the code in the PR by running the following Git commands. Insert the PR ID (that's the number after the `#` in the PR title) for the `<ID>` placeholders:

Listing 20-2

10. https://github.com/symfony/symfony/issues?utf8=%E2%9C%93&q=is%3Aopen+is%3Apr+label%3A%22Status%3A+Needs+Review%22+
11. https://symfony.com/roadmap
12. https://github.com/symfony/symfony-standard

```
$ cd vendor/symfony/symfony
$ git fetch origin pull/<ID>/head:pr<ID>
$ git checkout pr<ID>
```

For example:

Listing 20-3
```
$ git fetch origin pull/15723/head:pr15723
$ git checkout pr15723
```

Now you can *test the project* against the code in the PR.

6. **Update the PR Status**

   At last, add a comment to the PR. **Thank the contributor for working on the PR**. Include the line `Status: <status>` in your comment to trigger our *Carson Bot*[13] which updates the status label of the issue. You can set the status to one of the following:

   Needs Work If the PR is not yet ready to be merged, explain the issues that you found and move it to this status.

   Reviewed If the PR satisfies all the checks above, move it to this status. A core contributor will soon look at the PR and decide whether it can be merged or needs further work.

Example

Here is a sample comment for a PR that is not yet ready for merge:

Listing 20-4
```
Thank you @weaverryan for working on this! It seems that your test
cases don't cover the cases when the counter is zero or smaller.
Could you please add some tests for that?

Status: Needs Work
```

13. https://github.com/carsonbot/carsonbot

Chapter 21
Other Resources

In order to follow what is happening in the community you might find helpful these additional resources:

- List of open *pull requests*[1]
- List of recent *commits*[2]
- List of open *bugs and enhancements*[3]
- List of open source *bundles*[4]

1. https://github.com/symfony/symfony/pulls
2. https://github.com/symfony/symfony/commits/master
3. https://github.com/symfony/symfony/issues
4. https://github.com/search?q=topic%3Asymfony-bundle&type=Repositories