

Docs » Symfony Bundles » Twig Components Bundle

Twig Components Bundle

Register Derafu UI components, templates and translations in a Symfony application

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

Twig Components Bundle

github [derafu/symfony-twig-components-bundle](#)

CI passing

downloads **1**

This bundle integrates [derafu/twig-components](#) with an application's existing Symfony Twig environment. Symfony UX remains responsible for component creation, properties, attributes and rendering. The bundle registers the package's component services, template fallbacks and translation resources.

It does not require Content, the Derafu renderer or `derafu/twig`.

Installation

Requirements are PHP 8.5 with Intl and Symfony 8.1 components compatible with the Composer constraints.

```
composer require derafu/symfony-twig-components-bundle
```

Derafu dependencies use `dev-main`. If needed, allow development packages while preferring stable releases:

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

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

```
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],
];
```

Keep Symfony's translator enabled. For example:

```
# config/packages/translation.yaml
framework:
    default_locale: en
    translator:
        fallbacks: [en]
```

No `derafu_twig_components.yaml` configuration is required. The bundle resolves the installed library through Composer, so it does not assume the application uses the default vendor directory.

Use components

Use the existing component names and properties:

```
<twig:block-alert
    content="The operation completed successfully."
    type="success"
    dismissible
/>

<twig:block-card
    title="Documentation"
    content="Read the available guides."
    buttonText="Read more"
    buttonUrl="/docs"
/>
```

Symfony UX's native `component()` function also works:

```
{{ component('block-alert', {
    content: 'A reusable component',
    type: 'info'
}) }}
```

See the [component catalogue](#) for the available properties and examples. The integration shares the application's Twig extensions, globals, strict-variable setting and template cache. Application components continue to use the same Symfony UX factory.

Override templates

The package template directory is appended after the application's Twig paths. To override a component, create the matching template in the application:

```
{# templates/components/block-alert.html.twig #}  
<section class="application-alert" {{ attributes }}>  
    {{ this.content }}  
</section>
```

Other components continue to use the package's templates. No copied package templates are needed for components that the application does not override.

Translations

The [Translation Bundle](#) consumes the library's tagged translation provider. Component labels use the twig domain and their Spanish resources are supplied as ICU catalogues. Component exceptions use the `errors` domain, including their nested translated messages.

The application locale controls translation. Application catalogues retain priority, so a custom label can be provided in `translations/twig+intl-icu.es.yaml`:

```
Close: 'Close this notice'
```

Styles and JavaScript

The components use Bootstrap 5 classes, Font Awesome 6 icons and the CSS distributed by `derafu/twig-components`. Interactive Bootstrap components additionally need Bootstrap's JavaScript bundle.

Include these through the application's asset setup. The library's stylesheet is `resources/css/components.css`, which imports files from the adjacent `components/` directory. Preserve that directory structure when publishing or copying the styles.

The bundle does not install a frontend toolchain, add CDN tags or inject styles into an application's layout. AssetMapper, Encore and applications that publish static files can each integrate these resources using their existing process. See [component styles](#) for the files and requirements.

Verification

The tests boot a real Symfony kernel without Content or the Derafu Twig service. They verify the package YAML, private component services, all shipped component registrations, independent instances, translated labels and exceptions, application overrides and a real HTTP response. Rendering runs with strict Twig variables.

```
composer tests  
composer phpstan
```

composer phpcs

Last updated on 26/10/2026