Components

Components

Anonymous · 09/11/2026

Components

`Derafu\Twig\Service\ComponentRegistrar` registers Symfony UX Twig Components in the Twig environment, so a template can use the tags `<twig:...>` without Symfony. The environment does not have any component until it is given: without them, a tag `<twig:...>` is text of the template that Twig does not process, and it is in the output as it was written.

The UI components for websites are in `derafu/twig-components`; this page is about how the environment gets components, the ones of that package and the ones of your application.

A Component

A component is a class with the attribute `#[AsTwigComponent]`, and a template named `components/<name>.html.twig` in one of the paths of the environment:

```
use Symfony\UX\TwigComponent\Attribute\AsTwigComponent;

#[AsTwigComponent('alert')]
final class AlertComponent
{
    public string $content = '';

    public string $type = 'info';
}
```

```
{# templates/components/alert.html.twig #}
<div class="alert alert-{{ this.type }}">{{ this.content }}</div>
```

The component has the name of the attribute; if the attribute does not say one, it is the name of the class in lowercase. `#[AsTwigComponent('alert', template: 'other/alert.html.twig')]` has its own template, and `attributesVar:` the name of the variable of the attributes that the tag has and the component does not.

The methods marked with `#[PreMount]` and `#[PostMount]` work as in Symfony: the first changes the

data before they are set in the component, and the second runs after.

Giving the Components to the Environment

The `components` option has any of these:

```
use Derafu\Twig\Provider\ArrayComponentProvider;
use Derafu\Twig\Provider\ChainComponentProvider;
use Derafu\Twig\Provider\DirectoryComponentProvider;
use Derafu\Twig\Service\TwigService;

// The classes.
new TwigService(['components' => [AlertComponent::class]]);

// The instances.
new TwigService(['components' => [new AlertComponent()]]);

// Any iterable, such as the services that the container tags.
new TwigService(['components' => $iterable]);

// A provider.
new TwigService(['components' => new DirectoryComponentProvider(__DIR__ .
'/src/Component')]);
```

The providers implement `Derafu\Twig\Contract\ComponentProviderInterface`:

Provider	Components
`ArrayComponentProvider`	The classes that it receives.
`DirectoryComponentProvider`	The classes with `#[AsTwigComponent]` that are in a directory, in its subdirectories too.
`ChainComponentProvider`	The ones of several providers, without repeating them.
`IterableComponentProvider`	The classes from an iterable of component classes or instances, without repeating classes.

A file of the directory that is not readable is a translatable error that names the file.

With the Container

Import `resources/config/twig-services.yaml` and contribute a `ComponentProviderInterface` using the `derafu_twig.component_provider` tag. For example, a directory provider:

```
services:
    Derafu\Twig\Provider\DirectoryComponentProvider:
```

```
        arguments:
            $directory: '%kernel.project_dir%/src/Component'
        tags: ['derafu_twig.component_provider']
```

Or register `ArrayComponentProvider` with its `$components` constructor argument. Providers are collected in tag-priority order and evaluated when the environment is created. Their components are combined with any explicitly supplied PHP `components` option before registration.

Existing packages can also keep tagging individual component services:

```
services:
    App\Component\:
        resource: '../src/Component/'
        tags: ['derafu_twig.component']
```

The production YAML registers `IterableComponentProvider` to adapt that existing collection to the provider contract. It collects the component classes from the tagged instances, removing repeated classes; it does not preserve their instance state or inject their constructor dependencies into the component runtime. This matches the existing class-based component registrar. The services file of `derafu/twig-components` uses this individual component tag. No manual `components: !tagged_iterator` option is needed when using the package's default DI composition.

Tags must be explicit on services contributed by imported files; `_instanceof` rules do not propagate across those imports. An existing Symfony environment is left intact; its components are managed by the Symfony bundle integration.

The Attributes of a Component

The attributes that a tag has and the component does not have as properties are in `attributes`, and the template writes them as `{{ attributes }}`. They are HTML that is already escaped, so the environment marks `ComponentAttributes` as safe for the escaper.

The Cache

`ComponentRegistrar` reads the attributes and the methods of every component with reflection. That configuration is cached in a PSR-6 pool, and the key has a hash of the list of the components, so other components do not use it.

By default the pool is `Derafu\Twig\Cache\CacheItemPool`, a cache of files in `var/cache/derafu_twig` of the project. Give your own pool, any `Psr\Cache\CacheItemPoolInterface`, to the registrar or to the provider:

```
use Derafu\Twig\Provider\DirectoryComponentProvider;
use Derafu\Twig\Service\ComponentRegistrar;
```

```
use Derafu\Twig\Service\TwigCreator;
use Derafu\Twig\Service\TwigService;

$registrar = new ComponentRegistrar($pool);
$provider = new DirectoryComponentProvider(__DIR__ . '/src/Component', $pool);

$twig = new TwigService(new TwigCreator(['components' => $provider], $registrar));
```

Last updated on 09/11/2026