
 SymfonyCon	 Live	PARIS March 28-29	SÃO PAULO May 16-17	LONDON Sep. 13	NEW YORK Q4
AMSTERDAM Nov. 21-23	LILLE March 1	TUNIS April 27	WARSZAWA June 13-14	BERLIN Sep. 24-27	

Table of Contents

- [Creating Routes](#)
- [Localized Routing \(i18n\)](#)
- [Adding {wildcard} Requirements](#)
- [Giving {placeholders} a Default Value](#)
- [Listing all of your Routes](#)
- [Advanced Routing Example](#)
 - [Special Routing Parameters](#)
 - [Redirecting URLs with Trailing Slashes](#)
- [Controller Naming Pattern](#)
- [Generating URLs](#)

[Home](#) / [Documentation](#)

You are browsing the **Symfony 4 documentation**, which changes significantly from Symfony 3.x. If your app doesn't use Symfony 4 yet, browse the [Symfony 3.4 documentation](#).

Routing

4.2 version

[edit this page](#)

Beautiful URLs are a must for any serious web application. This means leaving behind ugly URLs like `index.php?article_id=57` in favor of something like `/read/intro-to-symfony`.

Having flexibility is even more important. What if you need to change the URL of a page from `/blog` to `/news`? How many links would you need to hunt down and update to make the change? If you're using Symfony's router, the change is simple.

Creating Routes ¶

A *route* is a map from a URL path to a controller. Suppose you want one route that matches `/blog` exactly and another more dynamic route that can match *any* URL like `/blog/my-post` or `/blog/all-about-symfony`.

Routes can be configured in YAML, XML and PHP. All formats provide the same features and performance, so choose the one you prefer. If you choose PHP annotations, run this command once in your app to add support for them:



```
$ composer require annotations
```

Now you can configure the routes:

Annotations

YAML

XML

PHP

```
1  // src/Controller/BlogController.php
2  namespace App\Controller;
3
4  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
5  use Symfony\Component\Routing\Annotation\Route;
6
7  class BlogController extends AbstractController
8  {
9      /**
10       * Matches /blog exactly
11       *
12       * @Route("/blog", name="blog_list")
13       */
14     public function list()
15     {
16         // ...
17     }
18
19     /**
20      * Matches /blog/*
21      *
22      * @Route("/blog/{slug}", name="blog_show")
23      */
24     public function show($slug)
25     {
26         // $slug will equal the dynamic part of the URL
27         // e.g. at /blog/yay-routing, then $slug='yay-routing'
28
29         // ...
30     }
31 }
```

Thanks to these two routes:

- If the user goes to `/blog`, the first route is matched and `list()` is executed;
- If the user goes to `/blog/*`, the second route is matched and `show()` is executed. Because the route path is `/blog/{slug}`, a `$slug` variable is passed to `show()` matching that value. For example, if the user goes to `/blog/yay-routing`, then `$slug` will equal `yay-routing`.

Whenever you have a `{placeholder}` in your route path, that portion becomes a wildcard: it matches *any* value. Your controller can now *also* have an argument called `$placeholder` (the wildcard and argument names *must* match).

Each route also has an internal name: `blog_list` and `blog_show`. These can be anything (as long as each is unique) and don't have any meaning yet. You'll use them later to [generate URLs](#).

Routing in Other Formats

The `@Route` above each method is called an *annotation*. If you'd rather configure your routes in YAML, XML or PHP, that's no problem! Create a new routing file (e.g. `routes.xml`) in the `config/` directory and Symfony will automatically use it.

Localized Routing (i18n) ¶

Routes can be localized to provide unique paths per *locale*. Symfony provides a handy way to declare localized routes without duplication.

Annotations

YAML

XML

PHP

```
1  // src/Controller/CompanyController.php
2  namespace App\Controller;
3
4  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
5  use Symfony\Component\Routing\Annotation\Route;
6
7  class CompanyController extends AbstractController
8  {
9      /**
10       * @Route({
11       *     "nl": "/over-ons",
12       *     "en": "/about-us"
13       * }, name="about_us")
14       */
15     public function about()
16     {
17         // ...
18     }
19 }
```

When a localized route is matched Symfony automatically knows which locale should be used during the request. Defining routes this way also eliminated the need for duplicate registration of routes which minimizes the risk for any bugs caused by definition inconsistency.



Tip

If the application uses full language + territory locales (e.g. `fr_FR`, `fr_BE`), you can use the language part only in your routes (e.g. `fr`). This prevents having to define multiple paths when you want to use the same route path for locales that share the same language.

A common requirement for internationalized applications is to prefix all routes with a locale. This can be done by defining a different prefix for each locale (and setting an empty prefix for your default locale if you prefer it):

YAML

XML

PHP

```
1  # config/routes/annotations.yaml
2  controllers:
```

```

3     resource: '../src/Controller/'
4     type: annotation
5     prefix:
6         en: ' ' # don't prefix URLs for English, the default locale
7         nl: '/nl'

```

Adding {wildcard} Requirements ¶

Imagine the `blog_list` route will contain a paginated list of blog posts, with URLs like `/blog/2` and `/blog/3` for pages 2 and 3. If you change the route's path to `/blog/{page}`, you'll have a problem:

- `blog_list`: `/blog/{page}` will match `/blog/*`;
- `blog_show`: `/blog/{slug}` will also match `/blog/*`.

When two routes match the same URL, the *first* route that's loaded wins. Unfortunately, that means that `/blog/yay-routing` will match the `blog_list`. No good!

To fix this, add a *requirement* that the `{page}` wildcard can *only* match numbers (digits):

Annotations

YAML

XML

PHP

```

1  // src/Controller/BlogController.php
2  namespace App\Controller;
3
4  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
5  use Symfony\Component\Routing\Annotation\Route;
6
7  class BlogController extends AbstractController
8  {
9      /**
10       * @Route("/blog/{page}", name="blog_list", requirements={"page"="\d+"})
11       */
12       public function list($page)
13       {
14           // ...
15       }
16
17       /**
18       * @Route("/blog/{slug}", name="blog_show")
19       */
20       public function show($slug)
21       {
22           // ...
23       }
24  }

```

The `\d+` is a regular expression that matches a *digit* of any length. Now:

URL	Route	Parameters
<code>/blog/2</code>	<code>blog_list</code>	<code>\$page = 2</code>

`/blog/yay-routing``blog_show``$slug = yay-routing`

If you prefer, requirements can be inlined in each placeholder using the syntax `{placeholder_name<requirements>}`. This feature makes configuration more concise, but it can decrease route readability when requirements are complex:

Annotations

YAML

XML

PHP

```
1  // src/Controller/BlogController.php
2  namespace App\Controller;
3
4  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
5  use Symfony\Component\Routing\Annotation\Route;
6
7  class BlogController extends AbstractController
8  {
9      /**
10       * @Route("/blog/{page<\d+>}", name="blog_list")
11       */
12       public function list($page)
13       {
14           // ...
15       }
16  }
```

To learn about other route requirements - like HTTP method, hostname and dynamic expressions - see [How to Define Route Requirements](#).

Giving {placeholders} a Default Value ¶

In the previous example, the `blog_list` has a path of `/blog/{page}`. If the user visits `/blog/1`, it will match. But if they visit `/blog`, it will **not** match. As soon as you add a `{placeholder}` to a route, it *must* have a value.

So how can you make `blog_list` once again match when the user visits `/blog`? By adding a *default* value:

Annotations

YAML

XML

PHP

```
1  // src/Controller/BlogController.php
2  namespace App\Controller;
3
4  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
5  use Symfony\Component\Routing\Annotation\Route;
6
7  class BlogController extends AbstractController
8  {
9      /**
10       * @Route("/blog/{page}", name="blog_list", requirements={"page"="\d+"})
11       */
12       public function list($page = 1)
13       {
14           // ...
15       }
16  }
```

```
15     }
16 }
```

If you want to always include some default value in the generated URL (for example to force the generation of `/blog/1` instead of `/blog` in the previous example) add the `!` character before the placeholder name: `/blog/{!page}`

New in version 4.3: The feature to force the inclusion of default values in generated URLs was introduced in Symfony 4.3.

As it happens with requirements, default values can also be inlined in each placeholder using the syntax `{placeholder_name?default_value}`. This feature is compatible with inlined requirements, so you can inline both in a single placeholder:

Annotations

YAML

XML

PHP

```
1  // src/Controller/BlogController.php
2  namespace App\Controller;
3
4  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
5  use Symfony\Component\Routing\Annotation\Route;
6
7  class BlogController extends AbstractController
8  {
9      /**
10       * @Route("/blog/{page<\d+>?1}", name="blog_list")
11       */
12       public function list($page)
13       {
14           // ...
15       }
16 }
```



Tip

To give a `null` default value to any placeholder, add nothing after the `?` character (e.g. `/blog/{page?}`).

Listing all of your Routes ¶

As your app grows, you'll eventually have a *lot* of routes! To see them all, run:

```
$ php bin/console debug:router
```

Name	Method	Path
app_lucky_number	ANY	/lucky/number/{max}

...

Advanced Routing Example ¶

With all of this in mind, check out this advanced example:

Annotations

YAML

XML

PHP

```
1  // src/Controller/ArticleController.php
2
3  // ...
4  class ArticleController extends AbstractController
5  {
6      /**
7       * @Route(
8       *     "/articles/{_locale}/{year}/{slug}.{_format}",
9       *     defaults={"_format": "html"},
10      *     requirements={
11      *         "_locale": "en|fr",
12      *         "_format": "html|rss",
13      *         "year": "\d+"
14      *     }
15      * )
16      */
17      public function show($_locale, $year, $slug)
18      {
19      }
20  }
```

As you've seen, this route will only match if the `{_locale}` portion of the URL is either `en` or `fr` and if the `{year}` is a number. This route also shows how you can use a dot between placeholders instead of a slash. URLs matching this route might look like:

- `/articles/en/2010/my-post`
- `/articles/fr/2010/my-post.rss`
- `/articles/en/2013/my-latest-post.html`

The Special `_format` Routing Parameter

This example also highlights the special `_format` routing parameter. When using this parameter, the matched value becomes the "request format" of the `Request` object.

Ultimately, the request format is used for such things as setting the `Content-Type` of the response (e.g. a `json` request format translates into a `Content-Type` of `application/json`).



Note

Sometimes you want to make certain parts of your routes globally configurable. Symfony provides you with a way to do this by leveraging service container parameters. Read more about this in "[How to Use Service Container Parameters in your Routes](#)".

Special Routing Parameters ¶

As you've seen, each routing parameter or default value is eventually available as an argument in the controller method. Additionally, there are four parameters that are special: each adds a unique piece of functionality inside your application:

`_controller`

As you've seen, this parameter is used to determine which controller is executed when the route is matched.

`_format`

Used to set the request format ([read more](#)).

`_fragment`

Used to set the fragment identifier, the optional last part of a URL that starts with a `#` character and is used to identify a portion of a document.

`_locale`

Used to set the locale on the request ([read more](#)).

Redirecting URLs with Trailing Slashes ¶

Historically, URLs have followed the UNIX convention of adding trailing slashes for directories (e.g. `https://example.com/foo/`) and removing them to refer to files (`https://example.com/foo`). Although serving different contents for both URLs is OK, nowadays it's common to treat both URLs as the same URL and redirect between them.

Symfony follows this logic to redirect between URLs with and without trailing slashes (but only for `GET` and `HEAD` requests):

Route path	If the requested URL is <code>/foo</code>	If the requested URL is <code>/foo/</code>
<code>/foo</code>	It matches (<code>200</code> status response)	It makes a <code>301</code> redirect to <code>/foo</code>
<code>/foo/</code>	It makes a <code>301</code> redirect to <code>/foo/</code>	It matches (<code>200</code> status response)

Note

If your application defines different routes for each path (`/foo` and `/foo/`) this automatic redirection doesn't take place and the right route is always matched.

Controller Naming Pattern ¶

The `_controller` value in your routes has the format `CONTROLLER_CLASS::METHOD`.

Tip

To refer to an action that is implemented as the `__invoke()` method of a controller class, you do not have to pass the method name, you can also use the fully qualified class name (e.g. `App\Controller\BlogController`).

Generating URLs ¶

The routing system can also generate URLs. In reality, routing is a bidirectional system: mapping the URL to a controller and also a route back to a URL.

To generate a URL, you need to specify the name of the route (e.g. `blog_show`) and any wildcards (e.g. `slug = my-blog-post`) used in the path for that route. With this information, an URL can be generated in a controller:

```
1  class MainController extends AbstractController
2  {
3      public function show($slug)
4      {
5          // ...
6
7          // /blog/my-blog-post
8          $url = $this->generateUrl(
9              'blog_show',
10             ['slug' => 'my-blog-post']
11         );
12     }
13 }
```

If you need to generate a URL from a service, type-hint the `UrlGeneratorInterface` service:

```
1  // src/Service/SomeService.php
2
3  use Symfony\Component\Routing\Generator\UrlGeneratorInterface;
4
5  class SomeService
6  {
7      private $router;
8
9      public function __construct(UrlGeneratorInterface $router)
10     {
11         $this->router = $router;
12     }
13
14     public function someMethod()
15     {
16         $url = $this->router->generate(
17             'blog_show',
18             ['slug' => 'my-blog-post']
19         );
20         // ...
21     }
22 }
```

Generating URLs with Query Strings ¶

The `generate()` method takes an array of wildcard values to generate the URI. But if you pass extra ones, they will be added to the URI as a query string:

```
1 $this->router->generate('blog', [  
2     'page' => 2,  
3     'category' => 'Symfony',  
4 ]);  
5 // /blog/2?category=Symfony
```

Generating Localized URLs ¶

When a route is localized, Symfony uses by default the current request locale to generate the URL. In order to generate the URL for a different locale you must pass the `__locale` in the parameters array:

```
$this->router->generate('about_us', [  
    '__locale' => 'nl',  
]);  
// generates: /over-ons
```

Generating URLs from a Template ¶

To generate URLs inside Twig, see the templating article: [Linking to Pages](#). If you also need to generate URLs in JavaScript, see [How to Generate Routing URLs in JavaScript](#).

Generating Absolute URLs ¶

By default, the router will generate relative URLs (e.g. `/blog`). From a controller, pass `UrlGeneratorInterface::ABSOLUTE_URL` to the third argument of the `generateUrl()` method:

```
use Symfony\Component\Routing\Generator\UrlGeneratorInterface;  
  
$this->generateUrl('blog_show', ['slug' => 'my-blog-post'], UrlGeneratorInterface::ABSOLUTE_URL);  
// http://www.example.com/blog/my-blog-post
```

Note

The host that's used when generating an absolute URL is automatically detected using the current `Request` object. When generating absolute URLs from outside the web context (for instance in a console command) this doesn't work. See [How to Generate URLs from the Console](#) to learn how to solve this problem.

Troubleshooting ¶

Here are some common errors you might see while working with routing:

Controller "App\Controller\BlogController::show()" requires that you provide a value for the "\$slug" argument.

This happens when your controller method has an argument (e.g. `$slug`):

```
public function show($slug)
{
    // ..
}
```

But your route path does *not* have a `{slug}` wildcard (e.g. it is `/blog/show`). Add a `{slug}` to your route path: `/blog/show/{slug}` or give the argument a default value (i.e. `$slug = null`).

Some mandatory parameters are missing ("slug") to generate a URL for route "blog_show".

This means that you're trying to generate a URL to the `blog_show` route but you are *not* passing a `slug` value (which is required, because it has a `{slug}`) wildcard in the route path. To fix this, pass a `slug` value when generating the route:

```
$this->generateUrl('blog_show', ['slug' => 'slug-value']);

// or, in Twig
// {{ path('blog_show', {'slug': 'slug-value'}) }}
```

Keep Going! ¶

Routing, check! Now, uncover the power of [controllers](#).

Learn more about Routing ¶

- [How to Restrict Route Matching through Conditions](#)
- [How to Create a custom Route Loader](#)
- [How to Visualize And Debug Routes](#)
- [How to Include External Routing Resources](#)
- [How to Pass Extra Information from a Route to a Controller](#)
- [How to Generate Routing URLs in JavaScript](#)
- [How to Match a Route Based on the Host](#)
- [How to Define Optional Placeholders](#)
- [How to Configure a Redirect without a custom Controller](#)
- [Redirect URLs with a Trailing Slash](#)
- [How to Define Route Requirements](#)
- [Looking up Routes from a Database: Symfony CMF DynamicRouter](#)
- [How to Force Routes to Always Use HTTPS or HTTP](#)
- [How to Use Service Container Parameters in your Routes](#)
- [How to Allow a "/" Character in a Route Parameter](#)

This work, including the code samples, is licensed under a [Creative Commons BY-SA 3.0 license](#).

Latest from the Symfony Blog

[Symfony 4.2.4 released](#)

March 3, 2019

[Symfony 3.4.23 released](#)

March 3, 2019

They Help Us Make Symfony



Thanks **Jordan Hoff** for being a Symfony contributor.

1 commit · 2 lines

Get Involved in the Community

A passionate group of over 600,000 developers from more than 120 countries, all committed to helping PHP surpass the impossible.

[Getting involved](#) →

Symfony™ is a trademark of Symfony SAS. All rights reserved.

What is Symfony?

- [Symfony at a Glance](#)
- [Symfony Components](#)
- [Case Studies](#)
- [Symfony Roadmap](#)
- [Security Policy](#)
- [Logo & Screenshots](#)
- [Trademark & Licenses](#)
- [symfony1 Legacy](#)

Screencasts

- [Learn Symfony](#)
- [Learn PHP](#)
- [Learn JavaScript](#)
- [Learn Drupal](#)
- [Learn RESTful APIs](#)

Blog

- [Events & Meetups](#)
- [A week of symfony](#)
- [Case studies](#)
- [Community](#)

Learn Symfony

- [Getting Started](#)
- [Components](#)
- [Best Practices](#)
- [Bundles](#)
- [Reference](#)
- [Training](#)
- [Certification](#)

Community

- [SymfonyConnect](#)
- [Support](#)
- [How to be Involved](#)
- [Events & Meetups](#)
- [Projects using Symfony](#)
- [Downloads Stats](#)
- [Contributors](#)

Services

- [Our services](#)
- [Train developers](#)
- [Manage your project quality](#)
- [Improve your project performance](#)

[Conferences](#)
[Diversity](#)
[Documentation](#)
[Living on the edge](#)
[Releases](#)
[Security Advisories](#)
[SymfonyInsight](#)

About
[SensioLabs](#)
[Careers](#)
[Support](#)

Follow Symfony



☒ Switch to dark theme