

Docs » Symfony Bundles » Content Bundle

Content Bundle

Publish Derafu content through Symfony routing, Twig and HTTP responses

Anonymous · 26/10/2026 · #php, #symfony, #content

Content Bundle

github [derafu/symfony-content-bundle](#)

CI  passing

downloads **1**

This bundle publishes [derafu/content](#) through the application's Symfony kernel. It uses Symfony's router, Twig environment, translator and UX Twig components. It does not start a Derafu HTTP kernel or create a second Twig environment.

Installation

The bundle requires PHP 8.5 and Symfony 8.1 components compatible with its Composer constraints.

```
composer require derafu/symfony-content-bundle
```

Derafu packages use `dev-main`. Applications that need to allow these versions can configure Composer with:

```
{
    "minimum-stability": "dev",
    "prefer-stable": true
}
```

Register the bundles in `config/bundles.php`, retaining the application's existing registrations:

```
return [
    Symfony\Bundle\FrameworkBundle\FrameworkBundle::class => ['all' => true],
    Symfony\Bundle\TwigBundle\TwigBundle::class => ['all' => true],
    Symfony\UX\TwigComponent\TwigComponentBundle::class => ['all' => true],
    Derafu\TranslationBundle\TranslationBundle::class => ['all' => true],
    Derafu\TwigComponentsBundle\TwigComponentsBundle::class => ['all' => true],
    Derafu\ContentBundle\ContentBundle::class => ['all' => true],
];
```

The [Translation Bundle](#) registers the package translation resources with Symfony's native translator. Keep the application's FrameworkBundle translation configuration enabled.

The [Twig Components Bundle](#) registers the reusable UI components, their template fallbacks and their translations. ContentBundle depends on that package and keeps only its Content-specific integration. Applications upgrading from the direct component registration must add `Derafu\TwigComponentsBundle\TwigComponentsBundle` to `config/bundles.php`.

Configuration

Configure the bundle in `config/packages/derafu_content.yaml`:

```
derafu_content:
  layout: base.html.twig
  content:
    title: My knowledge base
    url: https://example.com
    plugins:
      docs: ~
      pages: ~
      blog: ~
      sitemap: ~
      storage: ~
```

layout defaults to `base.html.twig`. The content map is passed to Content's own configuration, which validates its domain settings. Supplying `plugins` replaces the default plugin map. Configure `storage` when the site serves content attachments.

Without a plugin override, the package enables Academy, API, Blog, Docs, FAQ, Sitemap and Storage. Pages, Search and MCP require explicit configuration. Default content directories are relative to the Symfony project directory, such as `resources/content/docs` and `resources/content/pages`. Plugin-specific options follow the Content package's contracts.

Content's registry cache uses Symfony's `cache.app` pool. To disable registry caching:

```
derafu_content:
  content:
    cache:
      enabled: false
```

The bundle binds its renderer to Content's controllers. It does not replace an application's unrelated `RendererInterface` service. Applications can customize the Content renderer through the `derafu_content.renderer` service.

Import routes

Create `config/routes/derafu_content.yaml`:

```
derafu_content:
  resource: .
  type: derafu_content
```

A prefix is optional:

```
derafu_content:
  resource: .
  type: derafu_content
  prefix: /knowledge
```

The loader reads the installed package's `content-routes.yaml` and converts its requirements, defaults, methods and handlers into Symfony routes. Route names remain unchanged: for example, `docs`, `docs_doc`, `pages_page` and `blog_post`.

With the prefix above, a document is served at `/knowledge/docs/guide` and a nested page at `/knowledge/pages/team/member`. Symfony's native generation functions include the import prefix:

```
/docs/guide
/pages/team/member
```

Content items use the same Symfony URL generator for their links. Disabled plugins retain their route definitions; requesting their unavailable content returns 404. Route roles are retained as `_derafu_content_roles` metadata; access control belongs to the consuming application.

Routes registered by plugins

The loader initializes a separate, uncached Content context to collect routes registered through the existing `contentLoaded` action. These definitions become part of Symfony's compiled route collection. Initializing Content while serving a request does not publish new routes.

In development, content directories reported by plugins are registered as routing cache resources. In production, deploy changes to plugin route definitions with the normal cache rebuild:

```
php bin/console cache:clear --env=prod
php bin/console cache:warmup --env=prod
```

Adding a document under an existing route such as `/docs/{doc}` does not add a route definition.

Application layout and template overrides

Provide the application layout, for example `templates/base.html.twig`:

```
<!DOCTYPE html>
<html>
<head>
  <title>My website</title>
</head>
<body>
  </body>
</html>
```

The bundle supplies `layouts/default.html.twig`, extending the configured application layout and placing Content's content block inside `body`. Content templates can override `title`; the application's stylesheets remain inherited.

Application template paths take precedence over package fallbacks. Override a Content template by placing the corresponding file under `templates/`, for example `templates/pages/show.html.twig`. Applications can also replace `templates/layouts/default.html.twig` to define a different bridge.

Rendering shares the application's Twig globals, extensions, strict-variable setting and template cache. ContentBundle registers the Derafu Markdown and Twig filters and `is_active_path()`. Twig Components Bundle registers the UX components and their templates. Symfony retains its native `path()`, `url()` and translation integration.

```
<!-- TWIG_BLOCK_0 -->
```

Markdown content

Documentation section

Component styles and JavaScript are provided by the underlying packages and must be included through the application's asset setup. Rendering a component does not configure an asset pipeline.

HTTP formats and errors

The request bridge preserves method, URI, headers, body, query parameters, parsed form data, cookies, uploads and attributes in the concrete Derafu request expected by Content controllers. Content's existing format negotiation remains in effect.

Examples for a page named `about`:

Request	Response
<code>/pages/about</code>	HTML using the application layout
<code>/pages/about.json</code>	JSON with content and generated links

Request	Response
/pages/about.md	Markdown export
/pages/about.pdf	PDF export when mPDF is installed

Apply any configured route prefix to these paths. The existing full-export options of individual plugins remain available.

The response adapter reuses Derafu's `ResponseNormalizer` and Symfony's PSR HTTP-message bridge. It preserves PSR response status, headers, cookies and binary bodies; non-seekable bodies remain streamed. RSS conditional requests retain their 304 responses, and Sitemap keeps its XML response.

Content HTTP exceptions become Symfony HTTP exceptions with their status, headers and original cause. Search's invalid query input returns 400. Unrelated Symfony controllers and their responses remain under the application's own handling.

Optional integrations

The following packages are optional runtime dependencies:

Feature	Installation
PDF exports through mPDF	<code>composer require mpdf/mpdf</code>
MCP endpoint	<code>composer require mcp/sdk</code>
OpenAPI references rendered in document exports	<code>composer require utopia-php/openapi</code>

HTML, JSON and Markdown content without an OpenAPI reference work without these packages. An enabled MCP plugin does not require its SDK until the MCP action is requested. A missing optional dependency produces an error identifying it when the corresponding feature is used.

Search additionally requires its backend configuration and a compatible HTTP client, as described in the Content package documentation. Installing this bundle does not configure an external search or LLM service.

Verification

The package tests use real Symfony kernels, Content fixtures, Twig components, mPDF and the MCP protocol. They cover service visibility and aliases, effective translation tags, template overrides, prefixed routes, cached plugin routes, HTTP adaptation and response isolation.

Run the development checks with:

```
composer tests
```

```
composer phpstan  
composer phpcs
```

The separate runtime check verifies an installation without mPDF, MCP SDK or OpenAPI. Run it after installing only runtime dependencies:

```
composer install --no-dev  
composer tests-runtime
```

Restore development dependencies before running PHPUnit again.

Last updated on 26/10/2026